In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset
from torch.utils.data import DataLoader
from DataPreparation.dataset_preparation import get_binarymnist_dataset
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
%load_ext autoreload
%autoreload 2
```

A. Load the Dataset

Set the data directory to the path where the following files exist: binarized_mnist_train.amat, binarized_mnist_valid.amat, binarized_mnist_test.amat

In [2]:
```
data_dir = 'Dataset/BinaryMNIST/'
```

In [3]:
```
X_train, X_val, X_test, X_train_moments = get_binarymnist_dataset(data_dir, norma
mean_img, std_img = X_train_moments
print('Train data size: ', X_train.shape)
print('Val data size: ', X_val.shape)
print('Test data size: ', X_test.shape)
```

```
Train data size:  (50000, 1, 28, 28)
Val data size:  (10000, 1, 28, 28)
Test data size:  (10000, 1, 28, 28)
```

In [4]:
```
X_train_ = TensorDataset(torch.from_numpy(X_train))
loader_train  = DataLoader(X_train_, batch_size=64, shuffle=True)

X_val_ = TensorDataset(torch.from_numpy(X_val))
loader_val = DataLoader(X_val_, batch_size=64, shuffle=False)

X_test_ = TensorDataset(torch.from_numpy(X_test))
loader_test = DataLoader(X_test_, batch_size=64, shuffle=False)
```

B. Train the Model

Select device

In [5]:
```
# CUDA for PyTorch
use_cuda = torch.cuda.is_available()
device = torch.device("cuda:0" if use_cuda else "cpu")
print('Using device=GPU') if use_cuda else print('Using device=CPU')
```

```
Using device=GPU
```

Create Model

In [6]:
```
from models.vae import VAE
num_latent = 100
model = VAE(num_latent).to(device)
```

Start training

In [7]:
```
# Hyperparameters
learning_rate = 3e-4
num_epochs = 20
```

In [8]:
```
from utils.train_eval_utils import train_model
print('~~~ Training with GPU ~~~') if use_cuda else print('~~~ Training with CPU ~
num_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print('Model has %.2fK trainable parameters.\n' % (num_params/1000))

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
train_history = train_model(model, optimizer, loader_train,
                            loader_val, num_epochs,
                            device)
```

```
~~~ Training with GPU ~~~
Model has 938.83K trainable parameters.

Epoch 1:
Train: elbo -186.3591, logpx_z -169.2217, KL 17.1375
Validation: elbo -140.3563, logpx_z -117.9124, KL 22.4439
-----------
Epoch 2:
Train: elbo -127.1850, logpx_z -104.7561, KL 22.4289
Validation: elbo -118.3235, logpx_z -95.3080, KL 23.0155
-----------
Epoch 3:
Train: elbo -113.5228, logpx_z -90.3365, KL 23.1862
Validation: elbo -110.4282, logpx_z -86.9951, KL 23.4331
-----------
Epoch 4:
Train: elbo -108.1992, logpx_z -84.2305, KL 23.9687
Validation: elbo -106.7921, logpx_z -82.2773, KL 24.5147
-----------
Epoch 5:
Train: elbo -104.8240, logpx_z -80.3921, KL 24.4319
Validation: elbo -104.0739, logpx_z -79.8675, KL 24.2065
-----------
Epoch 6:
Train: elbo -102.5594, logpx_z -77.7199, KL 24.8395
Validation: elbo -102.1658, logpx_z -77.1786, KL 24.9872
-----------
Epoch 7:
Train: elbo -100.8396, logpx_z -75.7237, KL 25.1159
Validation: elbo -100.6385, logpx_z -75.6920, KL 24.9465
-----------
Epoch 8:
Train: elbo -99.5574, logpx_z -74.2242, KL 25.3332
Validation: elbo -100.0090, logpx_z -74.7373, KL 25.2717
-----------
Epoch 9:
Train: elbo -98.6307, logpx_z -73.1864, KL 25.4443
Validation: elbo -98.3265, logpx_z -72.6537, KL 25.6728
-----------
Epoch 10:
Train: elbo -97.8022, logpx_z -72.2822, KL 25.5200
Validation: elbo -98.7732, logpx_z -72.7918, KL 25.9814
-----------
Epoch 11:
Train: elbo -97.2000, logpx_z -71.5914, KL 25.6086
Validation: elbo -97.8511, logpx_z -72.0564, KL 25.7947
-----------
```

```
Epoch 12:
Train: elbo -96.6799, logpx_z -70.9974, KL 25.6825
Validation: elbo -96.4660, logpx_z -70.8864, KL 25.5796
-----------
Epoch 13:
Train: elbo -96.0924, logpx_z -70.4380, KL 25.6545
Validation: elbo -96.2487, logpx_z -70.5944, KL 25.6543
-----------
Epoch 14:
Train: elbo -95.7521, logpx_z -70.0322, KL 25.7198
Validation: elbo -96.1312, logpx_z -70.3957, KL 25.7355
-----------
Epoch 15:
Train: elbo -95.4095, logpx_z -69.6624, KL 25.7471
Validation: elbo -95.9126, logpx_z -69.9286, KL 25.9840
-----------
Epoch 16:
Train: elbo -95.0310, logpx_z -69.2819, KL 25.7491
Validation: elbo -95.1616, logpx_z -69.5236, KL 25.6381
-----------
Epoch 17:
Train: elbo -94.6776, logpx_z -68.9277, KL 25.7499
Validation: elbo -95.1575, logpx_z -69.1118, KL 26.0458
-----------
Epoch 18:
Train: elbo -94.3956, logpx_z -68.6026, KL 25.7930
Validation: elbo -94.9455, logpx_z -69.0867, KL 25.8587
-----------
Epoch 19:
Train: elbo -94.2376, logpx_z -68.4112, KL 25.8264
Validation: elbo -94.5331, logpx_z -68.8756, KL 25.6575
-----------
Epoch 20:
Train: elbo -93.9184, logpx_z -68.0668, KL 25.8517
Validation: elbo -94.3561, logpx_z -68.2783, KL 26.0778
-----------
```

C. Importance Sampling

One Minibatch

In [9]:

```
M = 32
K = 200
D = 784
X = np.reshape(X_val[:M], (M, D))
X = torch.from_numpy(X).to(device=device, dtype=torch.float32)
Z = torch.randn(M, K, num_latent) # Z gets "reparameterized" in minibatch_importa
Z = Z.to(device=device, dtype=torch.float32)
```

In [10]:
```
from utils.importance_sampling import minibatch_importance_sampling
logp = minibatch_importance_sampling(model, X, Z, device)
print('For one minibatch of validation data:')
print('(log p(x1), . . . , log p(xM)) estimates of size (M,):\n')
print(logp)
```

```
For one minibatch of validation data:
(log p(x1), . . . , log p(xM)) estimates of size (M,):

tensor([ -95.2595,  -63.9816, -113.0852,  -72.8841,  -84.2797,  -98.1954,
        -105.4902,  -70.6400,  -45.5383,  -61.5701,  -95.4371,  -96.1672,
        -102.5433,  -91.7915,  -74.7288,  -84.0134,  -54.0331,  -71.6146,
        -108.3958,  -89.2090,  -75.1369,  -81.7220,  -99.6269,  -89.2135,
        -101.3560,  -51.3683,  -90.8322,  -46.3075, -117.1302,  -86.5186,
         -43.3720, -121.1844], device='cuda:0')
```

Entire Validation and Test set

In [11]:
```
from utils.importance_sampling import importance_sampling
logp_val = importance_sampling(model, loader_val, device)
```

In [12]:
```
logp_test = importance_sampling(model, loader_test, device)
```

In [13]:
```
from utils.train_eval_utils import evaluation, criterion
val_elbo, val_logpx_z, val_kl = evaluation(model, loader_val, criterion, device)
test_elbo, test_logpx_z, test_kl = evaluation(model, loader_test, criterion, devi
```

In [14]:
```
print('Validation:')
print('(approximated) log-likelihood: %.4f' % (logp_val.cpu().numpy()))
print('ELBO: %.4f' % val_elbo.cpu().numpy())
print('~~~~~~~~~~~~~~')
print('Test:')
print('(approximated) log-likelihood: %.4f' % (logp_test.cpu().numpy()))
print('ELBO: %.4f' % test_elbo.cpu().numpy())
```

```
Validation:
(approximated) log-likelihood: -88.9486
ELBO: -94.4227
~~~~~~~~~~~~~~
Test:
(approximated) log-likelihood: -88.3406
ELBO: -93.6553
```

In [ ]:
```
```