VAE with Normalizing Flow

Zhenya Liu

Overview

1. VAE
2. VAE with Normalizing Flow
3. Experiment

Variational Autoencoder

Recall

Variational Autoencoder(VAE)(Kingma & Welling, 2014) is a generative model using neural networks to train on ELBO.

Variational Autoencoder

$$\begin{aligned}\log P(\mathbf{X}) &= D_{\mathrm{KL}}(Q(\mathbf{Z}) \| P(\mathbf{Z}|\mathbf{X})) - \mathbb{E}_{\mathbf{Q}}[\log Q(\mathbf{Z}) - \log P(\mathbf{Z}, \mathbf{X})] \\ &= D_{\mathrm{KL}}(Q(\mathbf{Z}) \| P(\mathbf{Z}|\mathbf{X})) + \mathcal{L}(Q)\end{aligned}$$

Here $\mathcal{L}(Q) = -\mathbb{E}_{\mathbf{Q}}[\log Q(\mathbf{Z}) - \log P(\mathbf{Z}, \mathbf{X})]$ is known as the **Evidence Lower Bound** (ELBO) of $\log P(\mathbf{X})$.

Variational Autoencoder

Construct neural networks $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ (*probabilistic decoder*) for likelihood distribution and $q_\phi(\mathbf{z}|\mathbf{x})$ (*probabilistic encoder*) for marginal distribution.

Objective: Maximize $\log p_{\boldsymbol{\theta}}(\mathbf{x})$, and approximate $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ by $q_\phi(\mathbf{z}|\mathbf{x})$.

Recall:
Maximize $\mathcal{L}(Q) \implies$ maximize $\log P(\mathbf{X})$ and minimize $D_{\mathrm{KL}}(Q(\mathbf{Z})|P(\mathbf{Z}|\mathbf{X}))$.

so the **ELBO** is a sufficient objective for jointly training $\boldsymbol{\theta}, \boldsymbol{\phi}$.

Variational Autoencoder

Marginal log likelihood for individual datapoint
$\log p_{\boldsymbol{\theta}}\left(\mathrm{x}^{(1)}, \cdots, \mathrm{x}^{(N)}\right)=\sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}\left(\mathrm{x}^{(i)}\right)$,

$$\log p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) \geq \mathcal{L}\left(\boldsymbol{\theta}, \boldsymbol{\phi} ; \mathbf{x}^{(i)}\right)=\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})}\left[-\log q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})+\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\right]$$

$$\mathcal{L}\left(\boldsymbol{\theta}, \boldsymbol{\phi} ; \mathbf{x}^{(i)}\right)=-D_{KL}\left(q_{\boldsymbol{\phi}}\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right) \| p_{\boldsymbol{\theta}}(\mathbf{z})\right)+\mathbb{E}_{q_{\boldsymbol{\phi}}\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right)}\left[\log p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)} \mid \mathbf{z}\right)\right]$$

The KL divergence term on **RHS** can always be analytically calculated, but the expectation term (reconstruction error) can be approximated by Monte Carlo sampling with reparameterization techniques.

VAE with Normalizing Flow

In VAE, $q_\phi(\mathbf{z} \mid \mathbf{x})$ is defined as factorial gaussian distribution. This may not be a good approximation to the true posterier distribution.

By constructing normalizing flows, we can finally output a flexible distribution which could be a better approximation to the posterior. We embed VAE with learnable normalizing flows.

VAE with Normalizing Flow

Recall the change of variables in normalizing flows:
Given initial distribution $\mathbf{z}_0$, and a sequence of bijective transformations, then we can get the final distribution $\mathbf{z}_K$,

$$\mathbf{z}_K = f_K \circ \ldots \circ f_2 \circ f_1 \left(\mathbf{z}_0\right)$$

$$\log q_K \left(\mathbf{z}_K\right) = \log q_0 \left(\mathbf{z}_0\right) - \sum_{k=1}^{K} \log \left|\det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right|$$

VAE with Normalizing Flow

We consider the **planar** flows for all transformation,

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h\left(\mathbf{w}^\top \mathbf{z} + b\right)$$

where $\lambda = \left\{\mathbf{w} \in \mathbb{R}^D, \mathbf{u} \in \mathbb{R}^D, b \in \mathbb{R}\right\}$ are free parameters, $h(\cdot)$ is a smooth element-wise non-linear function with derivative h'.

VAE with Normalizing Flow

From planar flows, we can calculate the jacobian determinant in the linear time $w.r.t$ dimension of $\mathbf{z}$,

$$\psi(\mathbf{z}) = h'\left(\mathbf{w}^\top \mathbf{z} + b\right)\mathbf{w}$$

$$\left|\det \frac{\partial f}{\partial \mathbf{z}}\right| = \left|\det\left(\mathbf{I} + \mathbf{u}\psi(\mathbf{z})^\top\right)\right| = \left|1 + \mathbf{u}^\top \psi(\mathbf{z})\right|$$

The last equality is about calculating the determinant of $\mathbf{I}$ plus rank 1 matrix, and

the result achieved by **Matrix Determinant Lemma**.

Matrix Determinant Lemma

Since

$$\begin{pmatrix} \mathbf{I} & 0 \\ \mathbf{v}^\top & 1 \end{pmatrix} \begin{pmatrix} \mathbf{I} + \mathbf{u}\mathbf{v}^\top & \mathbf{u} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{I} & 0 \\ -\mathbf{v}^\top & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{u} \\ 0 & 1 + \mathbf{v}^\top \mathbf{u} \end{pmatrix}$$

Caculating the determinant of both sides,

$$\det\left(\mathbf{I} + \mathbf{u}\mathbf{v}^\top\right) = \left(1 + \mathbf{v}^\top \mathbf{u}\right)$$

VAE with Normalizing Flow

Then we can conclude our normalizing flow

$$\mathbf{z}_K = f_K \circ f_{K-1} \circ \ldots \circ f_1(\mathbf{z})$$

$$\log q_K\left(\mathbf{z}_K\right) = \log q_0(\mathbf{z}) - \sum_{k=1}^{K} \log \left|1 + \mathbf{u}_k^\top \psi_k\left(\mathbf{z}_{k-1}\right)\right|$$

Remark

The planar flow f may not be invertible, but we have techniques to enforce the invertibility during the implementation.

For instance, when using $h(x) = \tanh(x)$, one sufficient condition for $f(\mathbf{z})$ to be invertible is $\mathbf{w}^T\mathbf{u} \geq -1$.

In practice, for arbitrary $\mathbf{u}$ and $\mathbf{w}$, we will calculate a new $\hat{\mathbf{u}}$,
$\hat{\mathbf{u}}(\mathbf{w}, \mathbf{u}) = \mathbf{u} + \left[m\left(\mathbf{w}^\top\mathbf{u}\right) - \left(\mathbf{w}^\top\mathbf{u}\right)\right]\frac{\mathbf{w}}{\|\mathbf{w}\|^2}$, where $m(x) = -1 + \log\left(1 + e^x\right)$.
so that $\mathbf{w}^T\hat{\mathbf{u}} \geq -1$.

VAE with Normalizing Flow

Back to VAE, we define the encoder $q_\phi(\mathbf{z} \mid \mathbf{x}) := q_K(\mathbf{z}_K)$, and get the negative ELBO as

$$
\begin{aligned}
\mathcal{F}(\mathbf{x}) = -\mathcal{L}(Q) &= \mathbb{E}_{q_\phi(z|x)}\left[\log q_\phi(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{x}, \mathbf{z})\right] \\
&= \mathbb{E}_{q_0(z_0)}\left[\log q_K(\mathbf{z}_K) - \log p(\mathbf{x}, \mathbf{z}_K)\right] \\
&= \mathbb{E}_{q_0(z_0)}\left[\log q_0(\mathbf{z}_0)\right] - \mathbb{E}_{q_0(z_0)}\left[\log p(\mathbf{x}, \mathbf{z}_K)\right] \\
&- \mathbb{E}_{q_0(z_0)}\left[\sum_{k=1}^{K} \log \left|1 + \mathbf{u}_k^\top \psi_k(\mathbf{z}_{k-1})\right|\right]
\end{aligned}
$$

where the initial density $q_0(\mathbf{z})$ is defined as $\mathcal{N}(\mu, \sigma)$, and μ, σ are trained with all parameters in the encoder and normalizing flows.

VAE with Normalizing Flow

In our implementation, we calculate $\mathcal{F}(x)$ in the same way as VAE, and we use Monte Carlo sampling to approximate the KL divergence between $q_K(\mathbf{z}_K)$ and $p(\mathbf{z})$.

$$
\begin{aligned}
\mathcal{F}(\mathbf{x}) = -\mathcal{L}(Q) &= \mathbb{E}_{q_\phi(z|x)}\left[\log q_\phi(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{x}, \mathbf{z})\right] \\
&= \mathbb{E}_{q_0(z_0)}\left[\log q_K\left(\mathbf{z}_K\right) - \log p\left(\mathbf{x}, \mathbf{z}_K\right)\right] \\
&= \mathbb{E}_{q_0(z_0)}\left[\log q_0\left(\mathbf{z}_0\right) - \sum_{k=1}^{K} \log \left|1 + \mathbf{u}_k^\top \psi_k\left(\mathbf{z}_{k-1}\right)\right| - \log p(\mathbf{z}_K)\right] \\
&- \mathbb{E}_{q_0(z_0)}\left[\log p\left(\mathbf{x}|\mathbf{z}_K\right)\right] \\
\\
&= \mathrm{KL}[q_K(\mathbf{z}_K) \| p(\mathbf{z})] - \text{Reconstruction Loss}
\end{aligned}
$$

Implementation

```
class PlanarFlow(nn.Module):
    def __init__(self,z_dim = None):
        super(PlanarFlow, self).__init__()
        self.init_sigma = 0.01
        self.n_features = z_dim
        self.weights = nn.Parameter(torch.randn(1, z_dim).normal_(0, self.init_sigma))
        self.bias = nn.Parameter(torch.zeros(1).normal_(0, self.init_sigma))
        self.u = nn.Parameter(torch.randn(1, z_dim).normal_(0, self.init_sigma))
```

Implementation

```python
class PlanarFlow(nn.Module):
    def forward(self, zk):
        u = self.u
        weights = self.weights
        bias = self.bias
        ## transforation for invertibility
        u_temp = (weights @ u.t.squeeze()
        m_u_temp = -1 + F.softplus(u_temp)
        uhat = u + (m_u_temp - u_temp) * (weights / (weights @ weights.t)
        z_temp = zk @ weights.t() + bias
        new_z = zk + uhat * torch.tanh(z_temp)
        ## now compute psi
        psi = (1 - torch.tanh(z_temp)**2) @ weights
        det_jac = 1 + psi @ uhat.t
        logdet_jacobian = torch.log(torch.abs(det_jac) + 1e-8).squeeze()

        return new_z, logdet_jacobian
```

Implementation

```python
class NormalizingFlows(nn.Module):
    def __init__(self, n_flows = 1, z_dim = None, flow_type = PlanarFlow):
        '''
        :param z_dims: dimension of the latent variables
        :param n_flows: how many flows we should use in term of sequence of function f_k (f_k-1(f_k-2(..
        :param flow_type: we have implemented only the Planar Flow
        '''
        super(NormalizingFlows, self).__init__()
        self.n_flows = n_flows
        self.flow_type = flow_type
        self.z_dim = z_dim

        flows_sequence = [self.flow_type(self.z_dim) for _ in range(self.n_flows)]

        self.flows = nn.ModuleList(flows_sequence)
```

Implementation

```python
class NormalizingFlows(nn.Module):


    def forward(self, z):

        logdet_jacobians = []
        for k, flow in enumerate(self.flows):
            z, logdet_j = flow(z)
            logdet_jacobians.append(logdet_j)

        z_k = z
        logdet_jacobians = torch.stack(logdet_jacobians, dim=1)
        return z_k, torch.sum(logdet_jacobians, 1)
```

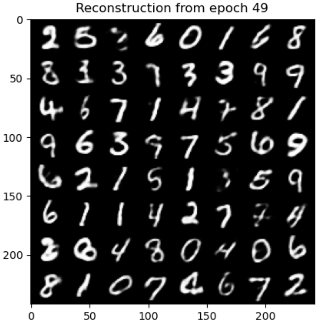
Implementation

```
BATCH_SIZE = 64
HIDDEN_LAYERS = [200,200]
Z_DIM = 40
N_FLOWS = 10

N_EPOCHS = 50
LEARNING_RATE = 1e-5

model = VariationalAutoencoderWithFlows(28*28, HIDDEN_LAYERS, Z_DIM,
    N_FLOWS)
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

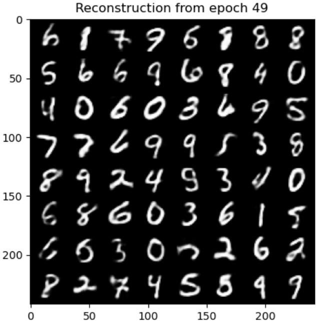
Implementation

```python
for epoch in range(N_EPOCHS):
    for i, data in enumerate(train_loader):
        images = data[0]

        # loss is KL + BCE(reconstruction)
        reconstruction = model(images)
        likelihood = F.binary_cross_entropy(reconstruction, images, reduction='sum')
        kl = model.qz - beta * model.pz
        bound = torch.sum(likelihood) + torch.sum(kl)
        L = bound / len(images)
        L.backward()
        optimizer.step()
        optimizer.zero_grad()
```

Experiment



(a) MNIST Reconstruction of VAE



(b) MNIST Reconstruction of VAE_NF

References

Rezende, D. & Mohamed, S. Variational inference with normalizing flows. *International Conference On Machine Learning*. pp. 1530-1538 (2015)