Automatic variational inference in probabilistic programming

Taku Yoshioka

Abstract

Probabilistic programming (PP) allows us to infer beliefs for unobservable events, represented as stochastic variables of probabilistic models. PPs have rely on Markov chain Monte Carlo (MCMC), however, MCMC is not efficient in the problems involving many (over thousands) variables. Recently, an automation procedure for variational inference, automatic differentiation variational inference (ADVI), has been proposed as an alternative to MCMC. ADVI has been implemented in PyMC3, a python library for PP. In this presentation, I will show the theory of ADVI and an application of PyMC3's ADVI on probabilistic models.

PP and PyMC3

- PPs allows us to write probabilistic generative models and infer unknown stochastic variables in the model.
- In PyMC3, probabilistic models is written as Python code.

Generative model

$p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$

$\mathbf{x}$: data

$\mathbf{z}$: unknown variables

Inference of posterior

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

Posterior distribution of unknown variables

Example: Gaussian mixture model (GMM)

Generative model of GMM

Prior $p(\mathbf{z})$

$$p(\pi) = \mathrm{Dir}(0.1, \cdots, 0.1)$$

$$p(\mathbf{m}_k) = N(0, \mathbf{I}_d)$$

Likelihood $p(\mathbf{x}|\mathbf{z})$

$$p(\mathbf{x}|\pi, \{\mathbf{m}_k\}_{k=1}^K) = \sum_{k=1}^K \pi_k N(\mathbf{x}|\mathbf{m}_k)$$

PyMC3 code

```python
def logp_gmix(mus, pi, tau):
    def logp_(value):
        logps = [tt.log(pi[i]) + logp_normal(mu, tau, value)
                 for i, mu in enumerate(mus)]

        return tt.sum(LogSumExp(tt.stacklists(logps)[:, :n_samples], axis=0))

    return logp_

with pm.Model() as model:
    mus = [MvNormal('mu_%d' % i, mu=np.zeros(2), tau=0.1 * np.eye(2), shape=(2,))
           for i in range(2)]
    pi = Dirichlet('pi', a=0.1 * np.ones(2), shape=(2,))
    xs = DensityDist('x', logp_gmix(mus, pi, np.eye(2)), observed=data)
```

Result

Green: samples in data
Red and blue: posterior distribution of $\mathbf{m}_k$ with precision proportional to the numbers of samples in each cluster

Automatic differentiation variational inference (ADVI)

Variational inference

Goal: minimize distance between variational posterior $q_\theta(\mathbf{z})$ and true posterior $p(\mathbf{z}|\mathbf{x})$ wrt parameter θ

- Distance: KL-divergence

$$KL(q_\theta(\mathbf{z})||p(\mathbf{z}|\mathbf{x}))$$

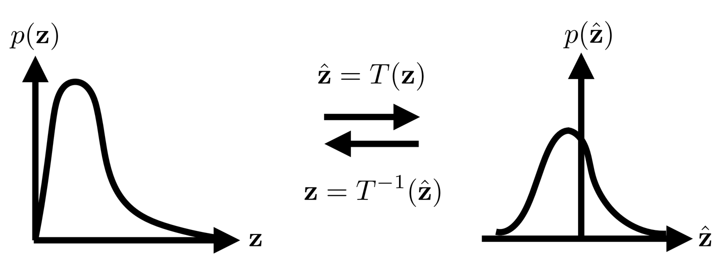
- Evicence lower bound (ELBO)

$$L[q_\theta(\mathbf{z})] \equiv E_{q_\theta(\mathbf{z})}[\log p(\mathbf{x}, \mathbf{z}) - \log q_\theta(\mathbf{z})]$$
$$= \log p(\mathbf{x}) - KL(q_\theta(\mathbf{z})||p(\mathbf{z}|\mathbf{x}))$$

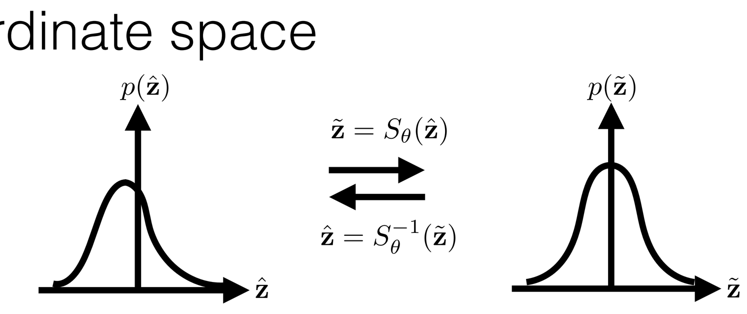
Since $\log p(\mathbf{x})$ is constant wrt θ, larger $L[q_\theta(\mathbf{z})]$, lower the distance $KL(q_\theta(\mathbf{z})||p(\mathbf{z}|\mathbf{x}))$

Variable transformation

1. From original constrained space (e.g., positive values or simplex) to (unconstrained) real coordinate space



2. From standardized space to real coordinate space



Parametrized distribution in the expectation of ELBO can be replaced with a fixed distribution, allowing to compute an accurate (low variance) stochastic gradient

$$\begin{aligned} L[q_\theta(\mathbf{z})] \;=\; & E_{N_s(\tilde{\mathbf{z}})}\left[\log p(\mathbf{x}, T^{-1}(S_\theta^{-1}(\tilde{\mathbf{z}}))\right] + \\ & E_{N_s(\tilde{\mathbf{z}})}\left[\log|\det J_{T^{-1}}(S_\theta^{-1}(\tilde{\mathbf{z}}))|\right] + \\ & H\left[q_\theta(\hat{\mathbf{z}})\right] \end{aligned}$$

Stochastic gradient

Monte Carlo sampling

$$\begin{aligned} \nabla_\theta L[q_\theta(\mathbf{z})] &= \nabla_\theta E_{q(\tilde{z})}[f_\theta(\tilde{z})] \\ &= E_{q(\tilde{z})}[\nabla_\theta f_\theta(\tilde{z})] \\ &\sim M^{-1}\textstyle\sum_{m=1}^M \nabla_\theta f_\theta(\tilde{z}^{(m)}) \end{aligned}$$

Example: latent dirichlet allocation (LDA) with variational autoencoder

Generative model of LDA

Word distribution of k-th topic

$$p(\beta_k) = \mathrm{Dir}(\beta_k|\gamma)$$

Topic distribution of i-th doc

$$p(\pi_i) = \mathrm{Dir}(\pi_i|\alpha)$$

Probability of words in i-th doc

$$p(\mathbf{x}_i|\pi_i, \beta) = \sum_{k=1}^K \pi_{i,k}\mathrm{Mult}(\mathbf{x}_i|\beta_k)$$

- Document $\mathbf{x}_i$ as bag-of-words: set of number of times of appearance of each word
- Word probability following a mixture of multinomials with *sample-dependent* mixing proportions

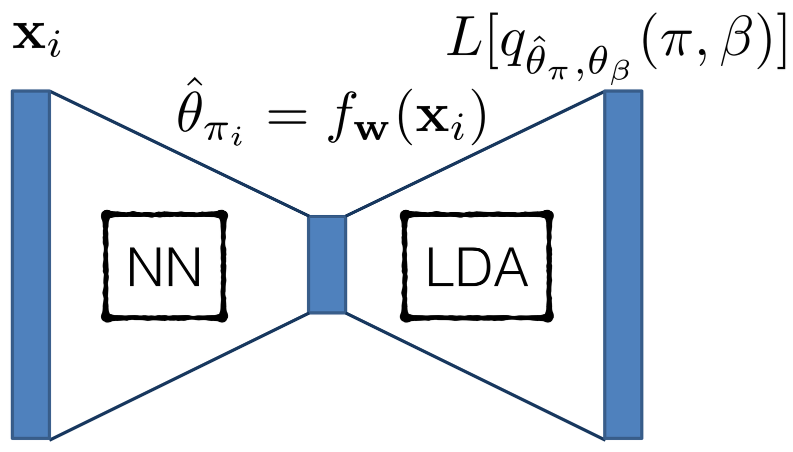
PyMC3 code

```python
with pm.Model() as model:
    pi = Dirichlet('pi', a=(1.0 / n_topics) * np.ones((minibatch_size, n_topics)),
                   shape=(minibatch_size, n_topics), transform=t_stick_breaking)
    beta = Dirichlet('beta', a=(1.0 / n_topics) * np.ones((n_topics, n_words)),
                     shape=(n_topics, n_words), transform=t_stick_breaking)
    doc = pm.DensityDist('doc', logp_lda_doc(beta, pi), observed=doc_t)


def logp_lda_doc(beta, pi):
    def ll_docs_f(docs):
        dixs, vixs = docs.nonzero()
        vfreqs = docs[dixs, vixs]
        ll_docs = vfreqs * pm.math.logsumexp(
            tt.log(pi[dixs]) + tt.log(beta.T[vixs]), axis=1).ravel()

        # Per-word log-likelihood times num of tokens in the whole dataset
        return tt.sum(ll_docs) / tt.sum(vfreqs) * n_tokens

    return ll_docs_f
```



Variational autoencoder

Unknown variables:

π_i : Depend on each sample

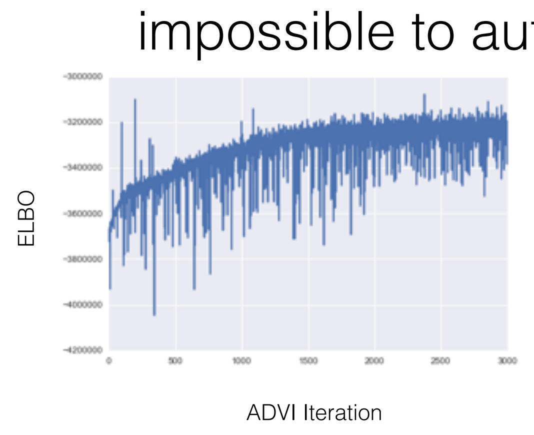
β_k : Depend on the model

```python
def encode(self, xs):
    w0 = self.w0.reshape(
        (self.n_words, self.n_hidden)
    )
    w1 = self.w1.reshape(
        (self.n_hidden, 2 * (self.n_topics - 1))
    )
    hs = tt.tanh(xs.dot(w0) + self.b0)
    zs = hs.dot(w1) + self.b1
    zs_mean = zs[:, :(self.n_topics - 1)]
    zs_std = zs[:, (self.n_topics - 1):]
    return zs_mean, zs_std
```

- Instead of estimating θ_{π_i} for each doc, estimate parameters of NN $\mathbf{w}$ which computes $\hat{\theta}_{\pi_i}$ given a sample $\mathbf{x}_i$: $\hat{\theta}_{\pi_i} = f_\mathbf{w}(\mathbf{x}_i)$ (PyMC3 code above)
- θ_β and $\mathbf{w}$ are simultaneously optimized

Results

- Estimate posterior distribution of 10,000 parameters impossible to automate with MCMC




```
Topic #0: don think just know make going like people want sure
Topic #1: year team game play win games players season period new
Topic #2: edu information com mail list send available university 1993 email
Topic #3: people state government gun world said years war states armenian
Topic #4: god people believe does true jesus say question life way
Topic #5: windows use thanks drive using window card file does work
Topic #6: key use chip encryption government public keys used law clipper
Topic #7: did time didn got said just day right thought let
Topic #8: good like ve better really car probably lot know problem
Topic #9: new power space 10 00 years used price 50 high
```

Summary

With automatic Bayesian inference, probabilistic programming allows us to estimate posterior distribution on high dimensional parameter space, which is impossible to automate with MCMC. Almost arbitrary probabilistic models can be applied. In addition, variational autoencoder can be incorporated when variational posterior is defined for latent variables corresponding to each sample in data. By using PyMC3, the model (and NN for autoencoding) is written as a Python code with a natural syntax. Users do not need to learn modelling languages specific to the library.