In [0]:
```
from torchvision.datasets import utils
import torch.utils.data as data_utils
import torch
import os
import numpy as np
from torch import nn
from torch.nn.modules import upsampling
from torch.functional import F
from torch.optim import Adam
import math

from torchvision.utils import save_image

import math

torch.manual_seed(1)
np.random.seed(1)
```

To do this assignment, I did look at this code for the general framework of how to implement VAE and I did copy some code from there:

https://github.com/pytorch/examples/blob/master/vae/main.py (https://github.com/pytorch/examples/blob/master/vae/main.py)

In [0]:
```
# If a GPU is available, use it
# Pytorch uses an elegant way to keep the code device agnostic
if torch.cuda.is_available():
    device = torch.device("cuda")
    use_cuda = True
else:
    device = torch.device("cpu")
    use_cuda = False

print(device)

#torch.manual_seed(1) # I may need to fix other seeds
```

```
cuda
```

In [0]:
```
def get_data_loader(dataset_location, batch_size):
    URL = "http://www.cs.toronto.edu/~larocheh/public/datasets/binarized_mnist/"
    # start processing
    def lines_to_np_array(lines):
        return np.array([[int(i) for i in line.split()] for line in lines])
    splitdata = []
    for splitname in ["train", "valid", "test"]:
        filename = "binarized_mnist_%s.amat" % splitname
        filepath = os.path.join(dataset_location, filename)
        utils.download_url(URL + filename, dataset_location)
        with open(filepath) as f:
            lines = f.readlines()
```

```
        x = lines_to_np_array(lines).astype('float32')
        x = x.reshape(x.shape[0], 1, 28, 28)
        # pytorch data loader
        dataset = data_utils.TensorDataset(torch.from_numpy(x))
        print(splitname, len(dataset))
        dataset_loader = data_utils.DataLoader(x, batch_size=batch_size, shuffle=splitname == "tra
in")
        splitdata.append(dataset_loader)
    return splitdata
```

In [0]:

```
batch_size = 64
train, valid, test = get_data_loader("binarized_mnist", 64)
```

```
  0%|          | 0/78400000 [00:00<?, ?it/s]
Downloading http://www.cs.toronto.edu/~larocheh/public/datasets/binarized_mnist/binarized_mnist_tr
ain.amat to binarized_mnist/binarized_mnist_train.amat
78405632it [00:03, 23874664.65it/s]
  0%|          | 40960/15680000 [00:00<00:41, 379166.21it/s]
train 50000
Downloading http://www.cs.toronto.edu/~larocheh/public/datasets/binarized_mnist/binarized_mnist_va
lid.amat to binarized_mnist/binarized_mnist_valid.amat
15687680it [00:01, 9274848.23it/s]
  0%|          | 49152/15680000 [00:00<00:35, 445598.62it/s]
valid 10000
Downloading http://www.cs.toronto.edu/~larocheh/public/datasets/binarized_mnist/binarized_mnist_te
st.amat to binarized_mnist/binarized_mnist_test.amat
15687680it [00:01, 10329388.84it/s]
test 10000
```

In [0]:

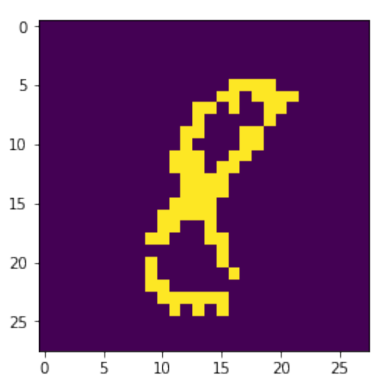
```
print(f"Your version of Pytorch is {torch.__version__}")
```

```
Your version of Pytorch is 1.0.1.post2
```

In [0]:

```
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
for x in train:
    print(x.shape)
    plt.imshow(x[0, 0])
    break
```

```
torch.Size([64, 1, 28, 28])
```



In [0]:

```
print(train)
```

```
<torch.utils.data.dataloader.DataLoader object at 0x7f39ba4478d0>
```

Question 2.1: Train a VAE (10pts)

In [0]:

```
class Q2_VAE(nn.Module):
    def __init__(self):
        super(Q2_VAE, self).__init__()
```

```
        self.m = nn.ELU()
        self.conv_e1 = nn.Conv2d(1, 32, (3, 3))
        # ELU
        self.avg_pool_e1 = nn.AvgPool2d(kernel_size = 2, stride=2)
        self.conv_e2 = nn.Conv2d(32, 64, (3, 3))
        # ELU
        self.avg_pool_e2 = nn.AvgPool2d(kernel_size = 2, stride=2)
        self.conv_e3 = nn.Conv2d(64, 256, (5, 5))
        # ELU
        # Ne pas oublier de mettre en ligne les 256 pour faire une couche de MLP

        self.linear_mean = nn.Linear(256, 100, bias=True)
        self.linear_log_var = nn.Linear(256, 100, bias=True)

        self.linear_d1 = nn.Linear(100, 256, bias=True)
        # ELU
        # Je dois augmenter de deux dimensions(inverse de .view())
        self.conv_d1 = nn.Conv2d(256, 64, kernel_size=(5, 5), padding=(4, 4))
        # ELU
        #self.upsamp_d1 =nn.UpsamplingBilinear2d(scale_factor=2, mode='bilinear')
        self.conv_d2 = nn.Conv2d(64, 32, kernel_size=(3, 3), padding=(2, 2))
        # ELU
        #self.upsamp_d2 =nn.UpsamplingBilinear2d(scale_factor=2, mode='bilinear')
        self.conv_d3 = nn.Conv2d(32, 16, kernel_size=(3, 3), padding=(2, 2))
        # ELU
        self.conv_d4 = nn.Conv2d(16, 1, kernel_size=(3, 3), padding=(2, 2))


    def encode(self, x):
        x = self.conv_e1(x)
        x = self.m(x)
        x = self.avg_pool_e1(x)
        #print("Ici: ", x.shape)
        x = self.conv_e2(x)
        x = self.m(x)
        x = self.avg_pool_e2(x)
        x = self.conv_e3(x)
        x = self.m(x)
        x = x.view(-1, 256)
        return self.linear_mean(x), self.linear_log_var(x)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5*logvar) + 10**(-7)
        eps = torch.randn_like(std)
        return mu + eps*std

    def decode(self, z):
        out = self.linear_d1(z)
        out = self.m(out)
        out = out.view(-1, 256, 1, 1) # LFPR: J'ai change ca aussi
        out = self.conv_d1(out)
        out = self.m(out)
        #out = self.upsamp_d1(out)
        out = F.interpolate(out, scale_factor=2, mode='bilinear', align_corners=True)
        out = self.conv_d2(out)
        out = self.m(out)
        #out = self.upsamp_d2(out)
        out = F.interpolate(out, scale_factor=2, mode='bilinear', align_corners=True)
        out = self.conv_d3(out)
        out = self.m(out)
        return self.conv_d4(out)


    def forward(self, x):
        #mu, logvar = self.encode(x.view(-1, 784))
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar
```

In [0]:

```
# Reconstruction + KL divergence losses summed over all elements and batch
```

```
# we return the negative of the ELBO for gradient descent
def loss_function(recon_x, x, mu, logvar):

    N_BCE=-torch.sum(F.binary_cross_entropy(torch.sigmoid(recon_x.view(-1, 784)\
                          ), x.view(-1, 784), reduction='none'),dim=1).mean()

    # 0.5 * sum(1 + log(sigma^2) - mu^2 - sigma^2)
    KLD = 0.5*torch.sum(-1 - logvar + mu.pow(2) + logvar.exp(), dim = 1).mean()

    #return BCE + KLD
    return - (N_BCE - KLD)




def train_VAE(epoch,loader):
    model.train()
    train_loss = 0
    #for batch_idx, (data, _) in enumerate(loader):
    for batch_idx, data in enumerate(loader):
        data = data.to(device)
        optimizer.zero_grad()
        recon_batch, mu, logvar = model(data)
        loss = loss_function(recon_batch, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()
        #if batch_idx % 100 == 0:
        #    print('Train Epoch: {} [{}/{} ({:.0f}%)]\t average ELBO: {:.6f}'.format(
        #        epoch, batch_idx * len(data), len(loader.dataset),
        #        100. * batch_idx / len(loader),
        #        - batch_size * loss.item() / len(data)))

    average_ELBO = - train_loss * batch_size / len(loader.dataset)
    print('====> Epoch: {} Train set average ELBO: {:.4f}'.format(
          epoch, average_ELBO))
    return average_ELBO


def test_VAE(epoch, loader, state = "Validation"):
    model.eval()
    test_loss = 0
    with torch.no_grad():
        #for i, (data, _) in enumerate(loader):
        for i, data in enumerate(loader):
            data = data.to(device)
            recon_batch, mu, logvar = model(data)
            test_loss += loss_function(recon_batch, data, mu, logvar).item()
            #if i == 0:
            #    n = min(data.size(0), 8)
            #    comparison = torch.cat([data[:n],
            #                           recon_batch.view(batch_size, 1, 28, 28)[:n]])
            #    save_image(comparison.cpu(),
            #               str(epoch) + '.png', nrow=n)

    test_loss /= (len(loader.dataset)/ batch_size)
    average_ELBO = - test_loss
    print('====> ' + state +' set average ELBO: {:.4f}'.format(average_ELBO))
    return average_ELBO
```

In [0]:
```
len(train.dataset)
```

Out[0]: 50000

In [0]:
```
model = Q2_VAE()
model = model.to(device)

#optimizer = Adam(model.parameters(), lr=1e-3)
optimizer = Adam(model.parameters(), lr=3 * 10**(-4))

print(model)
```

```
print("\n\n# Parameters: ", sum([param.nelement() for param in model.parameters()]))
```

```
Q2_VAE(
  (m): ELU(alpha=1.0)
  (conv_e1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
  (avg_pool_e1): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (conv_e2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (avg_pool_e2): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (conv_e3): Conv2d(64, 256, kernel_size=(5, 5), stride=(1, 1))
  (linear_mean): Linear(in_features=256, out_features=100, bias=True)
  (linear_log_var): Linear(in_features=256, out_features=100, bias=True)
  (linear_d1): Linear(in_features=100, out_features=256, bias=True)
  (conv_d1): Conv2d(256, 64, kernel_size=(5, 5), stride=(1, 1), padding=(4, 4))
  (conv_d2): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
  (conv_d3): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
  (conv_d4): Conv2d(16, 1, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
)


# Parameters:  938825
```

In [0]:

```
nb_epochs = 20
for epoch in range(1, nb_epochs + 1):
    train_VAE(epoch, train)
    test_VAE(epoch, valid)
with torch.no_grad():
    sample = torch.randn(64, 100).to(device)
    sample = model.decode(sample).cpu()
    save_image(sample.view(64, 1, 28, 28),
               str(epoch) + '.png')
```

```
====> Epoch: 1 Train set average ELBO: -182.2706
====> Validation set average ELBO: -137.3361
====> Epoch: 2 Train set average ELBO: -124.6678
====> Validation set average ELBO: -117.8182
====> Epoch: 3 Train set average ELBO: -112.1374
====> Validation set average ELBO: -110.0154
====> Epoch: 4 Train set average ELBO: -107.3396
====> Validation set average ELBO: -106.3443
====> Epoch: 5 Train set average ELBO: -104.4111
====> Validation set average ELBO: -104.0974
====> Epoch: 6 Train set average ELBO: -102.3293
====> Validation set average ELBO: -102.0641
====> Epoch: 7 Train set average ELBO: -100.8243
====> Validation set average ELBO: -101.0259
====> Epoch: 8 Train set average ELBO: -99.6585
====> Validation set average ELBO: -100.0452
====> Epoch: 9 Train set average ELBO: -98.7667
====> Validation set average ELBO: -99.0009
====> Epoch: 10 Train set average ELBO: -98.0199
====> Validation set average ELBO: -98.5278
====> Epoch: 11 Train set average ELBO: -97.2981
====> Validation set average ELBO: -97.7497
====> Epoch: 12 Train set average ELBO: -96.7091
====> Validation set average ELBO: -97.6286
====> Epoch: 13 Train set average ELBO: -96.3258
====> Validation set average ELBO: -96.9575
====> Epoch: 14 Train set average ELBO: -95.8360
====> Validation set average ELBO: -96.4233
====> Epoch: 15 Train set average ELBO: -95.3735
====> Validation set average ELBO: -96.0844
====> Epoch: 16 Train set average ELBO: -95.0206
====> Validation set average ELBO: -95.6550
====> Epoch: 17 Train set average ELBO: -94.7092
====> Validation set average ELBO: -95.4846
====> Epoch: 18 Train set average ELBO: -94.4579
====> Validation set average ELBO: -94.9379
====> Epoch: 19 Train set average ELBO: -94.1467
====> Validation set average ELBO: -95.0183
====> Epoch: 20 Train set average ELBO: -93.8983
====> Validation set average ELBO: -94.7597
```

As can be seen in the previous cell, we obtain an ELBO bigger than -96 on the validation set after 20 epochs.

Question 2.2: Evaluating log-likelihood with Variational Autoencoders (20 pts)

Question 2.2.1

In [0]:
```
def copy_tensor_K_times(tensor, K):
  return torch.stack([tensor.clone() for _ in range(K)])
```

In [0]:
```
def reconstruction(multi_x,z_x,model):
  recon_x = model.decode(z_x)
  return - torch.sum(F.binary_cross_entropy(torch.sigmoid(recon_x.view(-1, 784)), multi_x.view(-1,
784), reduction='none'),dim=1)
```

In [0]:
```
def log_gaussian_standard(z):
  # no more terms because mu = 0 and sigma = 1
  K = z.shape[1]
  pi_term = - (K/2) * torch.log(torch.tensor(2 * math.pi))
  return torch.sum(- z**2/2, dim = 1) + pi_term
```

In [0]:
```
def log_gaussian_density(z_x, multi_x, model):
  mu, logvar = model.encode(multi_x)
  K = z_x.shape[1]
  pi_term = - (K/2) * torch.log(torch.tensor(2 * math.pi))
  # We add 10**(-7) to avoid taking the log of 0 if logvar is very negative
  return torch.sum(-(z_x - mu)**2/(2*torch.exp(logvar)+ 10**(-7)) - torch.log(torch.exp(0.5*logvar
) + 10**(-7)), dim = 1) + pi_term
```

In [0]:
```
def importance_sampling(model, X, Z):
  X = X.view((-1,1,28,28))
  model.eval()
  log_probs = []
  with torch.no_grad():
    for pos, x in enumerate(X):
      z_x = Z[pos]
      multi_x = copy_tensor_K_times(x, Z.shape[1])
      log_p_x_z = reconstruction(multi_x,z_x,model)
      log_p_z = log_gaussian_standard(z_x)
      log_q_z_x = log_gaussian_density(z_x, multi_x, model)
      w =log_p_x_z + log_p_z - log_q_z_x
      m = torch.max(w)
      value = - torch.log(torch.tensor(Z.shape[1],dtype=torch.float64)) + m + torch.log(torch.sum(
torch.exp(w-m)))
      log_probs.append(value)
  return log_probs
```

Question 2.2.2

In [0]:
```
def all_log_X(model,loader,K):
  model.eval()
  les_log_probs = []
  with torch.no_grad():
    for batch_idx, data in enumerate(loader):
      data = data.to(device)
      for x in data:
        multi_x = copy_tensor_K_times(x, K)
        mu, log_var = model.encode(multi_x)
        z = model.reparameterize(mu, log_var)
        les_log_probs += importance_sampling(model, x.view((1,784)), z[None, : ,:])
  return les_log_probs
```

In [0]:
```
model = Q2_VAE()
model = model.to(device)

optimizer = Adam(model.parameters(), lr=3 * 10**(-4))

print(model)
print("\n\n# Parameters: ", sum([param.nelement() for param in model.parameters()]))
```

```
Q2_VAE(
  (m): ELU(alpha=1.0)
  (conv_e1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
  (avg_pool_e1): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (conv_e2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (avg_pool_e2): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (conv_e3): Conv2d(64, 256, kernel_size=(5, 5), stride=(1, 1))
  (linear_mean): Linear(in_features=256, out_features=100, bias=True)
  (linear_log_var): Linear(in_features=256, out_features=100, bias=True)
  (linear_d1): Linear(in_features=100, out_features=256, bias=True)
  (conv_d1): Conv2d(256, 64, kernel_size=(5, 5), stride=(1, 1), padding=(4, 4))
  (conv_d2): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
  (conv_d3): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
  (conv_d4): Conv2d(16, 1, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
)


# Parameters:  938825
```

In [0]:
```
nb_epochs = 20
valid_ELBOs = []
test_ELBOs = []
log_likeli_est_valid = []
log_likeli_est_test = []
for epoch in range(1, nb_epochs + 1):
    train_VAE(epoch, train)
    test_VAE(epoch, valid, "Validation")
    test_VAE(epoch, test, "Test")

    #with torch.no_grad():
    #    sample = torch.randn(64, 100).to(device)
    #    sample = model.decode(sample).cpu()
    #    save_image(sample.view(64, 1, 28, 28),
    #               str(epoch) + '.png')

v_log_like_est= torch.stack(all_log_X(model,valid,200)).mean()
print("valid log likelihood estimate: ", v_log_like_est.item())

t_log_like_est= torch.stack(all_log_X(model,test,200)).mean()
print("test log likelihood estimate: ", t_log_like_est.item())
```

```
====> Epoch: 1 Train set average ELBO: -179.5255
====> Validation set average ELBO: -137.1342
====> Test set average ELBO: -135.8453
====> Epoch: 2 Train set average ELBO: -124.9550
====> Validation set average ELBO: -117.7417
====> Test set average ELBO: -116.3942
====> Epoch: 3 Train set average ELBO: -112.5014
====> Validation set average ELBO: -110.2641
====> Test set average ELBO: -109.0000
====> Epoch: 4 Train set average ELBO: -106.9975
====> Validation set average ELBO: -106.1850
====> Test set average ELBO: -104.9958
====> Epoch: 5 Train set average ELBO: -104.1800
====> Validation set average ELBO: -103.6721
====> Test set average ELBO: -102.4390
====> Epoch: 6 Train set average ELBO: -102.1364
====> Validation set average ELBO: -101.7952
====> Test set average ELBO: -100.7761
====> Epoch: 7 Train set average ELBO: -100.7264
====> Validation set average ELBO: -101.1857
====> Test set average ELBO: -100.0746
====> Epoch: 8 Train set average ELBO: -99.5386
```

```
====> Validation set average ELBO: -99.8506
====> Test set average ELBO: -98.8698
====> Epoch: 9 Train set average ELBO: -98.6915
====> Validation set average ELBO: -99.0248
====> Test set average ELBO: -98.0717
====> Epoch: 10 Train set average ELBO: -97.9206
====> Validation set average ELBO: -98.4736
====> Test set average ELBO: -97.6371
====> Epoch: 11 Train set average ELBO: -97.3222
====> Validation set average ELBO: -97.7064
====> Test set average ELBO: -96.7827
====> Epoch: 12 Train set average ELBO: -96.7397
====> Validation set average ELBO: -97.4471
====> Test set average ELBO: -96.5815
====> Epoch: 13 Train set average ELBO: -96.2443
====> Validation set average ELBO: -97.0171
====> Test set average ELBO: -96.1624
====> Epoch: 14 Train set average ELBO: -95.9211
====> Validation set average ELBO: -96.6116
====> Test set average ELBO: -95.8621
====> Epoch: 15 Train set average ELBO: -95.4592
====> Validation set average ELBO: -96.4280
====> Test set average ELBO: -95.7773
====> Epoch: 16 Train set average ELBO: -95.1992
====> Validation set average ELBO: -96.0585
====> Test set average ELBO: -95.1844
====> Epoch: 17 Train set average ELBO: -94.8293
====> Validation set average ELBO: -95.9876
====> Test set average ELBO: -95.0994
====> Epoch: 18 Train set average ELBO: -94.5628
====> Validation set average ELBO: -95.2803
====> Test set average ELBO: -94.6178
====> Epoch: 19 Train set average ELBO: -94.3492
====> Validation set average ELBO: -95.0371
====> Test set average ELBO: -94.1706
====> Epoch: 20 Train set average ELBO: -94.0601
====> Validation set average ELBO: -94.7713
====> Test set average ELBO: -94.0951
valid log likelihood estimate:  -88.88284499816541
test log likelihood estimate:  -88.2384369109548
```

As can be seen in the results above, after training the model with 20 epochs, we obtain:

- **the ELBO on the validation set is given by -94.7713**
- **the ELBO on the test set is given by -94.0951**
- **the log-likelihood estimate on the validation set is given by -88.8828**
- **the log-likelihood estimate on the test set is given by -88.2384**

In [0]:

```python
# This is just to verify that the method importance_sampling(model, X, Z)
# accepts:
#
# a model
# An (M,D) array of xi's
# An (M,K,L) array of zik's
#
# returns
# (logp(x1),...,logp(xM)) estimates of size (M,)

K = 200

model.eval()
with torch.no_grad():
  for batch_idx, data in enumerate(train):
    data = data.to(device)
    count = 0
    les_x = []
    les_z = []
    for x in data:
      count+=1
      multi_x = copy_tensor_K_times(x, K)
```

```
      mu, log_var = model.encode(multi_x)
      z = model.reparameterize(mu, log_var)
      les_x.append(x.view((784)))
      #les_z.append(z[: ,:])
      les_z.append(z)
      if count == 5:
        les_x = torch.stack(les_x)
        les_z = torch.stack(les_z)
        break
    break
print(torch.stack(importance_sampling(model,les_x,les_z)).detach().numpy())
```