Install and import require packages

In [127]:
```
import os, time
%matplotlib inline
import matplotlib.pyplot as plt
import itertools
import pickle
import imageio

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable

from torchvision import datasets, transforms
```

Set number of epochs, learning rate and batch size

In [128]:
```
batch_size = 128
lr = 0.0002
num_of_epochs = 15 # 20
```

Load Dataset

In [129]:
```
img_size = 64
transform = transforms.Compose([
        transforms.Scale(img_size),
        transforms.ToTensor(),
        transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
])
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./data', train=True, download=True, transform=transform),
    batch_size=batch_size, shuffle=True)
```

Model

In [130]:
```
class ImageDiscriminator(nn.Module):
    def __init__(self, noise_vector_size=128):
        super(ImageDiscriminator, self).__init__()
        ##TODO: implement initial 4 Convolution layers using noise_vector_size with linear kernel(=4), stride(=2) and padding(=1)##
        ##TODO: add BatchNorm layers from second conv layer till fourth conv layer##
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=noise_vector_size, kernel_size=4, stride=2, padding=1)
        self.conv2 = nn.Conv2d(in_channels=noise_vector_size, out_channels=noise_vector_size*2, kernel_size=4, stride=2, padding=1)
        self.conv2_batchnorm = nn.BatchNorm2d(noise_vector_size*2)
        self.conv3 = nn.Conv2d(in_channels=noise_vector_size*2, out_channels=noise_vector_size*4, kernel_size=4, stride=2, padding=1)
        self.conv3_batchnorm = nn.BatchNorm2d(noise_vector_size*4)
        self.conv4 = nn.Conv2d(in_channels=noise_vector_size*4, out_channels=noise_vector_size*8, kernel_size=4, stride=2, padding=1)
        self.conv4_batchnorm = nn.BatchNorm2d(noise_vector_size*8)
        self.conv5 = nn.Conv2d(noise_vector_size * 8, 1, 4, 1, 0)

    def weight_init(self, mean, std):
        for m in self._modules:
            normal_init(self._modules[m], mean, std)

    def forward(self, input):
        ##TODO: implement forward pass (be cautious when adding BatchNorm layers)##
        x = self.conv1(input)
        x = F.relu(x)
        x = self.conv2(x)
        x = self.conv2_batchnorm(x)
        x = F.relu(x)
        x = self.conv3(x)
        x = self.conv3_batchnorm(x)
        x = F.relu(x)
        x = self.conv4(x)
        x = self.conv4_batchnorm(x)
        x = F.relu(x)
        x = F.sigmoid(self.conv5(x))

        return x
```

In [131]:
```
class ImageGenerator(nn.Module):
    def __init__(self, noise_vector_size=128):
        super(ImageGenerator, self).__init__()
        ##TODO: implement rest 4 Tranpose Convolution layers using noise_vector_size with linear kernel(=4), stride(=2) and padding(=1)##
        ##TODO: add BatchNorm layers from first conv layer till fourth conv layer##
        self.deconv1 = nn.ConvTranspose2d(100, noise_vector_size * 8, 4, 1, 0)
        self.deconv1_batchnorm = nn.BatchNorm2d(noise_vector_size*8)
        self.deconv2 = nn.ConvTranspose2d(in_channels=noise_vector_size * 8, out_channels=noise_vector_size * 4, kernel_size=4, stride=2, padding=1)
        self.deconv2_batchnorm = nn.BatchNorm2d(noise_vector_size*4)
        self.deconv3 = nn.ConvTranspose2d(in_channels=noise_vector_size * 4, out_channels=noise_vector_size * 2, kernel_size=4, stride=2, padding=1)
        self.deconv3_batchnorm = nn.BatchNorm2d(noise_vector_size*2)
        self.deconv4 = nn.ConvTranspose2d(in_channels=noise_vector_size * 2, out_channels=noise_vector_size, kernel_size=4, stride=2, padding=1)
        self.deconv4_batchnorm = nn.BatchNorm2d(noise_vector_size)
        self.deconv5 = nn.ConvTranspose2d(in_channels=noise_vector_size, out_channels=1, kernel_size=4, stride=2, padding=1)



    # weight_init
    def weight_init(self, mean, std):
        for m in self._modules:
            normal_init(self._modules[m], mean, std)

    # forward method
    def forward(self, input):
        ##TODO: implement forward pass (be cautious when adding BatchNorm layers)##

        x = self.deconv1(input)
        x = self.deconv1_batchnorm(x)
        x = F.relu(x)
        x = self.deconv2(x)
        x = self.deconv2_batchnorm(x)
        x = F.relu(x)
        x = self.deconv3(x)
        x = self.deconv3_batchnorm(x)
        x = F.relu(x)
        x = self.deconv4(x)
        x = self.deconv4_batchnorm(x)
        x = F.relu(x)
        x = self.deconv5(x)
        x = F.tanh(x)

        return x
```

In [132]:
```
###TODO: Replace normal weight initialization with xavier intialization##
def normal_init(m, mean, std):
    '''Init layer parameters.'''
    if isinstance(m, nn.ConvTranspose2d) or isinstance(m, nn.Conv2d):
        m.weight = torch.nn.init.xavier_normal(m.weight)
        if m.bias is not None:
            m.bias = torch.nn.init.constant(m.bias, 0)



def normal_init_old(m, mean, std):
    if isinstance(m, nn.ConvTranspose2d) or isinstance(m, nn.Conv2d):
        m.weight.data.normal_(mean, std)
        m.bias.data.zero_()
```

In [133]:
```
fixed_noise = torch.randn((5 * 5, 100)).view(-1, 100, 1, 1)
fixed_noise = Variable(fixed_noise.cuda(), volatile=True)
```

```
c:\program files\python35\lib\site-packages\ipykernel_launcher.py:2: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
```

In [134]:
```
def show_epoch_result(num_epoch, show = False, save = False, path = 'result.png', isFix=False):
    random_noise = torch.randn((5*5, 100)).view(-1, 100, 1, 1)
    random_noise = Variable(random_noise.cuda(), volatile=True)

    G.eval()
    if isFix:
        test_images = G(fixed_noise)
    else:
        test_images = G(random_noise)
    G.train()

    size_figure_grid = 5
    fig, ax = plt.subplots(size_figure_grid, size_figure_grid, figsize=(5, 5))
    for i, j in itertools.product(range(size_figure_grid), range(size_figure_grid)):
        ax[i, j].get_xaxis().set_visible(False)
        ax[i, j].get_yaxis().set_visible(False)

    for k in range(5*5):
        i = k // 5
        j = k % 5
        ax[i, j].cla()
        ax[i, j].imshow(test_images[k, 0].cpu().data.numpy(), cmap='gray')

    label = 'Epoch {0}'.format(num_epoch)
    fig.text(0.5, 0.04, label, ha='center')
    plt.savefig(path)

    if show:
        plt.show()
    else:
        plt.close()


def show_train_hist(hist, show = False, save = False, path = 'Train_hist.png'):
    ##TODO: implement show_train_hist function to plot losses histograms ##
    x = range(len(hist['D_losses']))

    y_discriminator = hist['D_losses']
    y_generator = hist['G_losses']

    plt.plot(x, y_discriminator, label='D_loss')
    plt.plot(x, y_generator, label='G_loss')

    plt.xlabel('Iterations')
    plt.ylabel('Loss values')

    plt.legend(loc=4)
    plt.grid(True)
    plt.tight_layout()


    if save:
        plt.savefig(path)

    if show:
        plt.show()
    else:
        plt.close()
```

In [135]:
```
G = ImageGenerator(128)
D = ImageDiscriminator(128)
G.weight_init(mean=0.0, std=0.02)
D.weight_init(mean=0.0, std=0.02)
G.cuda()
D.cuda()
```

```
c:\program files\python35\lib\site-packages\ipykernel_launcher.py:5: UserWarning: nn.init.xavier_normal is now deprecated in favor of nn.init.xavier_normal_.
  """
c:\program files\python35\lib\site-packages\ipykernel_launcher.py:7: UserWarning: nn.init.constant is now deprecated in favor of nn.init.constant_.
  import sys
```

Out[135]:
```
ImageDiscriminator(
  (conv1): Conv2d(1, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv2_batchnorm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv3_batchnorm): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv4): Conv2d(512, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv4_batchnorm): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv5): Conv2d(1024, 1, kernel_size=(4, 4), stride=(1, 1))
)
```

Optimization

In [136]:
```
BCE_loss = nn.BCELoss()
G_optimizer = optim.Adam(G.parameters(), lr=lr, betas=(0.5, 0.999))
D_optimizer = optim.Adam(D.parameters(), lr=lr, betas=(0.5, 0.999))
```

Results folder to save visualizations

In [137]:
```
if not os.path.isdir('results'):
    os.mkdir('results')
if not os.path.isdir('results/dcgan'):
    os.mkdir('results/dcgan')
if not os.path.isdir('results/dcgan/Random_results'):
    os.mkdir('results/dcgan/Random_results')
if not os.path.isdir('results/dcgan/Fixed_results'):
    os.mkdir('results/dcgan/Fixed_results')

train_hist = {}
train_hist['D_losses'] = []
train_hist['G_losses'] = []
train_hist['per_epoch_times'] = []
train_hist['total_time'] = []
num_iter = 0
```

Training

In [138]:
```
print('Started training...')
start_time = time.time()
for epoch in range(num_of_epochs):
    D_losses = []
    G_losses = []
    epoch_start_time = time.time()
    for x_, _ in list(train_loader):
        print("Iteration", num_iter, "of", len(train_loader))

        ##TODO: fill-in training discriminator procedure (hint: similar to mlp_gan.ipynb implementation) ##

        D.zero_grad()

        batch = x_.size()[0]

        pred_real = torch.ones(batch)
        pred_fake = torch.zeros(batch)

        x_, pred_real, pred_fake = Variable(x_.cuda()), Variable(pred_real.cuda()), Variable(pred_fake.cuda())
        result_Discriminator = D(x_).squeeze()
        real_loss_Discriminator = BCE_loss(result_Discriminator, pred_real)

        z = torch.randn((batch, 100)).view(-1, 100, 1, 1)
        z = Variable(z.cuda())
        result_Generator = G(z)

        result_Discriminator = D(result_Generator).squeeze()
        fake_loss_Discriminator = BCE_loss(result_Discriminator, pred_fake)
        fake_score_Discriminator = result_Discriminator.data.mean()

        train_loss_Discriminator = real_loss_Discriminator + fake_loss_Discriminator

        train_loss_Discriminator.backward()
        D_optimizer.step()

        D_losses.append(train_loss_Discriminator.item())

        ##TODO: fill-in training generator procedure (hint: similar to mlp_gan.ipynb implementation) ##
        G.zero_grad()

        z = torch.randn((batch, 100)).view(-1, 100, 1, 1)
        z = Variable(z.cuda())

        result_Generator = G(z)
        result_Discriminator = D(result_Generator).squeeze()
        train_loss_Generator = BCE_loss(result_Discriminator, pred_real)
        train_loss_Generator.backward()
        G_optimizer.step()
        G_losses.append(train_loss_Generator.item())

        num_iter += 1


    epoch_end_time = time.time()
    per_epoch_time = epoch_end_time - epoch_start_time


    print('[%d/%d] - time: %.2f, loss_d: %.3f, loss_g: %.3f' % ((epoch + 1), num_of_epochs, per_epoch_time, torch.mean(torch.FloatTensor(D_losses)),
                                                                torch.mean(torch.FloatTensor(G_losses))))
    p = 'results/dcgan/Random_results/' + str(epoch + 1) + '.png'
    fixed_p = 'results/dcgan/Fixed_results/' + str(epoch + 1) + '.png'

    ##TODO: save the epoch results with fixed_noise and randon_noise ##
    show_epoch_result((epoch+1), save=True, path=p, isFix=False)
    show_epoch_result((epoch+1), save=True, path=fixed_p, isFix=True)
    train_hist['D_losses'].append(torch.mean(torch.FloatTensor(D_losses)))
    train_hist['G_losses'].append(torch.mean(torch.FloatTensor(G_losses)))
    train_hist['per_epoch_times'].append(per_epoch_time)

end_time = time.time()
total_time = end_time - start_time
train_hist['total_time'].append(total_time)

print("Avg per epoch time: %.2f, total %d epochs time: %.2f" % (torch.mean(torch.FloatTensor(train_hist['per_epoch_times'])), num_of_epochs, total_time))
print("Training finish!... save training results")

##TODO: save the generator and discriminator weights into .pkl files ##
torch.save(G.state_dict(), "results/generator.pkl")
torch.save(D.state_dict(), "results/discriminator.pkl")

with open('results/dcgan/train_hist.pkl', 'wb') as f:
    pickle.dump(train_hist, f)

##TODO: save training histograms ##
show_train_hist(train_hist, save=True, path='results/DCGAN_train_hist.png')
```

```
Started training...
Iteration 0 of 469
Iteration 1 of 469
Iteration 2 of 469
Iteration 3 of 469
Iteration 4 of 469
Iteration 5 of 469
Iteration 6 of 469
Iteration 7 of 469
Iteration 8 of 469
Iteration 9 of 469
Iteration 10 of 469
Iteration 11 of 469
Iteration 12 of 469
Iteration 13 of 469
Iteration 14 of 469
Iteration 15 of 469
Iteration 16 of 469
Iteration 17 of 469
```

In [ ]: