GAN_starter_kit

February 15, 2021

The following model is the standard GAN which is part of **Exercise 1**. It is a very simple example and you can improve it by adding convolutions and many other ideas that we talked about if you want. Fill in the missing pieces and train it.

[14]:
```
import os
import numpy as np
import math

import torchvision.transforms as transforms
from torchvision.utils import save_image
from torch.optim.optimizer import Optimizer, required
from torch.utils.data import DataLoader
from torchvision import datasets
from torch.autograd import Variable

import torch.nn as nn
import torch.nn.functional as F
import torch

import matplotlib.pyplot as plt
```

[10]:
```
os.makedirs("images", exist_ok=True)

n_epochs = 50          #number of epochs of training
batch_size = 64        #size of the batches
lr = 0.0002            #adam: learning rate
b1 = 0.5               #adam: decay of first order momentum of gradient
b2 = 0.999             #adam: decay of second order momentum of gradient
n_cpu = 8,             #number of cpu threads to use during batch generation
latent_dim = 100       #dimensionality of the latent space
img_size = 28          #size of each image dimension
channels = 1           #number of image channels
sample_interval = 400 #interval between image samples


img_shape = (channels, img_size, img_size)

cuda = True if torch.cuda.is_available() else False
```

```python
random_seed = 1
torch.manual_seed(random_seed)

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

        def block(in_feat, out_feat, normalize=True):
            layers = [nn.Linear(in_feat, out_feat)]
            if normalize:
                layers.append(nn.BatchNorm1d(out_feat, 0.8))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers

        self.model = nn.Sequential(
            *block(latent_dim, 128, normalize=False),
            *block(128, 256),
            *block(256, 512),
            *block(512, 1024),
            nn.Linear(1024, int(np.prod(img_shape))),
            nn.Tanh()
        )

    def forward(self, z):
        img = self.model(z)
        img = img.view(img.size(0), *img_shape)
        return img


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

        self.model = nn.Sequential(
            nn.Linear(int(np.prod(img_shape)), 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 1),
            nn.Sigmoid(),
        )

    def forward(self, img):
        img_flat = img.view(img.size(0), -1)
        validity = self.model(img_flat)
```

```
        return validity


# Loss function
# Define your loss as Cross Entropy
adversarial_loss = nn.BCELoss()

# Initialize generator and discriminator
generator = Generator()
discriminator = Discriminator()

if cuda:
    generator.cuda()
    discriminator.cuda()
    adversarial_loss.cuda()

def maybe_cuda(x):
  if cuda:
    return x.cuda()
  return x

# Configure data loader
os.makedirs("../../data/mnist", exist_ok=True)
dataloader = torch.utils.data.DataLoader(
    datasets.MNIST(
        "../../data/mnist",
        train=True,
        download=True,
        transform=transforms.Compose(
            [transforms.Resize(img_size), transforms.ToTensor(), transforms.
 ↪Normalize([0.5], [0.5])]
        ),
    ),
    batch_size=batch_size,
    shuffle=True,
)

# Optimizers
optimizer_G = torch.optim.Adam(generator.parameters(), lr=lr, betas=(b1, b2))
optimizer_D = torch.optim.Adam(discriminator.parameters(), lr=lr, betas=(b1,␣
 ↪b2))

Tensor = torch.cuda.FloatTensor if cuda else torch.FloatTensor

# ----------
#  Training
# ----------
```

```
for epoch in range(n_epochs):
    for i, (imgs, _) in enumerate(dataloader):

        # Adversarial ground truths
        # We use the Cross Entropy (CE) loss. So we need labels. Define them␣
↪here:
        real_labels = maybe_cuda(torch.ones(imgs.shape[0], 1))
        fake_labels = maybe_cuda(torch.zeros(imgs.shape[0], 1))


        # Configure input
        real_imgs = maybe_cuda(imgs)  # input the real samples

        # -----------------
        #  Train Generator
        # -----------------

        optimizer_G.zero_grad()

        # Sample noise as generator input
        z = maybe_cuda(torch.randn(real_imgs.shape[0], latent_dim))  # sample z␣
↪from a standard gaussian

        # Generate a batch of images
        gen_imgs = generator(z)  # create generated images

        # Loss measures generator's ability to fool the discriminator
        # log(1 - discriminator(gen_imgs))
        d_gen_imgs = discriminator(gen_imgs)
        g_loss = -adversarial_loss(d_gen_imgs, fake_labels)  # think about how␣
↪to use the CE loss here

        g_loss.backward(retain_graph=True)
        optimizer_G.step()

        # ---------------------
        #  Train Discriminator
        # ---------------------

        optimizer_D.zero_grad()

        # Measure discriminator's ability to classify real from generated␣
↪samples
        d_real_imgs = discriminator(real_imgs)
        real_loss = adversarial_loss(d_real_imgs, real_labels)  # define loss
        gen_imgs = generator(z)
```

```
        d_gen_imgs = discriminator(gen_imgs)
        fake_loss = adversarial_loss(d_gen_imgs, fake_labels)  # define loss
        d_loss = (real_loss + fake_loss) / 2

        d_loss.backward()
        optimizer_D.step()
        if i%200 == 0:
          print(
              "[Epoch %d/%d] [Batch %d/%d] [D loss: %f] [G loss: %f]"
              % (epoch+1, n_epochs, i, len(dataloader), d_loss.item(), g_loss.
↪item())
        )

        batches_done = epoch * len(dataloader) + i
        if batches_done % sample_interval == 0:
            # You can also safe samples in your drive & maybe save your network␣
↪as well
            save_image(gen_imgs.data[:25], "images/GAN-%d.png" % batches_done,␣
↪nrow=5, normalize=True)
```

```
[Epoch 1/50] [Batch 0/938] [D loss: 0.719070] [G loss: -0.666384]
[Epoch 1/50] [Batch 200/938] [D loss: 0.620432] [G loss: -0.522527]
[Epoch 1/50] [Batch 400/938] [D loss: 0.539898] [G loss: -0.490950]
[Epoch 1/50] [Batch 600/938] [D loss: 0.465138] [G loss: -0.222982]
[Epoch 1/50] [Batch 800/938] [D loss: 0.358302] [G loss: -0.273690]
[Epoch 2/50] [Batch 0/938] [D loss: 0.647607] [G loss: -0.116920]
[Epoch 2/50] [Batch 200/938] [D loss: 0.276954] [G loss: -0.129999]
[Epoch 2/50] [Batch 400/938] [D loss: 0.363075] [G loss: -0.360594]
[Epoch 2/50] [Batch 600/938] [D loss: 0.395367] [G loss: -0.259472]
[Epoch 2/50] [Batch 800/938] [D loss: 0.484017] [G loss: -0.264310]
[Epoch 3/50] [Batch 0/938] [D loss: 0.363125] [G loss: -0.086154]
[Epoch 3/50] [Batch 200/938] [D loss: 0.316924] [G loss: -0.282790]
[Epoch 3/50] [Batch 400/938] [D loss: 0.265962] [G loss: -0.182572]
[Epoch 3/50] [Batch 600/938] [D loss: 0.329412] [G loss: -0.165280]
[Epoch 3/50] [Batch 800/938] [D loss: 0.328490] [G loss: -0.215933]
[Epoch 4/50] [Batch 0/938] [D loss: 0.445349] [G loss: -0.366410]
[Epoch 4/50] [Batch 200/938] [D loss: 0.502807] [G loss: -0.528171]
[Epoch 4/50] [Batch 400/938] [D loss: 0.383136] [G loss: -0.261337]
[Epoch 4/50] [Batch 600/938] [D loss: 0.802100] [G loss: -0.029695]
[Epoch 4/50] [Batch 800/938] [D loss: 0.393779] [G loss: -0.234597]
[Epoch 5/50] [Batch 0/938] [D loss: 0.479821] [G loss: -0.137306]
[Epoch 5/50] [Batch 200/938] [D loss: 0.360998] [G loss: -0.190544]
[Epoch 5/50] [Batch 400/938] [D loss: 0.335974] [G loss: -0.050763]
[Epoch 5/50] [Batch 600/938] [D loss: 0.322075] [G loss: -0.139466]
[Epoch 5/50] [Batch 800/938] [D loss: 0.439046] [G loss: -0.485727]
[Epoch 6/50] [Batch 0/938] [D loss: 0.471572] [G loss: -0.498054]
[Epoch 6/50] [Batch 200/938] [D loss: 0.365572] [G loss: -0.341190]
```

```
[Epoch 6/50] [Batch 400/938] [D loss: 0.411294] [G loss: -0.229558]
[Epoch 6/50] [Batch 600/938] [D loss: 0.428022] [G loss: -0.155167]
[Epoch 6/50] [Batch 800/938] [D loss: 0.385816] [G loss: -0.345837]
[Epoch 7/50] [Batch 0/938] [D loss: 0.489136] [G loss: -0.519783]
[Epoch 7/50] [Batch 200/938] [D loss: 0.570234] [G loss: -0.643384]
[Epoch 7/50] [Batch 400/938] [D loss: 0.483424] [G loss: -0.206697]
[Epoch 7/50] [Batch 600/938] [D loss: 0.379876] [G loss: -0.147225]
[Epoch 7/50] [Batch 800/938] [D loss: 0.325825] [G loss: -0.277685]
[Epoch 8/50] [Batch 0/938] [D loss: 0.575325] [G loss: -0.118298]
[Epoch 8/50] [Batch 200/938] [D loss: 0.688316] [G loss: -0.038731]
[Epoch 8/50] [Batch 400/938] [D loss: 0.448214] [G loss: -0.360264]
[Epoch 8/50] [Batch 600/938] [D loss: 0.337375] [G loss: -0.125416]
[Epoch 8/50] [Batch 800/938] [D loss: 0.344038] [G loss: -0.242805]
[Epoch 9/50] [Batch 0/938] [D loss: 0.535264] [G loss: -0.639246]
[Epoch 9/50] [Batch 200/938] [D loss: 0.334861] [G loss: -0.315512]
[Epoch 9/50] [Batch 400/938] [D loss: 0.344922] [G loss: -0.300779]
[Epoch 9/50] [Batch 600/938] [D loss: 0.350584] [G loss: -0.225593]
[Epoch 9/50] [Batch 800/938] [D loss: 0.508282] [G loss: -0.128805]
[Epoch 10/50] [Batch 0/938] [D loss: 0.786434] [G loss: -1.105237]
[Epoch 10/50] [Batch 200/938] [D loss: 0.560966] [G loss: -0.553727]
[Epoch 10/50] [Batch 400/938] [D loss: 0.635337] [G loss: -0.845545]
[Epoch 10/50] [Batch 600/938] [D loss: 0.555918] [G loss: -0.465314]
[Epoch 10/50] [Batch 800/938] [D loss: 0.511244] [G loss: -0.332363]
[Epoch 11/50] [Batch 0/938] [D loss: 0.507019] [G loss: -0.564396]
[Epoch 11/50] [Batch 200/938] [D loss: 0.470309] [G loss: -0.342311]
[Epoch 11/50] [Batch 400/938] [D loss: 0.440164] [G loss: -0.260680]
[Epoch 11/50] [Batch 600/938] [D loss: 0.661844] [G loss: -0.847209]
[Epoch 11/50] [Batch 800/938] [D loss: 0.665851] [G loss: -0.898821]
[Epoch 12/50] [Batch 0/938] [D loss: 0.513522] [G loss: -0.459040]
[Epoch 12/50] [Batch 200/938] [D loss: 1.053154] [G loss: -1.561723]
[Epoch 12/50] [Batch 400/938] [D loss: 0.568880] [G loss: -0.278795]
[Epoch 12/50] [Batch 600/938] [D loss: 0.509995] [G loss: -0.228194]
[Epoch 12/50] [Batch 800/938] [D loss: 0.602813] [G loss: -0.300621]
[Epoch 13/50] [Batch 0/938] [D loss: 0.484475] [G loss: -0.212881]
[Epoch 13/50] [Batch 200/938] [D loss: 0.579153] [G loss: -0.648610]
[Epoch 13/50] [Batch 400/938] [D loss: 0.569198] [G loss: -0.355166]
[Epoch 13/50] [Batch 600/938] [D loss: 0.650215] [G loss: -0.817834]
[Epoch 13/50] [Batch 800/938] [D loss: 0.590909] [G loss: -0.337163]
[Epoch 14/50] [Batch 0/938] [D loss: 0.514208] [G loss: -0.347629]
[Epoch 14/50] [Batch 200/938] [D loss: 0.533948] [G loss: -0.498666]
[Epoch 14/50] [Batch 400/938] [D loss: 0.577162] [G loss: -0.650702]
[Epoch 14/50] [Batch 600/938] [D loss: 0.538392] [G loss: -0.282966]
[Epoch 14/50] [Batch 800/938] [D loss: 0.576388] [G loss: -0.375773]
[Epoch 15/50] [Batch 0/938] [D loss: 0.499657] [G loss: -0.332492]
[Epoch 15/50] [Batch 200/938] [D loss: 0.664489] [G loss: -0.839258]
[Epoch 15/50] [Batch 400/938] [D loss: 0.681388] [G loss: -0.179321]
[Epoch 15/50] [Batch 600/938] [D loss: 0.496778] [G loss: -0.254243]
[Epoch 15/50] [Batch 800/938] [D loss: 0.589024] [G loss: -0.575741]
```

```
[Epoch 16/50] [Batch 0/938] [D loss: 0.563620] [G loss: -0.474582]
[Epoch 16/50] [Batch 200/938] [D loss: 0.489932] [G loss: -0.461608]
[Epoch 16/50] [Batch 400/938] [D loss: 0.506182] [G loss: -0.490470]
[Epoch 16/50] [Batch 600/938] [D loss: 0.551605] [G loss: -0.530183]
[Epoch 16/50] [Batch 800/938] [D loss: 0.560571] [G loss: -0.621348]
[Epoch 17/50] [Batch 0/938] [D loss: 0.619251] [G loss: -0.198664]
[Epoch 17/50] [Batch 200/938] [D loss: 0.556527] [G loss: -0.524632]
[Epoch 17/50] [Batch 400/938] [D loss: 0.566506] [G loss: -0.352740]
[Epoch 17/50] [Batch 600/938] [D loss: 0.668517] [G loss: -0.856788]
[Epoch 17/50] [Batch 800/938] [D loss: 0.511103] [G loss: -0.333741]
[Epoch 18/50] [Batch 0/938] [D loss: 0.532065] [G loss: -0.211808]
[Epoch 18/50] [Batch 200/938] [D loss: 0.673237] [G loss: -0.877042]
[Epoch 18/50] [Batch 400/938] [D loss: 0.558543] [G loss: -0.565773]
[Epoch 18/50] [Batch 600/938] [D loss: 0.560247] [G loss: -0.376485]
[Epoch 18/50] [Batch 800/938] [D loss: 0.589773] [G loss: -0.694209]
[Epoch 19/50] [Batch 0/938] [D loss: 0.593178] [G loss: -0.521993]
[Epoch 19/50] [Batch 200/938] [D loss: 0.534756] [G loss: -0.412776]
[Epoch 19/50] [Batch 400/938] [D loss: 0.571998] [G loss: -0.529671]
[Epoch 19/50] [Batch 600/938] [D loss: 0.597593] [G loss: -0.574472]
[Epoch 19/50] [Batch 800/938] [D loss: 0.552221] [G loss: -0.283197]
[Epoch 20/50] [Batch 0/938] [D loss: 0.546381] [G loss: -0.401016]
[Epoch 20/50] [Batch 200/938] [D loss: 0.534572] [G loss: -0.557831]
[Epoch 20/50] [Batch 400/938] [D loss: 0.543113] [G loss: -0.501013]
[Epoch 20/50] [Batch 600/938] [D loss: 0.560517] [G loss: -0.531284]
[Epoch 20/50] [Batch 800/938] [D loss: 0.552216] [G loss: -0.332831]
[Epoch 21/50] [Batch 0/938] [D loss: 0.593327] [G loss: -0.546960]
[Epoch 21/50] [Batch 200/938] [D loss: 0.615690] [G loss: -0.628901]
[Epoch 21/50] [Batch 400/938] [D loss: 0.519938] [G loss: -0.410395]
[Epoch 21/50] [Batch 600/938] [D loss: 0.510480] [G loss: -0.477084]
[Epoch 21/50] [Batch 800/938] [D loss: 0.511740] [G loss: -0.507527]
[Epoch 22/50] [Batch 0/938] [D loss: 0.541231] [G loss: -0.533881]
[Epoch 22/50] [Batch 200/938] [D loss: 0.604860] [G loss: -0.572438]
[Epoch 22/50] [Batch 400/938] [D loss: 0.573521] [G loss: -0.571445]
[Epoch 22/50] [Batch 600/938] [D loss: 0.594440] [G loss: -0.735699]
[Epoch 22/50] [Batch 800/938] [D loss: 0.549519] [G loss: -0.572315]
[Epoch 23/50] [Batch 0/938] [D loss: 0.500337] [G loss: -0.485752]
[Epoch 23/50] [Batch 200/938] [D loss: 0.562362] [G loss: -0.591108]
[Epoch 23/50] [Batch 400/938] [D loss: 0.567807] [G loss: -0.431523]
[Epoch 23/50] [Batch 600/938] [D loss: 0.627663] [G loss: -0.605864]
[Epoch 23/50] [Batch 800/938] [D loss: 0.629687] [G loss: -0.290701]
[Epoch 24/50] [Batch 0/938] [D loss: 0.583065] [G loss: -0.631712]
[Epoch 24/50] [Batch 200/938] [D loss: 0.552667] [G loss: -0.360667]
[Epoch 24/50] [Batch 400/938] [D loss: 0.531170] [G loss: -0.536615]
[Epoch 24/50] [Batch 600/938] [D loss: 0.543336] [G loss: -0.257196]
[Epoch 24/50] [Batch 800/938] [D loss: 0.574033] [G loss: -0.519316]
[Epoch 25/50] [Batch 0/938] [D loss: 0.559727] [G loss: -0.597832]
[Epoch 25/50] [Batch 200/938] [D loss: 0.662629] [G loss: -0.728528]
[Epoch 25/50] [Batch 400/938] [D loss: 0.594232] [G loss: -0.275385]
```

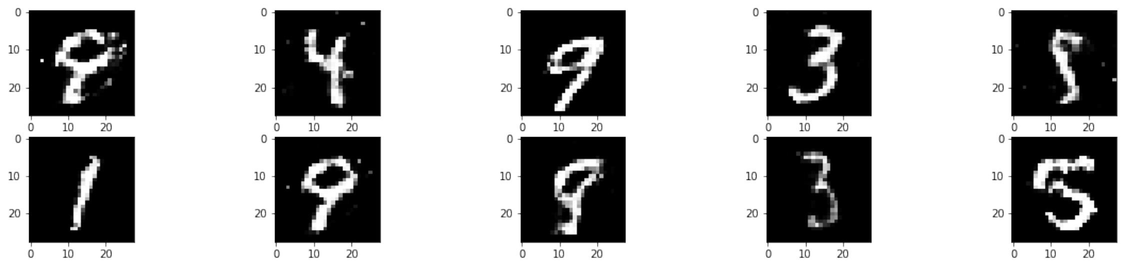
```
[Epoch 25/50] [Batch 600/938] [D loss: 0.504942] [G loss: -0.425920]
[Epoch 25/50] [Batch 800/938] [D loss: 0.515520] [G loss: -0.300349]
[Epoch 26/50] [Batch 0/938] [D loss: 0.534066] [G loss: -0.515727]
[Epoch 26/50] [Batch 200/938] [D loss: 0.624107] [G loss: -0.655980]
[Epoch 26/50] [Batch 400/938] [D loss: 0.610028] [G loss: -0.219762]
[Epoch 26/50] [Batch 600/938] [D loss: 0.547188] [G loss: -0.379519]
[Epoch 26/50] [Batch 800/938] [D loss: 0.546712] [G loss: -0.568862]
[Epoch 27/50] [Batch 0/938] [D loss: 0.604889] [G loss: -0.292726]
[Epoch 27/50] [Batch 200/938] [D loss: 0.663931] [G loss: -0.262780]
[Epoch 27/50] [Batch 400/938] [D loss: 0.571764] [G loss: -0.395743]
[Epoch 27/50] [Batch 600/938] [D loss: 0.534084] [G loss: -0.427217]
[Epoch 27/50] [Batch 800/938] [D loss: 0.557374] [G loss: -0.364255]
[Epoch 28/50] [Batch 0/938] [D loss: 0.580426] [G loss: -0.367734]
[Epoch 28/50] [Batch 200/938] [D loss: 0.553450] [G loss: -0.468786]
[Epoch 28/50] [Batch 400/938] [D loss: 0.501617] [G loss: -0.381050]
[Epoch 28/50] [Batch 600/938] [D loss: 0.598516] [G loss: -0.643869]
[Epoch 28/50] [Batch 800/938] [D loss: 0.573362] [G loss: -0.487548]
[Epoch 29/50] [Batch 0/938] [D loss: 0.707538] [G loss: -0.817951]
[Epoch 29/50] [Batch 200/938] [D loss: 0.570059] [G loss: -0.531946]
[Epoch 29/50] [Batch 400/938] [D loss: 0.603145] [G loss: -0.306942]
[Epoch 29/50] [Batch 600/938] [D loss: 0.565580] [G loss: -0.439573]
[Epoch 29/50] [Batch 800/938] [D loss: 0.576133] [G loss: -0.385863]
[Epoch 30/50] [Batch 0/938] [D loss: 0.553738] [G loss: -0.336012]
[Epoch 30/50] [Batch 200/938] [D loss: 0.547603] [G loss: -0.454067]
[Epoch 30/50] [Batch 400/938] [D loss: 0.592135] [G loss: -0.567741]
[Epoch 30/50] [Batch 600/938] [D loss: 0.522043] [G loss: -0.256358]
[Epoch 30/50] [Batch 800/938] [D loss: 0.568954] [G loss: -0.499043]
[Epoch 31/50] [Batch 0/938] [D loss: 0.502423] [G loss: -0.369959]
[Epoch 31/50] [Batch 200/938] [D loss: 0.619589] [G loss: -0.702646]
[Epoch 31/50] [Batch 400/938] [D loss: 0.584475] [G loss: -0.666871]
[Epoch 31/50] [Batch 600/938] [D loss: 0.618271] [G loss: -0.520516]
[Epoch 31/50] [Batch 800/938] [D loss: 0.526133] [G loss: -0.480065]
[Epoch 32/50] [Batch 0/938] [D loss: 0.569545] [G loss: -0.314777]
[Epoch 32/50] [Batch 200/938] [D loss: 0.568001] [G loss: -0.528085]
[Epoch 32/50] [Batch 400/938] [D loss: 0.645188] [G loss: -0.805319]
[Epoch 32/50] [Batch 600/938] [D loss: 0.561515] [G loss: -0.479390]
[Epoch 32/50] [Batch 800/938] [D loss: 0.544350] [G loss: -0.323643]
[Epoch 33/50] [Batch 0/938] [D loss: 0.556631] [G loss: -0.199238]
[Epoch 33/50] [Batch 200/938] [D loss: 0.510597] [G loss: -0.547755]
[Epoch 33/50] [Batch 400/938] [D loss: 0.651324] [G loss: -0.311990]
[Epoch 33/50] [Batch 600/938] [D loss: 0.486677] [G loss: -0.313135]
[Epoch 33/50] [Batch 800/938] [D loss: 0.601334] [G loss: -0.198083]
[Epoch 34/50] [Batch 0/938] [D loss: 0.448143] [G loss: -0.335516]
[Epoch 34/50] [Batch 200/938] [D loss: 0.539384] [G loss: -0.273428]
[Epoch 34/50] [Batch 400/938] [D loss: 0.502289] [G loss: -0.394718]
[Epoch 34/50] [Batch 600/938] [D loss: 0.588071] [G loss: -0.492902]
[Epoch 34/50] [Batch 800/938] [D loss: 0.557721] [G loss: -0.343359]
[Epoch 35/50] [Batch 0/938] [D loss: 0.567961] [G loss: -0.357449]
```

```
[Epoch 35/50] [Batch 200/938] [D loss: 0.485485] [G loss: -0.328760]
[Epoch 35/50] [Batch 400/938] [D loss: 0.536179] [G loss: -0.439546]
[Epoch 35/50] [Batch 600/938] [D loss: 0.669529] [G loss: -0.790606]
[Epoch 35/50] [Batch 800/938] [D loss: 0.574636] [G loss: -0.439668]
[Epoch 36/50] [Batch 0/938] [D loss: 0.571153] [G loss: -0.355596]
[Epoch 36/50] [Batch 200/938] [D loss: 0.529330] [G loss: -0.396028]
[Epoch 36/50] [Batch 400/938] [D loss: 0.530129] [G loss: -0.392815]
[Epoch 36/50] [Batch 600/938] [D loss: 0.604868] [G loss: -0.217481]
[Epoch 36/50] [Batch 800/938] [D loss: 0.545203] [G loss: -0.347484]
[Epoch 37/50] [Batch 0/938] [D loss: 0.587658] [G loss: -0.674753]
[Epoch 37/50] [Batch 200/938] [D loss: 0.576026] [G loss: -0.570405]
[Epoch 37/50] [Batch 400/938] [D loss: 0.644013] [G loss: -0.325585]
[Epoch 37/50] [Batch 600/938] [D loss: 0.511834] [G loss: -0.442569]
[Epoch 37/50] [Batch 800/938] [D loss: 0.579919] [G loss: -0.407314]
[Epoch 38/50] [Batch 0/938] [D loss: 0.549567] [G loss: -0.484055]
[Epoch 38/50] [Batch 200/938] [D loss: 0.547679] [G loss: -0.613152]
[Epoch 38/50] [Batch 400/938] [D loss: 0.541529] [G loss: -0.433255]
[Epoch 38/50] [Batch 600/938] [D loss: 0.592753] [G loss: -0.428420]
[Epoch 38/50] [Batch 800/938] [D loss: 0.596378] [G loss: -0.670839]
[Epoch 39/50] [Batch 0/938] [D loss: 0.473172] [G loss: -0.245465]
[Epoch 39/50] [Batch 200/938] [D loss: 0.585799] [G loss: -0.602012]
[Epoch 39/50] [Batch 400/938] [D loss: 0.556681] [G loss: -0.566754]
[Epoch 39/50] [Batch 600/938] [D loss: 0.681560] [G loss: -0.776798]
[Epoch 39/50] [Batch 800/938] [D loss: 0.552723] [G loss: -0.530697]
[Epoch 40/50] [Batch 0/938] [D loss: 0.591115] [G loss: -0.621717]
[Epoch 40/50] [Batch 200/938] [D loss: 0.553914] [G loss: -0.399567]
[Epoch 40/50] [Batch 400/938] [D loss: 0.553135] [G loss: -0.435349]
[Epoch 40/50] [Batch 600/938] [D loss: 0.535679] [G loss: -0.440585]
[Epoch 40/50] [Batch 800/938] [D loss: 0.581461] [G loss: -0.464400]
[Epoch 41/50] [Batch 0/938] [D loss: 0.577761] [G loss: -0.487081]
[Epoch 41/50] [Batch 200/938] [D loss: 0.563506] [G loss: -0.416572]
[Epoch 41/50] [Batch 400/938] [D loss: 0.578263] [G loss: -0.639890]
[Epoch 41/50] [Batch 600/938] [D loss: 0.589166] [G loss: -0.295568]
[Epoch 41/50] [Batch 800/938] [D loss: 0.559783] [G loss: -0.341698]
[Epoch 42/50] [Batch 0/938] [D loss: 0.815439] [G loss: -0.178157]
[Epoch 42/50] [Batch 200/938] [D loss: 0.577658] [G loss: -0.510794]
[Epoch 42/50] [Batch 400/938] [D loss: 0.565305] [G loss: -0.351893]
[Epoch 42/50] [Batch 600/938] [D loss: 0.518892] [G loss: -0.422861]
[Epoch 42/50] [Batch 800/938] [D loss: 0.568270] [G loss: -0.379645]
[Epoch 43/50] [Batch 0/938] [D loss: 0.528470] [G loss: -0.323028]
[Epoch 43/50] [Batch 200/938] [D loss: 0.535841] [G loss: -0.467934]
[Epoch 43/50] [Batch 400/938] [D loss: 0.558594] [G loss: -0.457768]
[Epoch 43/50] [Batch 600/938] [D loss: 0.550889] [G loss: -0.482551]
[Epoch 43/50] [Batch 800/938] [D loss: 0.680833] [G loss: -0.292648]
[Epoch 44/50] [Batch 0/938] [D loss: 0.557200] [G loss: -0.428637]
[Epoch 44/50] [Batch 200/938] [D loss: 0.561399] [G loss: -0.418635]
[Epoch 44/50] [Batch 400/938] [D loss: 0.563892] [G loss: -0.502338]
[Epoch 44/50] [Batch 600/938] [D loss: 0.607684] [G loss: -0.492990]
```

```
[Epoch 44/50] [Batch 800/938] [D loss: 0.614519] [G loss: -0.602393]
[Epoch 45/50] [Batch 0/938] [D loss: 0.563497] [G loss: -0.519847]
[Epoch 45/50] [Batch 200/938] [D loss: 0.539772] [G loss: -0.432100]
[Epoch 45/50] [Batch 400/938] [D loss: 0.563283] [G loss: -0.518921]
[Epoch 45/50] [Batch 600/938] [D loss: 0.558377] [G loss: -0.501122]
[Epoch 45/50] [Batch 800/938] [D loss: 0.531544] [G loss: -0.330587]
[Epoch 46/50] [Batch 0/938] [D loss: 0.584842] [G loss: -0.493271]
[Epoch 46/50] [Batch 200/938] [D loss: 0.588942] [G loss: -0.299990]
[Epoch 46/50] [Batch 400/938] [D loss: 0.598409] [G loss: -0.438681]
[Epoch 46/50] [Batch 600/938] [D loss: 0.522821] [G loss: -0.544778]
[Epoch 46/50] [Batch 800/938] [D loss: 0.546299] [G loss: -0.379598]
[Epoch 47/50] [Batch 0/938] [D loss: 0.518788] [G loss: -0.308364]
[Epoch 47/50] [Batch 200/938] [D loss: 0.595823] [G loss: -0.521731]
[Epoch 47/50] [Batch 400/938] [D loss: 0.578555] [G loss: -0.354177]
[Epoch 47/50] [Batch 600/938] [D loss: 0.651510] [G loss: -0.556522]
[Epoch 47/50] [Batch 800/938] [D loss: 0.558893] [G loss: -0.351895]
[Epoch 48/50] [Batch 0/938] [D loss: 0.541208] [G loss: -0.253346]
[Epoch 48/50] [Batch 200/938] [D loss: 0.559518] [G loss: -0.628083]
[Epoch 48/50] [Batch 400/938] [D loss: 0.619046] [G loss: -0.419768]
[Epoch 48/50] [Batch 600/938] [D loss: 0.532836] [G loss: -0.339162]
[Epoch 48/50] [Batch 800/938] [D loss: 0.604739] [G loss: -0.550234]
[Epoch 49/50] [Batch 0/938] [D loss: 0.535094] [G loss: -0.290533]
[Epoch 49/50] [Batch 200/938] [D loss: 0.575945] [G loss: -0.418953]
[Epoch 49/50] [Batch 400/938] [D loss: 0.516855] [G loss: -0.345411]
[Epoch 49/50] [Batch 600/938] [D loss: 0.581655] [G loss: -0.507269]
[Epoch 49/50] [Batch 800/938] [D loss: 0.557316] [G loss: -0.356614]
[Epoch 50/50] [Batch 0/938] [D loss: 0.581339] [G loss: -0.291083]
[Epoch 50/50] [Batch 200/938] [D loss: 0.592606] [G loss: -0.629883]
[Epoch 50/50] [Batch 400/938] [D loss: 0.607784] [G loss: -0.458029]
[Epoch 50/50] [Batch 600/938] [D loss: 0.595798] [G loss: -0.514410]
[Epoch 50/50] [Batch 800/938] [D loss: 0.579557] [G loss: -0.444930]
```

[12]:
```
torch.manual_seed(42)
example_z = maybe_cuda(torch.randn(10, latent_dim))
example_gen = generator(example_z)
```

[20]:
```
plt.figure(figsize=(20, 4))
imshow_shape = img_shape[1:] + img_shape[:1]
if imshow_shape[-1] == 1:
  imshow_shape = imshow_shape[:2]
for i in range(10):
  plt.subplot(2, 5, i + 1)
  plt.imshow(example_gen[i].cpu().detach().reshape(*imshow_shape), cmap='gray')
```



[23]:
```python
from collections import Counter
from functools import wraps, partial


def collect_after(c):
  def decorator(f):
    @wraps(f)
    def g(*args, **kwargs):
      return c(f(*args, **kwargs))
    return g
  return decorator


class KNearestNeighbors():
    '''
    Think about defining more functions that will help you building this algorithm.
    Optimally, one that takes in k and a test image as a parameter.
    '''
    def squared_euclidean_distance(self, x_1, x_2, axis=1):
      '''
      np.sum(x, axis = 1) will be summing all elements over the pixel dimension (axis = 1)
      '''
      return np.sum((x_1 - x_2) ** 2, axis=axis)

    def set_k(self, k):
      self.k = k

    def __init__(self, k=3):
      self.set_k(k)

    def fit(self, X, y):
      self.X = X
      self.y = y
```

```
  @collect_after(tuple)
  def calculate_nearest(self, X, k=None):
    if k is None:
      k = self.k
    for obj in X:
      nearest = []
      for i, known in enumerate(self.X):
        d = self.squared_euclidean_distance(known, obj, axis=0)
        nearest.append((d, i))
        nearest.sort()
        nearest[k:] = []
      yield tuple(nearest)

  @collect_after(partial(np.fromiter, dtype=float))
  def predict_class(self, X, nearest=None):
    res = []
    k = self.k
    if nearest is None:
      nearest = self.calculate_nearest(X, k)
    for nearest_current in nearest:
      yield Counter(self.y[i] for _, i in nearest_current[:k]).
↪most_common(1)[0][0]

  def predict_class_different_k(self, X, ks=(3,)):
    ks = tuple(ks)
    nearest = self.calculate_nearest(X, max(ks))
    for k in ks:
      self.set_k(k)
      yield (k, self.predict_class(X, nearest=nearest))


def accuracy(y_actual, y_pred):
  return (y_actual == y_pred).sum() / len(y_actual)
```

[30]:

```
dataset_train = datasets.MNIST(
        "../../data/mnist",
        train=True,
        download=True,
        transform=transforms.Compose(
            [transforms.Resize(img_size), transforms.ToTensor(), transforms.
↪Normalize([0.5], [0.5])]
        ),
    )
X_train, y_train = next(iter(DataLoader(dataset_train, len(dataset_train))))
X_train = X_train.reshape(X_train.shape[0], -1).numpy()
y_train = y_train.numpy()
```

[29]:
```
dataset_test = datasets.MNIST(
        "../../data/mnist",
        train=False,
        download=True,
        transform=transforms.Compose(
            [transforms.Resize(img_size), transforms.ToTensor(), transforms.
↪Normalize([0.5], [0.5])]
        ),
    )
X_test, y_test = next(iter(DataLoader(dataset_test, len(dataset_test))))
X_test = X_test.reshape(X_test.shape[0], -1).numpy()
y_test = y_test.numpy()
```

[34]:
```
%%time
knn_model = KNearestNeighbors()
knn_model.fit(X_train[::12], y_train[::12])
y_pred = knn_model.predict_class(X_test[::100])
print(accuracy(y_test[::100], y_pred))
```

```
0.92
CPU times: user 3.85 s, sys: 2 ms, total: 3.85 s
Wall time: 3.85 s
```

[38]:
```
%%time
torch.manual_seed(42)
z = maybe_cuda(torch.randn(1000, latent_dim))
gen_img = generator(z).cpu().detach().reshape(z.shape[0], -1).numpy()
```

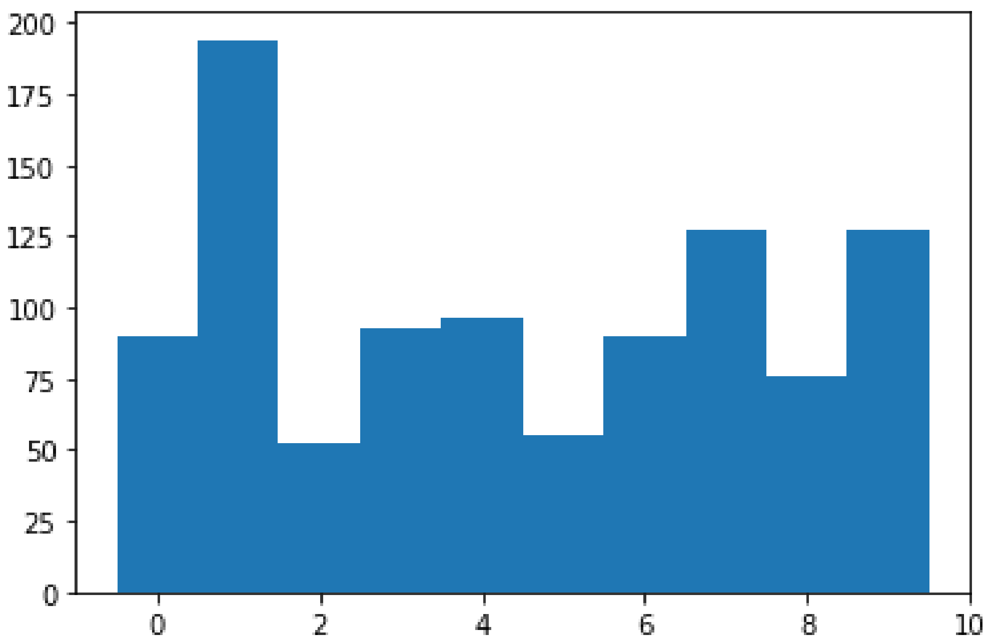
```
CPU times: user 7.18 ms, sys: 1 ms, total: 8.19 ms
Wall time: 9.82 ms
```

[43]:
```
%%time
gen_img_classes = knn_model.predict_class(gen_img)
```

```
CPU times: user 38.6 s, sys: 13.6 ms, total: 38.6 s
Wall time: 38.6 s
```

[49]:
```
plt.hist(gen_img_classes, bins=np.array(list(range(11))) - .5)
```

[49]:
```
(array([ 90., 194.,  52.,  93.,  96.,  55.,  90., 127.,  76., 127.]),
 array([-0.5,  0.5,  1.5,  2.5,  3.5,  4.5,  5.5,  6.5,  7.5,  8.5,  9.5]),
 <a list of 10 Patch objects>)
```



[67]:
```
class CGenerator(nn.Module):
    def __init__(self, y_len):
        super(CGenerator, self).__init__()

        def block(in_feat, out_feat, normalize=True):
            layers = [nn.Linear(in_feat, out_feat)]
            if normalize:
                layers.append(nn.BatchNorm1d(out_feat, 0.8))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers

        self.model = nn.Sequential(
            *block(latent_dim + y_len, 128, normalize=False),
            *block(128, 256),
            *block(256, 512),
            *block(512, 1024),
            nn.Linear(1024, int(np.prod(img_shape))),
            nn.Tanh()
        )

    def forward(self, z, y):
        img = self.model(torch.cat((z, y), 1))
        img = img.view(img.size(0), *img_shape)
        return img
```

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

        self.model = nn.Sequential(
            nn.Linear(int(np.prod(img_shape) + y_len), 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 1),
            nn.Sigmoid(),
        )

    def forward(self, img, y):
        img_flat = img.view(img.size(0), -1)
        validity = self.model(torch.cat((img_flat, y), 1))

        return validity
```

[75]:

```
%%time
torch.manual_seed(random_seed)
cgenerator = maybe_cuda(CGenerator(1))
cdiscriminator = maybe_cuda(CDiscriminator(1))
dataloader = torch.utils.data.DataLoader(
    datasets.MNIST(
        "../../data/mnist",
        train=True,
        download=True,
        transform=transforms.Compose(
            [transforms.Resize(img_size), transforms.ToTensor(), transforms.
 ↪Normalize([0.5], [0.5])]
        ),
    ),
    batch_size=batch_size,
    shuffle=True,
)

# Optimizers
optimizer_CG = torch.optim.Adam(cgenerator.parameters(), lr=lr, betas=(b1, b2))
optimizer_CD = torch.optim.Adam(cdiscriminator.parameters(), lr=lr, betas=(b1,␣
 ↪b2))

# ----------
#  Training
# ----------

for epoch in range(n_epochs):
```

```python
    for i, (imgs, real_y) in enumerate(dataloader):

        # Adversarial ground truths
        # We use the Cross Entropy (CE) loss. So we need labels. Define them␣
↪here:
        real_labels = maybe_cuda(torch.ones(imgs.shape[0], 1))
        fake_labels = maybe_cuda(torch.zeros(imgs.shape[0], 1))


        # Configure input
        real_imgs = maybe_cuda(imgs)  # input the real samples
        real_y = maybe_cuda(real_y.reshape(-1, 1))

        # -----------------
        #  Train Generator
        # -----------------

        optimizer_CG.zero_grad()

        # Sample noise as generator input
        z = maybe_cuda(torch.randn(real_imgs.shape[0], latent_dim))  # sample z␣
↪from a standard gaussian
        fake_y = maybe_cuda(torch.randint(0, 10, (real_imgs.shape[0], 1)))  #␣
↪is there a better way?

        # Generate a batch of images
        gen_imgs = cgenerator(z, fake_y)  # create generated images

        # Loss measures generator's ability to fool the discriminator
        # log(1 - discriminator(gen_imgs))
        d_gen_imgs = cdiscriminator(gen_imgs, fake_y)
        g_loss = -adversarial_loss(d_gen_imgs, fake_labels)  # think about how␣
↪to use the CE loss here

        g_loss.backward()
        optimizer_CG.step()

        # ---------------------
        #  Train Discriminator
        # ---------------------

        optimizer_CD.zero_grad()

        # Measure discriminator's ability to classify real from generated␣
↪samples
        d_real_imgs = cdiscriminator(real_imgs, real_y)
        real_loss = adversarial_loss(d_real_imgs, real_labels)  # define loss
```

```
        #gen_imgs = cgenerator(z, fake_y).detach()
        gen_imgs = cgenerator(z, fake_y)
        d_gen_imgs = cdiscriminator(gen_imgs, fake_y)
        fake_loss = adversarial_loss(d_gen_imgs, fake_labels)  # define loss
        d_loss = (real_loss + fake_loss) / 2

        d_loss.backward()
        optimizer_CD.step()
        if i%200 == 0:
          print(
              "[Epoch %d/%d] [Batch %d/%d] [D loss: %f] [G loss: %f]"
              % (epoch+1, n_epochs, i, len(dataloader), d_loss.item(), g_loss.
↪item())
        )

        batches_done = epoch * len(dataloader) + i
        if batches_done % sample_interval == 0:
            # You can also safe samples in your drive & maybe save your network␣
↪as well
            save_image(gen_imgs.data[:25], "images/CGAN-%d.png" % batches_done,␣
↪nrow=5, normalize=True)
```

```
[Epoch 1/50] [Batch 0/938] [D loss: 0.705842] [G loss: -0.690002]
[Epoch 1/50] [Batch 200/938] [D loss: 0.549300] [G loss: -0.594692]
[Epoch 1/50] [Batch 400/938] [D loss: 0.472197] [G loss: -0.315305]
[Epoch 1/50] [Batch 600/938] [D loss: 0.341053] [G loss: -0.295353]
[Epoch 1/50] [Batch 800/938] [D loss: 0.430473] [G loss: -0.250295]
[Epoch 2/50] [Batch 0/938] [D loss: 0.699149] [G loss: -0.592636]
[Epoch 2/50] [Batch 200/938] [D loss: 0.381761] [G loss: -0.303871]
[Epoch 2/50] [Batch 400/938] [D loss: 0.392030] [G loss: -0.334732]
[Epoch 2/50] [Batch 600/938] [D loss: 0.340351] [G loss: -0.079854]
[Epoch 2/50] [Batch 800/938] [D loss: 0.503274] [G loss: -0.469074]
[Epoch 3/50] [Batch 0/938] [D loss: 0.530194] [G loss: -0.049962]
[Epoch 3/50] [Batch 200/938] [D loss: 0.317677] [G loss: -0.308035]
[Epoch 3/50] [Batch 400/938] [D loss: 0.430354] [G loss: -0.413808]
[Epoch 3/50] [Batch 600/938] [D loss: 0.321960] [G loss: -0.129277]
[Epoch 3/50] [Batch 800/938] [D loss: 0.365198] [G loss: -0.324309]
[Epoch 4/50] [Batch 0/938] [D loss: 0.448758] [G loss: -0.104852]
[Epoch 4/50] [Batch 200/938] [D loss: 0.224488] [G loss: -0.207917]
[Epoch 4/50] [Batch 400/938] [D loss: 0.482807] [G loss: -0.562844]
[Epoch 4/50] [Batch 600/938] [D loss: 0.501324] [G loss: -0.515775]
[Epoch 4/50] [Batch 800/938] [D loss: 0.278035] [G loss: -0.220674]
[Epoch 5/50] [Batch 0/938] [D loss: 0.347011] [G loss: -0.090251]
[Epoch 5/50] [Batch 200/938] [D loss: 0.403058] [G loss: -0.409845]
[Epoch 5/50] [Batch 400/938] [D loss: 0.481523] [G loss: -0.058373]
[Epoch 5/50] [Batch 600/938] [D loss: 0.471271] [G loss: -0.232258]
[Epoch 5/50] [Batch 800/938] [D loss: 0.259859] [G loss: -0.151546]
```

```
[Epoch 6/50] [Batch 0/938] [D loss: 0.470887] [G loss: -0.031780]
[Epoch 6/50] [Batch 200/938] [D loss: 0.492347] [G loss: -0.537291]
[Epoch 6/50] [Batch 400/938] [D loss: 0.304123] [G loss: -0.123261]
[Epoch 6/50] [Batch 600/938] [D loss: 0.396931] [G loss: -0.118722]
[Epoch 6/50] [Batch 800/938] [D loss: 0.414158] [G loss: -0.319882]
[Epoch 7/50] [Batch 0/938] [D loss: 0.473204] [G loss: -0.406807]
[Epoch 7/50] [Batch 200/938] [D loss: 0.420949] [G loss: -0.132458]
[Epoch 7/50] [Batch 400/938] [D loss: 0.556340] [G loss: -0.705285]
[Epoch 7/50] [Batch 600/938] [D loss: 0.483421] [G loss: -0.236560]
[Epoch 7/50] [Batch 800/938] [D loss: 0.454564] [G loss: -0.147211]
[Epoch 8/50] [Batch 0/938] [D loss: 0.712582] [G loss: -0.092782]
[Epoch 8/50] [Batch 200/938] [D loss: 0.539856] [G loss: -0.443660]
[Epoch 8/50] [Batch 400/938] [D loss: 0.434870] [G loss: -0.306260]
[Epoch 8/50] [Batch 600/938] [D loss: 0.450507] [G loss: -0.465401]
[Epoch 8/50] [Batch 800/938] [D loss: 0.526125] [G loss: -0.382320]
[Epoch 9/50] [Batch 0/938] [D loss: 0.577022] [G loss: -0.625036]
[Epoch 9/50] [Batch 200/938] [D loss: 0.618327] [G loss: -0.223914]
[Epoch 9/50] [Batch 400/938] [D loss: 0.519299] [G loss: -0.283864]
[Epoch 9/50] [Batch 600/938] [D loss: 0.439904] [G loss: -0.237945]
[Epoch 9/50] [Batch 800/938] [D loss: 0.398598] [G loss: -0.162207]
[Epoch 10/50] [Batch 0/938] [D loss: 0.503813] [G loss: -0.284134]
[Epoch 10/50] [Batch 200/938] [D loss: 0.553466] [G loss: -0.701928]
[Epoch 10/50] [Batch 400/938] [D loss: 0.499472] [G loss: -0.282371]
[Epoch 10/50] [Batch 600/938] [D loss: 0.617721] [G loss: -0.234769]
[Epoch 10/50] [Batch 800/938] [D loss: 0.549715] [G loss: -0.582274]
[Epoch 11/50] [Batch 0/938] [D loss: 0.445729] [G loss: -0.293955]
[Epoch 11/50] [Batch 200/938] [D loss: 0.459551] [G loss: -0.427352]
[Epoch 11/50] [Batch 400/938] [D loss: 0.461053] [G loss: -0.427846]
[Epoch 11/50] [Batch 600/938] [D loss: 0.558747] [G loss: -0.189934]
[Epoch 11/50] [Batch 800/938] [D loss: 0.505160] [G loss: -0.365937]
[Epoch 12/50] [Batch 0/938] [D loss: 0.477547] [G loss: -0.339381]
[Epoch 12/50] [Batch 200/938] [D loss: 0.685972] [G loss: -0.110046]
[Epoch 12/50] [Batch 400/938] [D loss: 0.509498] [G loss: -0.205269]
[Epoch 12/50] [Batch 600/938] [D loss: 0.549195] [G loss: -0.394101]
[Epoch 12/50] [Batch 800/938] [D loss: 0.500605] [G loss: -0.558685]
[Epoch 13/50] [Batch 0/938] [D loss: 0.562872] [G loss: -0.773384]
[Epoch 13/50] [Batch 200/938] [D loss: 0.481322] [G loss: -0.336353]
[Epoch 13/50] [Batch 400/938] [D loss: 0.622733] [G loss: -0.157053]
[Epoch 13/50] [Batch 600/938] [D loss: 0.491388] [G loss: -0.263451]
[Epoch 13/50] [Batch 800/938] [D loss: 0.472478] [G loss: -0.363692]
[Epoch 14/50] [Batch 0/938] [D loss: 0.552238] [G loss: -0.373074]
[Epoch 14/50] [Batch 200/938] [D loss: 0.636641] [G loss: -0.778095]
[Epoch 14/50] [Batch 400/938] [D loss: 0.461117] [G loss: -0.247738]
[Epoch 14/50] [Batch 600/938] [D loss: 0.518754] [G loss: -0.400599]
[Epoch 14/50] [Batch 800/938] [D loss: 0.550283] [G loss: -0.497959]
[Epoch 15/50] [Batch 0/938] [D loss: 0.492605] [G loss: -0.556146]
[Epoch 15/50] [Batch 200/938] [D loss: 0.597418] [G loss: -0.632966]
[Epoch 15/50] [Batch 400/938] [D loss: 0.526905] [G loss: -0.500160]
```

```
[Epoch 15/50] [Batch 600/938] [D loss: 0.503572] [G loss: -0.426405]
[Epoch 15/50] [Batch 800/938] [D loss: 0.551046] [G loss: -0.632066]
[Epoch 16/50] [Batch 0/938] [D loss: 0.471051] [G loss: -0.185653]
[Epoch 16/50] [Batch 200/938] [D loss: 0.646359] [G loss: -0.807617]
[Epoch 16/50] [Batch 400/938] [D loss: 0.549695] [G loss: -0.567273]
[Epoch 16/50] [Batch 600/938] [D loss: 0.578807] [G loss: -0.678210]
[Epoch 16/50] [Batch 800/938] [D loss: 0.615090] [G loss: -0.827501]
[Epoch 17/50] [Batch 0/938] [D loss: 0.534330] [G loss: -0.258853]
[Epoch 17/50] [Batch 200/938] [D loss: 0.593918] [G loss: -0.663938]
[Epoch 17/50] [Batch 400/938] [D loss: 0.641908] [G loss: -0.758302]
[Epoch 17/50] [Batch 600/938] [D loss: 0.625425] [G loss: -0.347295]
[Epoch 17/50] [Batch 800/938] [D loss: 0.488582] [G loss: -0.290506]
[Epoch 18/50] [Batch 0/938] [D loss: 0.589086] [G loss: -0.554654]
[Epoch 18/50] [Batch 200/938] [D loss: 0.547792] [G loss: -0.344359]
[Epoch 18/50] [Batch 400/938] [D loss: 0.646784] [G loss: -0.762636]
[Epoch 18/50] [Batch 600/938] [D loss: 0.530045] [G loss: -0.579963]
[Epoch 18/50] [Batch 800/938] [D loss: 0.579467] [G loss: -0.563219]
[Epoch 19/50] [Batch 0/938] [D loss: 0.674717] [G loss: -0.332494]
[Epoch 19/50] [Batch 200/938] [D loss: 0.517007] [G loss: -0.511908]
[Epoch 19/50] [Batch 400/938] [D loss: 0.561962] [G loss: -0.384775]
[Epoch 19/50] [Batch 600/938] [D loss: 0.581682] [G loss: -0.572691]
[Epoch 19/50] [Batch 800/938] [D loss: 0.534609] [G loss: -0.525396]
[Epoch 20/50] [Batch 0/938] [D loss: 0.577138] [G loss: -0.418849]
[Epoch 20/50] [Batch 200/938] [D loss: 0.560748] [G loss: -0.388612]
[Epoch 20/50] [Batch 400/938] [D loss: 0.635994] [G loss: -0.159727]
[Epoch 20/50] [Batch 600/938] [D loss: 0.559242] [G loss: -0.657510]
[Epoch 20/50] [Batch 800/938] [D loss: 0.640203] [G loss: -0.854321]
[Epoch 21/50] [Batch 0/938] [D loss: 0.491516] [G loss: -0.386707]
[Epoch 21/50] [Batch 200/938] [D loss: 0.549417] [G loss: -0.296937]
[Epoch 21/50] [Batch 400/938] [D loss: 0.615568] [G loss: -0.785346]
[Epoch 21/50] [Batch 600/938] [D loss: 0.518173] [G loss: -0.322229]
[Epoch 21/50] [Batch 800/938] [D loss: 0.616405] [G loss: -0.650611]
[Epoch 22/50] [Batch 0/938] [D loss: 0.488806] [G loss: -0.423153]
[Epoch 22/50] [Batch 200/938] [D loss: 0.559048] [G loss: -0.484575]
[Epoch 22/50] [Batch 400/938] [D loss: 0.523868] [G loss: -0.580655]
[Epoch 22/50] [Batch 600/938] [D loss: 0.597653] [G loss: -0.518753]
[Epoch 22/50] [Batch 800/938] [D loss: 0.518859] [G loss: -0.338897]
[Epoch 23/50] [Batch 0/938] [D loss: 0.510442] [G loss: -0.348496]
[Epoch 23/50] [Batch 200/938] [D loss: 0.556305] [G loss: -0.421952]
[Epoch 23/50] [Batch 400/938] [D loss: 0.614485] [G loss: -0.306151]
[Epoch 23/50] [Batch 600/938] [D loss: 0.441229] [G loss: -0.436681]
[Epoch 23/50] [Batch 800/938] [D loss: 0.539870] [G loss: -0.440973]
[Epoch 24/50] [Batch 0/938] [D loss: 0.552054] [G loss: -0.318357]
[Epoch 24/50] [Batch 200/938] [D loss: 0.490962] [G loss: -0.494032]
[Epoch 24/50] [Batch 400/938] [D loss: 0.559929] [G loss: -0.353842]
[Epoch 24/50] [Batch 600/938] [D loss: 0.561248] [G loss: -0.674218]
[Epoch 24/50] [Batch 800/938] [D loss: 0.522740] [G loss: -0.499892]
[Epoch 25/50] [Batch 0/938] [D loss: 0.473847] [G loss: -0.424652]
```

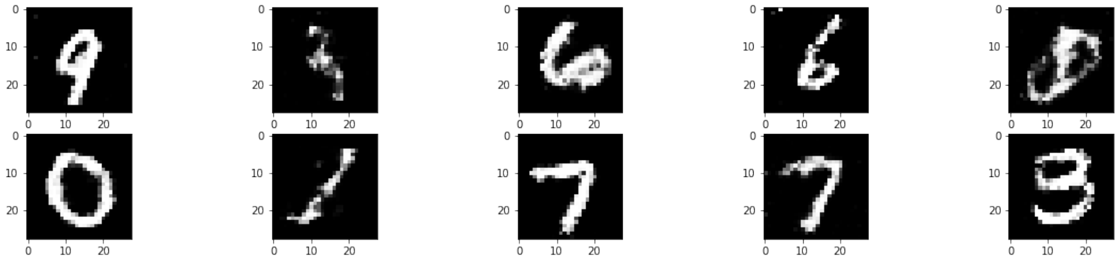
```
[Epoch 25/50] [Batch 200/938] [D loss: 0.511986] [G loss: -0.373064]
[Epoch 25/50] [Batch 400/938] [D loss: 0.491050] [G loss: -0.301430]
[Epoch 25/50] [Batch 600/938] [D loss: 0.536915] [G loss: -0.493017]
[Epoch 25/50] [Batch 800/938] [D loss: 0.520535] [G loss: -0.470841]
[Epoch 26/50] [Batch 0/938] [D loss: 0.564097] [G loss: -0.420802]
[Epoch 26/50] [Batch 200/938] [D loss: 0.482333] [G loss: -0.375533]
[Epoch 26/50] [Batch 400/938] [D loss: 0.555486] [G loss: -0.265412]
[Epoch 26/50] [Batch 600/938] [D loss: 0.476908] [G loss: -0.427231]
[Epoch 26/50] [Batch 800/938] [D loss: 0.473258] [G loss: -0.382890]
[Epoch 27/50] [Batch 0/938] [D loss: 0.534431] [G loss: -0.363377]
[Epoch 27/50] [Batch 200/938] [D loss: 0.541348] [G loss: -0.418230]
[Epoch 27/50] [Batch 400/938] [D loss: 0.581832] [G loss: -0.602318]
[Epoch 27/50] [Batch 600/938] [D loss: 0.512931] [G loss: -0.525505]
[Epoch 27/50] [Batch 800/938] [D loss: 0.574144] [G loss: -0.499322]
[Epoch 28/50] [Batch 0/938] [D loss: 0.547373] [G loss: -0.330248]
[Epoch 28/50] [Batch 200/938] [D loss: 0.673778] [G loss: -0.912287]
[Epoch 28/50] [Batch 400/938] [D loss: 0.571149] [G loss: -0.457027]
[Epoch 28/50] [Batch 600/938] [D loss: 0.509589] [G loss: -0.533982]
[Epoch 28/50] [Batch 800/938] [D loss: 0.492504] [G loss: -0.354503]
[Epoch 29/50] [Batch 0/938] [D loss: 0.549127] [G loss: -0.576102]
[Epoch 29/50] [Batch 200/938] [D loss: 0.482164] [G loss: -0.311763]
[Epoch 29/50] [Batch 400/938] [D loss: 0.508148] [G loss: -0.262033]
[Epoch 29/50] [Batch 600/938] [D loss: 0.497248] [G loss: -0.478924]
[Epoch 29/50] [Batch 800/938] [D loss: 0.594896] [G loss: -0.271733]
[Epoch 30/50] [Batch 0/938] [D loss: 0.550597] [G loss: -0.592908]
[Epoch 30/50] [Batch 200/938] [D loss: 0.578543] [G loss: -0.258491]
[Epoch 30/50] [Batch 400/938] [D loss: 0.500028] [G loss: -0.363217]
[Epoch 30/50] [Batch 600/938] [D loss: 0.540480] [G loss: -0.475842]
[Epoch 30/50] [Batch 800/938] [D loss: 0.514019] [G loss: -0.462817]
[Epoch 31/50] [Batch 0/938] [D loss: 0.521599] [G loss: -0.284645]
[Epoch 31/50] [Batch 200/938] [D loss: 0.501503] [G loss: -0.459545]
[Epoch 31/50] [Batch 400/938] [D loss: 0.476899] [G loss: -0.289764]
[Epoch 31/50] [Batch 600/938] [D loss: 0.429228] [G loss: -0.329469]
[Epoch 31/50] [Batch 800/938] [D loss: 0.516123] [G loss: -0.445352]
[Epoch 32/50] [Batch 0/938] [D loss: 0.474847] [G loss: -0.432490]
[Epoch 32/50] [Batch 200/938] [D loss: 0.628887] [G loss: -0.762807]
[Epoch 32/50] [Batch 400/938] [D loss: 0.498118] [G loss: -0.373685]
[Epoch 32/50] [Batch 600/938] [D loss: 0.491651] [G loss: -0.236449]
[Epoch 32/50] [Batch 800/938] [D loss: 0.527294] [G loss: -0.314138]
[Epoch 33/50] [Batch 0/938] [D loss: 0.539074] [G loss: -0.559366]
[Epoch 33/50] [Batch 200/938] [D loss: 0.483358] [G loss: -0.430786]
[Epoch 33/50] [Batch 400/938] [D loss: 0.490766] [G loss: -0.217033]
[Epoch 33/50] [Batch 600/938] [D loss: 0.560357] [G loss: -0.368152]
[Epoch 33/50] [Batch 800/938] [D loss: 0.508840] [G loss: -0.277565]
[Epoch 34/50] [Batch 0/938] [D loss: 0.637712] [G loss: -0.262799]
[Epoch 34/50] [Batch 200/938] [D loss: 0.489687] [G loss: -0.401869]
[Epoch 34/50] [Batch 400/938] [D loss: 0.526074] [G loss: -0.474210]
[Epoch 34/50] [Batch 600/938] [D loss: 0.476694] [G loss: -0.312157]
```

```
[Epoch 34/50] [Batch 800/938] [D loss: 0.552222] [G loss: -0.669087]
[Epoch 35/50] [Batch 0/938] [D loss: 0.488664] [G loss: -0.391613]
[Epoch 35/50] [Batch 200/938] [D loss: 0.483273] [G loss: -0.462406]
[Epoch 35/50] [Batch 400/938] [D loss: 0.500050] [G loss: -0.426971]
[Epoch 35/50] [Batch 600/938] [D loss: 0.517268] [G loss: -0.512785]
[Epoch 35/50] [Batch 800/938] [D loss: 0.516866] [G loss: -0.395977]
[Epoch 36/50] [Batch 0/938] [D loss: 0.460363] [G loss: -0.456489]
[Epoch 36/50] [Batch 200/938] [D loss: 0.430005] [G loss: -0.384721]
[Epoch 36/50] [Batch 400/938] [D loss: 0.480791] [G loss: -0.205469]
[Epoch 36/50] [Batch 600/938] [D loss: 0.546307] [G loss: -0.262905]
[Epoch 36/50] [Batch 800/938] [D loss: 0.478316] [G loss: -0.362148]
[Epoch 37/50] [Batch 0/938] [D loss: 0.576422] [G loss: -0.544788]
[Epoch 37/50] [Batch 200/938] [D loss: 0.454923] [G loss: -0.350769]
[Epoch 37/50] [Batch 400/938] [D loss: 0.460085] [G loss: -0.416640]
[Epoch 37/50] [Batch 600/938] [D loss: 0.495228] [G loss: -0.439662]
[Epoch 37/50] [Batch 800/938] [D loss: 0.465824] [G loss: -0.487965]
[Epoch 38/50] [Batch 0/938] [D loss: 0.578846] [G loss: -0.283163]
[Epoch 38/50] [Batch 200/938] [D loss: 0.430851] [G loss: -0.251828]
[Epoch 38/50] [Batch 400/938] [D loss: 0.533464] [G loss: -0.449811]
[Epoch 38/50] [Batch 600/938] [D loss: 0.591723] [G loss: -0.604514]
[Epoch 38/50] [Batch 800/938] [D loss: 0.566026] [G loss: -0.412820]
[Epoch 39/50] [Batch 0/938] [D loss: 0.489486] [G loss: -0.443262]
[Epoch 39/50] [Batch 200/938] [D loss: 0.510076] [G loss: -0.541402]
[Epoch 39/50] [Batch 400/938] [D loss: 0.538934] [G loss: -0.378657]
[Epoch 39/50] [Batch 600/938] [D loss: 0.498323] [G loss: -0.364619]
[Epoch 39/50] [Batch 800/938] [D loss: 0.511621] [G loss: -0.522872]
[Epoch 40/50] [Batch 0/938] [D loss: 0.512035] [G loss: -0.342413]
[Epoch 40/50] [Batch 200/938] [D loss: 0.496831] [G loss: -0.408408]
[Epoch 40/50] [Batch 400/938] [D loss: 0.496482] [G loss: -0.322527]
[Epoch 40/50] [Batch 600/938] [D loss: 0.515469] [G loss: -0.447171]
[Epoch 40/50] [Batch 800/938] [D loss: 0.471411] [G loss: -0.252487]
[Epoch 41/50] [Batch 0/938] [D loss: 0.480326] [G loss: -0.298457]
[Epoch 41/50] [Batch 200/938] [D loss: 0.503809] [G loss: -0.433015]
[Epoch 41/50] [Batch 400/938] [D loss: 0.568959] [G loss: -0.630902]
[Epoch 41/50] [Batch 600/938] [D loss: 0.477568] [G loss: -0.432710]
[Epoch 41/50] [Batch 800/938] [D loss: 0.549481] [G loss: -0.342500]
[Epoch 42/50] [Batch 0/938] [D loss: 0.487331] [G loss: -0.545421]
[Epoch 42/50] [Batch 200/938] [D loss: 0.500520] [G loss: -0.419616]
[Epoch 42/50] [Batch 400/938] [D loss: 0.574070] [G loss: -0.647605]
[Epoch 42/50] [Batch 600/938] [D loss: 0.536721] [G loss: -0.312979]
[Epoch 42/50] [Batch 800/938] [D loss: 0.471676] [G loss: -0.249610]
[Epoch 43/50] [Batch 0/938] [D loss: 0.545678] [G loss: -0.582453]
[Epoch 43/50] [Batch 200/938] [D loss: 0.576455] [G loss: -0.524464]
[Epoch 43/50] [Batch 400/938] [D loss: 0.537345] [G loss: -0.569417]
[Epoch 43/50] [Batch 600/938] [D loss: 0.475981] [G loss: -0.390299]
[Epoch 43/50] [Batch 800/938] [D loss: 0.561725] [G loss: -0.636481]
[Epoch 44/50] [Batch 0/938] [D loss: 0.548086] [G loss: -0.358362]
[Epoch 44/50] [Batch 200/938] [D loss: 0.617005] [G loss: -0.720775]
```

```
[Epoch 44/50] [Batch 400/938] [D loss: 0.552139] [G loss: -0.529369]
[Epoch 44/50] [Batch 600/938] [D loss: 0.553388] [G loss: -0.290832]
[Epoch 44/50] [Batch 800/938] [D loss: 0.445265] [G loss: -0.287596]
[Epoch 45/50] [Batch 0/938] [D loss: 0.511456] [G loss: -0.303417]
[Epoch 45/50] [Batch 200/938] [D loss: 0.490483] [G loss: -0.391769]
[Epoch 45/50] [Batch 400/938] [D loss: 0.513716] [G loss: -0.327690]
[Epoch 45/50] [Batch 600/938] [D loss: 0.534981] [G loss: -0.367363]
[Epoch 45/50] [Batch 800/938] [D loss: 0.513150] [G loss: -0.377592]
[Epoch 46/50] [Batch 0/938] [D loss: 0.550523] [G loss: -0.648824]
[Epoch 46/50] [Batch 200/938] [D loss: 0.475971] [G loss: -0.374996]
[Epoch 46/50] [Batch 400/938] [D loss: 0.520287] [G loss: -0.465198]
[Epoch 46/50] [Batch 600/938] [D loss: 0.579003] [G loss: -0.363628]
[Epoch 46/50] [Batch 800/938] [D loss: 0.528247] [G loss: -0.420856]
[Epoch 47/50] [Batch 0/938] [D loss: 0.442094] [G loss: -0.455978]
[Epoch 47/50] [Batch 200/938] [D loss: 0.521366] [G loss: -0.611973]
[Epoch 47/50] [Batch 400/938] [D loss: 0.497671] [G loss: -0.325705]
[Epoch 47/50] [Batch 600/938] [D loss: 0.480964] [G loss: -0.421176]
[Epoch 47/50] [Batch 800/938] [D loss: 0.559000] [G loss: -0.590523]
[Epoch 48/50] [Batch 0/938] [D loss: 0.511802] [G loss: -0.295667]
[Epoch 48/50] [Batch 200/938] [D loss: 0.520438] [G loss: -0.544687]
[Epoch 48/50] [Batch 400/938] [D loss: 0.476892] [G loss: -0.462221]
[Epoch 48/50] [Batch 600/938] [D loss: 0.502219] [G loss: -0.377665]
[Epoch 48/50] [Batch 800/938] [D loss: 0.522829] [G loss: -0.361594]
[Epoch 49/50] [Batch 0/938] [D loss: 0.552367] [G loss: -0.546945]
[Epoch 49/50] [Batch 200/938] [D loss: 0.594879] [G loss: -0.734752]
[Epoch 49/50] [Batch 400/938] [D loss: 0.540251] [G loss: -0.400382]
[Epoch 49/50] [Batch 600/938] [D loss: 0.607716] [G loss: -0.501220]
[Epoch 49/50] [Batch 800/938] [D loss: 0.560425] [G loss: -0.361607]
[Epoch 50/50] [Batch 0/938] [D loss: 0.525257] [G loss: -0.419870]
[Epoch 50/50] [Batch 200/938] [D loss: 0.590902] [G loss: -0.523522]
[Epoch 50/50] [Batch 400/938] [D loss: 0.483355] [G loss: -0.458430]
[Epoch 50/50] [Batch 600/938] [D loss: 0.521077] [G loss: -0.275907]
[Epoch 50/50] [Batch 800/938] [D loss: 0.565887] [G loss: -0.622994]
CPU times: user 13min 55s, sys: 6.23 s, total: 14min 1s
Wall time: 14min 5s
```

[76]:
```python
torch.manual_seed(42)
cexample_z = maybe_cuda(torch.randn(10, latent_dim))
cexample_y = maybe_cuda(torch.randint(0, 10, (10, 1)))
cexample_gen = cgenerator(cexample_z, cexample_y)
```

[77]:
```python
plt.figure(figsize=(20, 4))
for i in range(10):
  plt.subplot(2, 5, i + 1)
  plt.imshow(cexample_gen[i].cpu().detach().reshape(*imshow_shape), cmap='gray')
```



[77]: