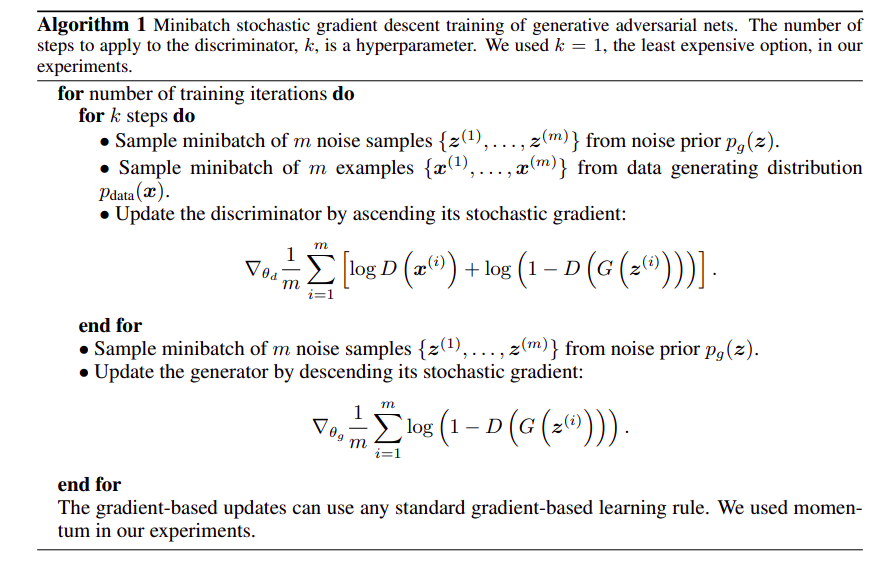
**Simple Gan Implementation**

**Paper : https://arxiv.org/pdf/1406.2661.pdf**

****

| *# -\*- coding: utf-8 -\*-* """ Created on Thu Jun 16:11:54 2021  @author: siddy """  **import** torch **import** torch.nn **as** nn **import** torch.optim **as** optim **import** torchvision **import** torchvision.datasets **as** datasets **from** torch.utils.data **import** DataLoader **import** torchvision.transforms **as** transforms **from** torch.utils.tensorboard **import** SummaryWriter *# to print to tensorboard*   **class** **Discriminator**(nn.Module):  **def** **\_\_init\_\_**(self, in\_features):  super().\_\_init\_\_()  self.disc = nn.Sequential(  nn.Linear(in\_features, 128),  nn.LeakyReLU(0.01),  nn.Linear(128, 1),  nn.Sigmoid(),  )   **def** **forward**(self, x):  **return** self.disc(x)   **class** **Generator**(nn.Module):  **def** **\_\_init\_\_**(self, z\_dim, img\_dim):  super().\_\_init\_\_()  self.gen = nn.Sequential(  nn.Linear(z\_dim, 256),  nn.LeakyReLU(0.01),  nn.Linear(256, img\_dim),  nn.Tanh(), *# normalize inputs to [-1, 1] so make outputs [-1, 1]*  )   **def** **forward**(self, x):  **return** self.gen(x)   *# Hyperparameters etc.* device = "cuda" **if** torch.cuda.is\_available() **else** "cpu" lr = 3e-4 z\_dim = 64 image\_dim = 28 \* 28 \* 1 *# 784* batch\_size = 32 num\_epochs = 50  disc = Discriminator(image\_dim).to(device) gen = Generator(z\_dim, image\_dim).to(device) fixed\_noise = torch.randn((batch\_size, z\_dim)).to(device) transforms = transforms.Compose(  [transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,)),] )  dataset = datasets.MNIST(root="dataset/", transform=transforms, download=**False**)   loader = DataLoader(dataset, batch\_size=batch\_size, shuffle=**True**)  opt\_disc = optim.Adam(disc.parameters(), lr=lr) opt\_gen = optim.Adam(gen.parameters(), lr=lr) criterion = nn.BCELoss()  writer\_fake = SummaryWriter(f"logs/fake") writer\_real = SummaryWriter(f"logs/real") step = 0  **for** epoch **in** range(num\_epochs):  **for** batch\_idx, (real, \_) **in** enumerate(loader):  real = real.view(-1, 784).to(device)  batch\_size = real.shape[0]   *### Train Discriminator: max log(D(x)) + log(1 - D(G(z)))*  noise = torch.randn(batch\_size, z\_dim).to(device)  fake = gen(noise)  disc\_real = disc(real).view(-1)  lossD\_real = criterion(disc\_real, torch.ones\_like(disc\_real))  disc\_fake = disc(fake).view(-1)  lossD\_fake = criterion(disc\_fake, torch.zeros\_like(disc\_fake))  lossD = (lossD\_real + lossD\_fake) / 2  disc.zero\_grad()  lossD.backward(retain\_graph=**True**)  opt\_disc.step()   *### Train Generator: min log(1 - D(G(z))) <-> max log(D(G(z))*  *# where the second option of maximizing doesn't suffer from*  *# saturating gradients*  output = disc(fake).view(-1)  lossG = criterion(output, torch.ones\_like(output))  gen.zero\_grad()  lossG.backward()  opt\_gen.step()   **if** batch\_idx == 0:  print(  f"Epoch [{epoch}/{num\_epochs}] Batch {batch\_idx}/{len(loader)} \  Loss D: {lossD:.4f}, loss G: {lossG:.4f}"  )   **with** torch.no\_grad():  fake = gen(fixed\_noise).reshape(-1, 1, 28, 28)  data = real.reshape(-1, 1, 28, 28)  img\_grid\_fake = torchvision.utils.make\_grid(fake, normalize=**True**)  img\_grid\_real = torchvision.utils.make\_grid(data, normalize=**True**)   writer\_fake.add\_image(  "Mnist Fake Images", img\_grid\_fake, global\_step=step  )  writer\_real.add\_image(  "Mnist Real Images", img\_grid\_real, global\_step=step  )  step += 1 |
| --- |