

How to write Variational Inference and Generative Models for Natural Language Processing

Yao Fu

YAO.FU@ED.AC.UK

*Institute for Language, Cognition and Computation
University of Edinburgh*

1. The Recipe

Variational inference and generative models are widely used in modern NLP. However, the terminologies used in the literature is not yet clear and standard. NLP people misuses many ML terms while ML people misuses many NLP terms. Our goal is to introduce a convention that seamlessly navigate through graphical models, neural networks, and computational linguistics. This note can be viewed as a recipe for writing VAE models in an NLP paper. Generally, to introduce a generative model, we follow the steps below:

1. Introduce the NLP task.
2. Identify the random variables. Identify the observed and the latent variables.
3. Propose a generative model.
4. Derive the posterior of the generative model, identify its computational complexity.
5. Propose an inference model to approximate the posterior.
6. Derive the ELBO objective.
7. (Optional) propose the regularization.
8. Propose the parameterization of the two models with neural networks.
9. Propose an optimization method for training.
10. (Optional) Inference with the two models.

2. Step-by-step Explanation

We use a neural language model with a discrete latent variable as our example. Notes, caveats, comments and related techniques are written with bullets in small Sans Serif font. Taking away these bullets, the main content can be viewed as a walk-through of the recipe.

The NLP task Given a dataset containing sentences of different latent classes, we want to infer the class of a given sentence.

- Note: In this part we should use strict NLP language.

Random variables A sentence $x = [x_1, \dots, x_n]$ is a sequence of words where each word $x_i \in \mathcal{V}$ is an instance of a discrete random variable. Denote n the length of the sentence and \mathcal{V} the vocabulary. The class of the sentence $z \in \mathcal{Z}$ is an instance of a discrete random variable and \mathcal{Z} is the set of all possible classes. We assume x is **observed** and z is **latent**.

- Note: this is the place where we map language concepts to math. After this mapping, we use strict probabilistic language.
- Caveat: we say "an **instance** of a r.v." rather than "a r.v.". Roughly, a random variable always means its value is not observed so it is like a placeholder not being filled in. An instance of a r.v. is a real valued number and is always observed and have a value. Little letters x, y, z usually mean instances while large letters X, Y, Z usually mean r.v.s.
- Note: by observed we mean we have it in the dataset and by latent we mean it is not in the dataset. We also call x in the dataset the **observations** or **data points**.

Generative Model We assume a generative model for x and z as:

$$p_\theta(x, z) \tag{1}$$

where θ is the model parameter. We assume the joint distribution factorizes as:

$$p_\theta(x, z) = p_\theta(z)p_\theta(x|z) \tag{2}$$

We further consider an autoregressive model for x as:

$$p_\theta(x|z) = \prod_t p_\theta(x_t|x_{1:t-1}, z) \tag{3}$$

So the full factorization of the generative model is:

$$p_\theta(x, z) = p_\theta(z)p_\theta(x|z) = p_\theta(z) \prod_t p_\theta(x_t|x_{1:t}, z) \tag{4}$$

- Definition: **A generative model is a joint distribution of all variables**. Just as is defined in equation 1.
- Note: We can decide how the joint probability factorize. The way we factorize the probability reveals the underlying data generation process. Here we assume we first generate the class of the sentence, then generate the sentence conditioned on the class.
- Note: **factorization, generative process, and directed graph** are three equivalent ways for representing a generative model. In above we use the factorization. For more discussions about the other two, see Bishop (2006), Chapter 8.
- Note: we can also factorize the model as $p(x, z) = p(x)p(z|x)$ which means we first generate a sentence then generate its class. This generative process makes less linguistic sense.
- Definition: **the prior is the marginal probability of the latent variable**. In this case, $p_\theta(z)$ is the prior.

- Note: The term $p_\theta(x|z)$ does not really have a name, so we call it the conditional probability of x given z .
- Caveat: some paper call $p_\theta(x|z)$ the generative model. This is wrong because it ignores the prior. The prior is part of the generative model.
- Note: many paper call $p_\theta(x|z)$ the *decoder*. This does not differentiate the probabilistic model and the **parameterization** of the model. Decoder is usually a neural network term used for parameterizing the model. One should say: we use an LSTM decoder/ transformer decoder to parameterize $p_\theta(x|z)$.
- Caveat: the prior is the marginal distribution of the latent variable. It represents our belief of a r.v. without seeing anything. Other distributions, especially distributions that conditioned on part of the observed variables, are NOT the prior because we have observed things. Many papers use the word "prior" in a wrong way.
- Note: we can set the prior fixed or learn the prior from the data. Learning the prior from the data is called **empirical Bayes**.
- Note: since the prior is our belief of the marginal distribution if the latent variable, it can be of any distribution at our choice. It does not need to be standard Gaussian.

Posterior From Bayes' rule, the **posterior** probability of z conditioned on x is:

$$p_\theta(z|x) = \frac{p_\theta(z, x)}{p_\theta(x)} = \frac{p_\theta(z, x)}{\sum_z p_\theta(z, x)} \quad (5)$$

- Caveat: after the definition of the generative model, it will directly induce a posterior with the Bayes' rule. This is to say, we do NOT define a posterior, we define a generative model and get the posterior from it.
- Note: we say the posterior is **tractable** if equation 5 can be calculated exactly, otherwise we say the posterior is **intractable**. In our case, if we only have a small number of classes, then our posterior is tractable since we can exactly calculate it.
- Note: the **intractable** case comes from when the summation in the denominator is too computationally intensive. Suppose in another model we have a latent variable $z_i \in \mathcal{Z}$ (similar to POS tag) for each x_i and we have a generative model that generate x_i and z_i interchangeably (Li and Rush, 2020):

$$p(x, z) = \prod_t^n p(z_t|x_{1:t-1}, z_{1:t-1})p(x_t|z_{1:t}, x_{1:t-1}) \quad (6)$$

Then the following summation is intractable because the space of z is too large:

$$p(x) = \sum_z \prod_t^n p(z_t|x_{1:t-1}, z_{1:t-1})p(x_t|z_{1:t}, x_{1:t-1}) \quad (7)$$

$$p(x) = \sum_{z_1} \sum_{z_2} \cdots \sum_{z_n} \prod_t^n p(z_t|x_{1:t-1}, z_{1:t-1})p(x_t|z_{1:t}, x_{1:t-1}) \quad (8)$$

where we have $n|\mathcal{V}|$ different sequences of z to sum over. Then the posterior $p(z|x)$ is intractable because $p(x)$ is intractable. This is where variational inference comes into play: we use a inference model to approximate the intractable the intractable posterior.

- Note: although $p(x)$ is also a marginal distribution, we do not call $p(x)$ the prior because is observed, we call it the **likelihood**.

Inference Model We assume a large set of possible classes, thus the exact computation of the posterior in equation 5 is expensive. So we use an inference model $q_\phi(z|x)$ to approximate the posterior.

- Note: in most of the cases, the posterior of the generative model $p_\theta(z|x)$ is intractable. Thus we use an inference model $q_\phi(z|x)$, also called the **variational posterior**, to approximate the posterior of the generative model. The inference model itself is a **discriminative model** since it is always conditioned on the observation x .
- Caveat: some paper call $q_\phi(z|x)$ the *encoder*. Again this mixes the model and the neural parameterization up. A better statement would be: we use an LSTM/ pretrained encoder to parameterize $q_\phi(z|x)$.
- Note: again, the inference model does not need to be a Gaussian. It can be of arbitrary choice.
- Caveat: when the inference model is chosen to be Gaussian, it does not necessarily imply that the posterior is also Gaussian – the posterior can be of any possible distribution. In this case, we are using a Gaussian distribution to approximate the intractable posterior.

The ELBO Objective We optimize the following ELBO objective:

$$\text{ELBO} = \mathbb{E}_{q_\phi(z|x)} \left[\log \frac{p_\theta(x, z)}{q_\phi(z|x)} \right] \quad (9)$$

$$= \mathbb{E}_{q_\phi(z|x)} \left[\log \frac{p_\theta(x|z)p_\theta(z)}{q_\phi(z|x)} \right] \quad (10)$$

$$= \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)] - \text{KL}[q_\phi(z|x) || p_\theta(z)] \quad (11)$$

- Note: in variational inference, we want the two objectively jointly:
 - Maximize the likelihood $p_\theta(x)$, as is down in maximum likelihood estimation.
 - Minimize the KL divergence of $q_\phi(z|x)$ and $p_\theta(z|x)$. This will ensure we get a good approximate.

The ELBO objective allow us to achieve these two goals jointly:

$$\log p_\theta(x) - \text{KL}[q_\phi(z|x) || p_\theta(z|x)] = \text{ELBO} = \mathbb{E}_{q_\phi(z|x)} \left[\log \frac{p_\theta(x, z)}{q_\phi(z|x)} \right] \quad (12)$$

When fixing θ , optimizing ϕ in the r.h.s is equivalent to making the KL term in the l.h.s. closer to 0. When fixing ϕ , optimizing θ in the r.h.s. will make (a). the likelihood term in the l.h.s. larger and the (b). the KL term in the l.h.s. closer to 0.

There are multiple ways to decompose the ELBO objective. Equation 11 is the common way. Another way is to keep the whole generative model:

$$\text{ELBO} = \mathbb{E}_{q_\phi(z|x)} \left[\log \frac{p_\theta(x, z)}{q_\phi(z|x)} \right] \quad (13)$$

$$= \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x, z)] + \mathcal{H}[q_\phi(z|x)] \quad (14)$$

where $\mathcal{H}[\cdot]$ denotes the entropy. This factorization is used in many NLP settings. A typical example is the unsupervised recurrent neural network grammars (Kim et al., 2019).

Parameterization We use a uniform prior for $p(z)$, and use an LSTM decoder for $p_\theta(x|z)$ with the embedding of z as the an additional input for each LSTM step:

$$h_t = \text{Dec}(h_{t-1}, e(x_{t-1}), e(z)) \quad (15)$$

$$p(x_t | x_{1:t-1}, z) = \text{softmax}(\text{FF}(h_t)) \quad (16)$$

where h_t is the hidden state at step t , $e(\cdot)$ denotes the embedding, $\text{Dec}(\cdot)$ denotes the decoder, $\text{FF}(\cdot)$ denotes a feed-forward network.

- Note: in this section we use the language of neural networks.
- Note: in this setting, θ denotes the set of parameters of the embedding, the decoder LSTM and the feed-forward layer.

For the inference network, we parameterize it with an LSTM encoder:

$$s = \text{Enc}([e(x_1), \dots, e(x_n)]) \quad (17)$$

$$q(z|x) = \text{softmax}(\text{FF}(s)) \quad (18)$$

where s is the representation of the sentence x and $\text{Enc}(\cdot)$ is the encoder.

- Note: in this setting, ϕ denotes the parameters of the LSTM encoder.

Optimization We apply gradient based optimization to θ and ϕ . The **Monte-Carlo (MC) gradient** (Mohamed et al., 2019) for θ can be directly estimated with:

$$\nabla_{\theta} \text{ELBO} = \nabla_{\theta} \mathbb{E}_{q_{\phi}(z|x)} [\log \frac{p_{\theta}(x, z)}{q_{\phi}(z|x)}] \quad (19)$$

$$= \mathbb{E}_{q_{\phi}(z|x)} [\nabla_{\theta} \log p_{\theta}(x, z)] \quad (20)$$

- Technique: **Monte-Carlo integral for expectations**. Suppose one want to know the value of an expectation

$$\mathbb{E}_{p(x)}[f(x)] \quad (21)$$

with the following setting: (a). it is difficult to analytically compute such value; (b) it is easy to get samples x^1, \dots, x^N from $p(x)$. Then one can approximate the expectation with:

$$\mathbb{E}_{p(x)}[f(x)] \approx \frac{1}{N} \sum_{i=1}^N f(x^i) \quad (22)$$

Plugging this to equation 20, we have: (a). direct calculation of equation 20 is computationally expensive since we need to enumerate all possible z (yet it is indeed tractable), (b). it is easy to sample from $q_{\phi}(z|x)$. So we get a sample z^1, \dots, z^N from $q_{\phi}(z|x)$ and approximate the gradient with:

$$\nabla_{\theta} \text{ELBO} = \mathbb{E}_{q_{\phi}(z|x)} [\nabla_{\theta} \log p_{\theta}(x, z)] \quad (23)$$

$$\approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log p_{\theta}(x, z^i) \quad (24)$$

Gradients estimated with this technique is called the Monte-Carlo gradient. In practice, one can use as small as one single sample for computational efficiency.

For the MC gradient of the inference network, we consider two gradient estimators: (a). **score function gradient estimator** (Ranganath et al., 2013) (REINFORCE); (b). **reparameterized gradient estimator** with Gumbel-Softmax. The score function estimator

(with baseline function $b(x)$) takes the following form:

$$\nabla_{\phi} \text{ELBO} = \nabla_{\phi} \mathbb{E}_{q_{\phi}(z|x)} \left[\log \frac{p_{\theta}(x, z)}{q_{\phi}(z|x)} \right] \quad (25)$$

$$= \nabla_{\phi} \mathbb{E}_{q_{\phi}(z|x)} [\log p_{\theta}(x, z)] + \nabla_{\phi} \mathcal{H}[q_{\phi}(z|x)] \quad (26)$$

$$= \mathbb{E}_{q_{\phi}(z|x)} [(\log p_{\theta}(x, z) - b(x)) \nabla_{\phi} \log q_{\phi}(z|x)] + \nabla_{\phi} \mathcal{H}[q_{\phi}(z|x)] \quad (27)$$

- Note: pay attention to the differences between equation 20 and 27. In equation 20 the expectation is taking over q_{ϕ} , which does not involve θ , so the gradient operation can go inside the expectation. In equation 25, since we are taking the gradient of ϕ , which is involved in q_{ϕ} with which the expectation is taking over. So the gradient operation cannot directly go inside the expectation.
- Caveat: the baseline function $b(x)$ a function solely of x and should be **independent of** the sample z .
- Note: the gradient of the entropy term can be calculated analytically. It is the gradient of the log likelihood term that is difficult to estimate.

For the reparameterized estimator, denote $\tilde{q}_{\phi, \tau}(\tilde{z}|x)$ as the Gumbel-Softmax distribution (or the Concrete distribution Maddison et al., 2016) induced from the discrete distribution $q_{\phi}(z|x)$. We use \tilde{z} as the continuous relaxation of the discrete z , and approximate equation 25 with:

$$\nabla_{\phi} \text{ELBO} = \nabla_{\phi} \mathbb{E}_{q_{\phi}(z|x)} \left[\log \frac{p_{\theta}(x, z)}{q_{\phi}(z|x)} \right] \quad (28)$$

$$\approx \mathbb{E}_{\epsilon \sim \text{Gumbel}(0,1)} [\nabla_{\phi} \log p_{\theta}(x, \tilde{z}(\phi, \tau, \epsilon)) - b(x)] + \nabla_{\phi} \mathcal{H}[q_{\phi}(z|x)] \quad (29)$$

where $\tilde{z} \rightarrow z$ as $\tau \rightarrow 0$.

- Note: we view the Gumbel-Softmax as a distribution of a continuous random variable \tilde{z} used for approximating the target discrete variable z . The distribution for \tilde{z} is written as $\tilde{q}_{\phi, \tau}(\tilde{z}|x)$. The subscripts ϕ, τ mean that it shares the same parameter ϕ with the target discrete distribution $q_{\phi}(z|x)$.
- **Continuous Relaxation** means we approximate the discrete z with the continuous \tilde{z} . **Reparameterization** (Kingma and Welling, 2013) means we transform the sample operation of \tilde{z} as a function of the model parameter ϕ and some random noise ϵ . Reparameterization allows us to view the sample \tilde{z} as a function of the model parameter ϕ , the temperature τ , and the gumbel noise ϵ , written as $\tilde{z}(\phi, \tau, \epsilon)$. Then the numerator inside the expected log in equation 28 can be re-written as:

$$\nabla_{\phi} \mathbb{E}_{q_{\phi}(z|x)} [\log p_{\theta}(x, z)] \quad (30)$$

$$\approx \nabla_{\phi} \mathbb{E}_{\tilde{q}_{\phi}(\tilde{z}|x)} [\log p_{\theta}(x, \tilde{z})] \quad \# \text{ continuous relaxation} \quad (31)$$

$$\approx \nabla_{\phi} \mathbb{E}_{\epsilon \sim \text{Gumbel}(0,1)} [\log p_{\theta}(x, \tilde{z}(\phi, \tau, \epsilon))] \quad \# \text{ reparameterization} \quad (32)$$

$$\approx \mathbb{E}_{\epsilon \sim \text{Gumbel}(0,1)} [\nabla_{\phi} \log p_{\theta}(x, \tilde{z}(\phi, \tau, \epsilon))] \quad \# \text{ move the grad operator inside expectation} \quad (33)$$

Inference We consider the following inference operations: (a). sampling and conditional generation, (b). inferring the latent with the variational distribution, (c). exact likelihood evaluation, (d). likelihood estimation with **importance sampling** (Dyer et al., 2016).

- Generally, "Inference" can mean a lot of things with a probabilistic model, and it means differently when it is used in the word "inference model". When one say "Inference" with a probabilistic model, one usually view a probabilistic model as a class (in a programming sense) and inference as a set of member functions defined within that class. This typically include: sampling, likelihood evaluation, find the instance with maximum probability (mode), calculate entropy, .etc.

Sampling and conditional generation Given a trained model, we can sample new data from the model following its generative process by first sample a z from the prior $p_\theta(z)$ then sample an x from $p_\theta(x|z)$. Furthermore, for a given z^* , we can generate its most representative x^* with:

$$x^* = \arg \max_x p_\theta(x|z^*) \quad (34)$$

As $\arg \max_x p_\theta(x|z^*)$ is intractable for an autoregressive model, we approximate it with beam search.

Inferring the latent Given a trained model and a sentence x^* , we can infer the most probable z^* for x^* by taking:

$$z^* = \arg \max_{z \in \mathcal{Z}} q_\phi(z|x^*) \quad (35)$$

Exact likelihood evaluation Given a trained model and a sentence x^* , we can evaluate the likelihood of x^* with:

$$p_\theta(x^*) = \sum_{z \in \mathcal{Z}} p_\theta(z, x^*) \quad (36)$$

- This is tractable only when the space of z is small. Usually we use importance sampling rather than exact marginalization.

Likelihood estimation with importance sampling When the space of the latent variable is too large to enumerate, we can approximate equation 36 with importance sampling using $q_\phi(z|x)$ as the proposal distribution:

$$p_\theta(x^*) = \sum_{z \in \mathcal{Z}} p_\theta(z, x^*) \quad (37)$$

$$= \sum_{z \in \mathcal{Z}} q_\phi(z|x) \cdot \frac{p_\theta(z, x^*)}{q_\phi(z|x)} \quad (38)$$

$$= \mathbb{E}_{q_\phi(z|x)} \left[\frac{p_\theta(z, x^*)}{q_\phi(z|x)} \right] \quad (39)$$

- Again, equation 39 can be estimated with Monte-Carlo integral in equation 22.

3. Examples of Generative Models

URNNG is a generative model that jointly generate a sentence and the parse tree over the sentence (Kim et al., 2019). The generative model is a transition-based shift-reduce generation system for joint parsing and generation. The inference model is a constituency Tree-CRF. They use the second form of the ELBO (equation 14), making it NOT a VAE. They use the score function estimator for optimization, and their baseline is a simplified version of VIMCO (Mnih and Rezende, 2016).

Differentiable Perturb and Parse is a generative model that firstly generate a tree then generate a sentence conditioned on the tree (Corro and Titov, 2018). The inference model is a dependency parser. The difference of this model with URNNG is that while URNNG generates a partial tree and a partial sentence in an interleaved fashion, this model generate a whole tree before generating the sentence. In their case, it is possible to decompose the generative model as a product of the prior and the conditional, so they use the first form of ELBO (equation 11), making their model a VAE. They further put partial supervision to the inference model, making the setting semi-supervised learning. Instead of using the score function estimator, they use the reparameterized estimator with continuous relaxation. One of their contribution is to relax the Eisner’s algorithm. It will be helpful to compare this model with the URNNG model.

4. Further References

It is suggested to start from Bishop (2006) Chapter 8.1, then Chapter 10 for the introduction of VI and directed graphs. Then Blei et al. (2017) for a detailed understanding of VI. Then Columbia DGM seminar (Cunningham) for in-depth study of generative models in the neural network era. Finally, DGM4NLG (Fu, 2018) gives a complete landscape of deep generative model for NLP. In the mean time, keep a statistics text book like Wasserman (2010) and a graphical model text book like Bishop (2006); Murphy (2012) would always be helpful.

References

- Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.
- David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.
- Caio Corro and Ivan Titov. Differentiable perturb-and-parse: Semi-supervised parsing with a structured variational autoencoder. *arXiv preprint arXiv:1807.09875*, 2018.
- John Cunningham. Deep generative models. URL <http://stat.columbia.edu/~cunningham/teaching/GR8201/>.
- Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776*, 2016.
- Yao Fu. Deep generative models for natural language processing. 2018. URL <https://github.com/FranxYao/Deep-Generative-Models-for-Natural-Language-Processing>.
- Yoon Kim, Alexander M Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. Unsupervised recurrent neural network grammars. *arXiv preprint arXiv:1904.03746*, 2019.
- Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- Xiang Lisa Li and Alexander M Rush. Posterior control of blackbox generation. *arXiv preprint arXiv:2005.04560*, 2020.
- Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.
- Andriy Mnih and Danilo Rezende. Variational inference for monte carlo objectives. In *International Conference on Machine Learning*, pages 2188–2196. PMLR, 2016.
- Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *arXiv preprint arXiv:1906.10652*, 2019.
- Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020.
- Rajesh Ranganath, Sean Gerrish, and David M Blei. Black box variational inference. *arXiv preprint arXiv:1401.0118*, 2013.
- Larry Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer Publishing Company, Incorporated, 2010. ISBN 1441923225.