20.1 Generative Adversarial Networks

```
!pip install d2l==1.0.0-alpha1.post0
```

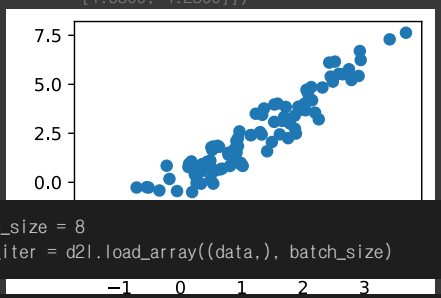
```
Requirement already satisfied: d2l==1.0.0-alpha1.post0 in /usr/local/lib/python3.10/dist-packages (1.0.0a1.post0)
Requirement already satisfied: jupyter in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.0-alpha1.post0) (1.0.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.0-alpha1.post0) (1.23.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.0-alpha1.post0) (3.7.1)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.0-alpha1.post0) (0.1.6)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.0-alpha1.post0) (2.31.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.0-alpha1.post0) (1.5.3)
Requirement already satisfied: gym in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.0-alpha1.post0) (0.25.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from gym->d2l==1.0.0-alpha1.post0) (2.2.1)
Requirement already satisfied: gym-notices>=0.0.4 in /usr/local/lib/python3.10/dist-packages (from gym->d2l==1.0.0-alpha1.post0) (0.0.8)
Requirement already satisfied: notebook in /usr/local/lib/python3.10/dist-packages (from jupyter->d2l==1.0.0-alpha1.post0) (6.5.5)
Requirement already satisfied: qtconsole in /usr/local/lib/python3.10/dist-packages (from jupyter->d2l==1.0.0-alpha1.post0) (5.5.1)
Requirement already satisfied: jupyter-console in /usr/local/lib/python3.10/dist-packages (from jupyter->d2l==1.0.0-alpha1.post0) (6.1.0)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-packages (from jupyter->d2l==1.0.0-alpha1.post0) (6.5.4)
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-packages (from jupyter->d2l==1.0.0-alpha1.post0) (5.5.6)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist-packages (from jupyter->d2l==1.0.0-alpha1.post0) (7.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->d2l==1.0.0-alpha1.post0) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->d2l==1.0.0-alpha1.post0) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->d2l==1.0.0-alpha1.post0) (4.44.3
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->d2l==1.0.0-alpha1.post0) (1.4.5
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->d2l==1.0.0-alpha1.post0) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->d2l==1.0.0-alpha1.post0) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->d2l==1.0.0-alpha1.post0) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->d2l==1.0.0-alpha1.post0) (2.8
Requirement already satisfied: traitlets in /usr/local/lib/python3.10/dist-packages (from matplotlib-inline->d2l==1.0.0-alpha1.post0) (5.7.1)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->d2l==1.0.0-alpha1.post0) (2023.3.post1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->d2l==1.0.0-alpha1.post0) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->d2l==1.0.0-alpha1.post0) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->d2l==1.0.0-alpha1.post0) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->d2l==1.0.0-alpha1.post0) (2023.7
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->d2l==1.0.0-alpha1.
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter->d2l==1.0.0-alpha1.post0)
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter->d2l==1.0.0-alpha1.post0) (7
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter->d2l==1.0.0-alpha1.post0) (6
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter->d2l==1.0.0-alpha1.post0) (6.3
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter->d2l==1.0.0-alpi
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter->d2l==1.0.0-alpi
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from jupyter-console->
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from jupyter-console->jupyter->d2l==1.0.0-alpha1.post0) (2
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.post0) (4.9.3)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.post0) (4
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.post0) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.post0) (0.7.
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.post
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.post0) (3.1
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.post0
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.pos
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.post0)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.post0
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.post0)
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.post0) (5
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.pos
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->d2l==1.0.0-alpha1.post0) (1.2.1)
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter->d2l==1.0.0-alpha1.post0) (23
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter->d2l==1.0.0-alpha1.post0) (23.1
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter->d2l==1.0.0-alpha1.post0)
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter->d2l==1.0.0-alpha1.post0)
```

```
%matplotlib inline
import torch
from torch import nn
from d2l import torch as d2l
```

```
X = torch.normal(0.0, 1, (1000, 2))
A = torch.tensor([[1, 2], [-0.1, 0.5]])
b = torch.tensor([1, 2])
data = torch.matmul(X, A) + b
```

```
d2l.set_figsize()
d2l.plt.scatter(data[:100, (0)].detach().numpy(), data[:100, (1)].detach().numpy());
print(f'The covariance matrix is₩n{torch.matmul(A.T, A)}')
```

```
The covariance matrix is
tensor([[1.0100, 1.9500],
        [1.9500, 4.2500]])
```




```
batch_size = 8
data_iter = d2l.load_array((data,), batch_size)
```

```
net_G = nn.Sequential(nn.Linear(2, 2))
```

```
net_D = nn.Sequential(
    nn.Linear(2, 5), nn.Tanh(),
    nn.Linear(5, 3), nn.Tanh(),
    nn.Linear(3, 1))
```

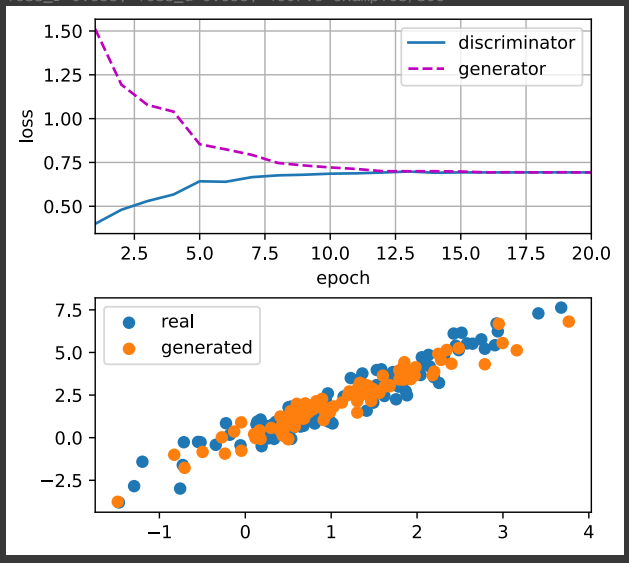
```
def update_D(X, Z, net_D, net_G, loss, trainer_D):
    """Update discriminator."""
    batch_size = X.shape[0]
    ones = torch.ones((batch_size,), device=X.device)
    zeros = torch.zeros((batch_size,), device=X.device)
    trainer_D.zero_grad()
    real_Y = net_D(X)
    fake_X = net_G(Z)
    # Do not need to compute gradient for `net_G`, detach it from
    # computing gradients.
    fake_Y = net_D(fake_X.detach())
    loss_D = (loss(real_Y, ones.reshape(real_Y.shape)) +
              loss(fake_Y, zeros.reshape(fake_Y.shape))) / 2
    loss_D.backward()
    trainer_D.step()
    return loss_D
```

```
def update_G(Z, net_D, net_G, loss, trainer_G):
    """Update generator."""
    batch_size = Z.shape[0]
    ones = torch.ones((batch_size,), device=Z.device)
    trainer_G.zero_grad()
    # We could reuse `fake_X` from `update_D` to save computation
    fake_X = net_G(Z)
    # Recomputing `fake_Y` is needed since `net_D` is changed
    fake_Y = net_D(fake_X)
    loss_G = loss(fake_Y, ones.reshape(fake_Y.shape))
    loss_G.backward()
    trainer_G.step()
    return loss_G
```

```
def train(net_D, net_G, data_iter, num_epochs, lr_D, lr_G, latent_dim, data):
    loss = nn.BCEWithLogitsLoss(reduction='sum')
    for w in net_D.parameters():
        nn.init.normal_(w, 0, 0.02)
    for w in net_G.parameters():
        nn.init.normal_(w, 0, 0.02)
    trainer_D = torch.optim.Adam(net_D.parameters(), lr=lr_D)
    trainer_G = torch.optim.Adam(net_G.parameters(), lr=lr_G)
    animator = d2l.Animator(xlabel='epoch', ylabel='loss',
                            xlim=[1, num_epochs], nrows=2, figsize=(5, 5),
                            legend=['discriminator', 'generator'])
    animator.fig.subplots_adjust(hspace=0.3)
    for epoch in range(num_epochs):
        # Train one epoch
        timer = d2l.Timer()
        metric = d2l.Accumulator(3)  # loss_D, loss_G, num_examples
        for (X,) in data_iter:
            batch_size = X.shape[0]
            Z = torch.normal(0, 1, size=(batch_size, latent_dim))
            metric.add(update_D(X, Z, net_D, net_G, loss, trainer_D),
                       update_G(Z, net_D, net_G, loss, trainer_G),
                       batch_size)
        # Visualize generated examples
        Z = torch.normal(0, 1, size=(100, latent_dim))
        fake_X = net_G(Z).detach().numpy()
        animator.axes[1].cla()
        animator.axes[1].scatter(data[:, 0], data[:, 1])
        animator.axes[1].scatter(fake_X[:, 0], fake_X[:, 1])
        animator.axes[1].legend(['real', 'generated'])
        # Show the losses
        loss_D, loss_G = metric[0]/metric[2], metric[1]/metric[2]
        animator.add(epoch + 1, (loss_D, loss_G))
    print(f'loss_D {loss_D:.3f}, loss_G {loss_G:.3f}, '
          f'{metric[2] / timer.stop():.1f} examples/sec')
```

```
lr_D, lr_G, latent_dim, num_epochs = 0.05, 0.005, 2, 20
train(net_D, net_G, data_iter, num_epochs, lr_D, lr_G,
      latent_dim, data[:100].detach().numpy())
```

```
loss_D 0.693, loss_G 0.693, 1607.6 examples/sec
```



[Readings]

- Generative modeling: label이 없는 large dataset이 주어졌을 때, 이 dataset의 특성을 간결하게 포착하는 모델을 학습한다. 이러한 모델이 주어졌을 때 훈련 데이터의 분포와 유사한 합성 데이터 예시들을 샘플링할 수 있다.
- recurrent neural network language 모델들은 discriminative network의 예시 중에 하나이다. 이것들은 일단 훈련되면 generative 모델처럼 행동할 수 있다.
- GAN은 우리가 fake data와 real data를 분류할 수 없다면, 그 data generator는 좋다는 아이디어에 기반한다.
- GAN은 two-sample test를 이용하여 generative 모델에게 training signal을 준다. 이를 이용하여 real data와 유사한 것을 generate하도록 data generator를 발전시킬 수 있게 한다.
- GAN architecture는 generator network와 discriminator network로 구성된다. 두 개의 network는 서로 대립되는 것으로, generator network는 discriminator network를 속이려고 하고, discriminator network는 새로운 fake data에 적응한다. 이것은 다시 generator network가 발전되게 한다.
- generator는 cross-entropy loss를 maximize하고, disciminator는 cross-entropy loss를 minimize 한다. 이를 minimax game이라 한다.

20.2 Deep Convolutional Generative Adversarial Networks

```
import warnings
import torch
import torchvision
from torch import nn
from d2l import torch as d2l
```

```
d2l.DATA_HUB['pokemon'] = (d2l.DATA_URL + 'pokemon.zip',
                           'c065c0e2593b8b161a2d7873e42418bf6a21106c')

data_dir = d2l.download_extract('pokemon')
pokemon = torchvision.datasets.ImageFolder(data_dir)
```

```
Downloading ../data/pokemon.zip from http://d2l-data.s3-accelerate.amazonaws.com/pokemon.zip...
```

```
batch_size = 256
transformer = torchvision.transforms.Compose([
    torchvision.transforms.Resize((64, 64)),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize(0.5, 0.5)
])
pokemon.transform = transformer
data_iter = torch.utils.data.DataLoader(
    pokemon, batch_size=batch_size,
    shuffle=True, num_workers=d2l.get_dataloader_workers())
```

```
warnings.filterwarnings('ignore')
d2l.set_figsize((4, 4))
for X, y in data_iter:
    imgs = X[:20,:,:,:].permute(0, 2, 3, 1)/2+0.5
    d2l.show_images(imgs, num_rows=4, num_cols=5)
    break
```

```
class G_block(nn.Module):
    def __init__(self, out_channels, in_channels=3, kernel_size=4, strides=2,
                 padding=1, **kwargs):
        super(G_block, self).__init__(**kwargs)
        self.conv2d_trans = nn.ConvTranspose2d(in_channels, out_channels,
                                kernel_size, strides, padding, bias=False)
        self.batch_norm = nn.BatchNorm2d(out_channels)
        self.activation = nn.ReLU()

    def forward(self, X):
        return self.activation(self.batch_norm(self.conv2d_trans(X)))
```

```
x = torch.zeros((2, 3, 16, 16))
g_blk = G_block(20)
g_blk(x).shape
```

```
torch.Size([2, 20, 32, 32])
```

```
x = torch.zeros((2, 3, 1, 1))
g_blk = G_block(20, strides=1, padding=0)
g_blk(x).shape
```

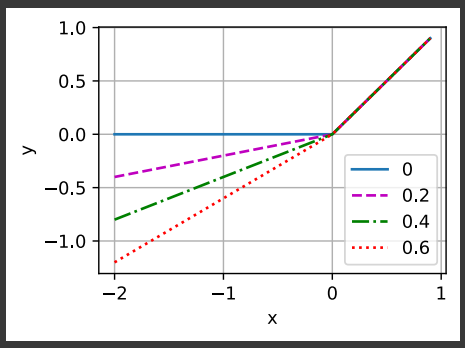
```
torch.Size([2, 20, 4, 4])
```

```
n_G = 64
net_G = nn.Sequential(
    G_block(in_channels=100, out_channels=n_G*8,
            strides=1, padding=0),                  # Output: (64 * 8, 4, 4)
    G_block(in_channels=n_G*8, out_channels=n_G*4), # Output: (64 * 4, 8, 8)
    G_block(in_channels=n_G*4, out_channels=n_G*2), # Output: (64 * 2, 16, 16)
    G_block(in_channels=n_G*2, out_channels=n_G),   # Output: (64, 32, 32)
    nn.ConvTranspose2d(in_channels=n_G, out_channels=3,
                       kernel_size=4, stride=2, padding=1, bias=False),
    nn.Tanh())  # Output: (3, 64, 64)
```

```
x = torch.zeros((1, 100, 1, 1))
net_G(x).shape
```

```
torch.Size([1, 3, 64, 64])
```

```
alphas = [0, .2, .4, .6, .8, 1]
x = torch.arange(-2, 1, 0.1)
Y = [nn.LeakyReLU(alpha)(x).detach().numpy() for alpha in alphas]
d2l.plot(x.detach().numpy(), Y, 'x', 'y', alphas)
```




```
class D_block(nn.Module):
    def __init__(self, out_channels, in_channels=3, kernel_size=4, strides=2,
                 padding=1, alpha=0.2, **kwargs):
        super(D_block, self).__init__(**kwargs)
        self.conv2d = nn.Conv2d(in_channels, out_channels, kernel_size,
                                strides, padding, bias=False)
        self.batch_norm = nn.BatchNorm2d(out_channels)
        self.activation = nn.LeakyReLU(alpha, inplace=True)

    def forward(self, X):
        return self.activation(self.batch_norm(self.conv2d(X)))
```

```
x = torch.zeros((2, 3, 16, 16))
d_blk = D_block(20)
d_blk(x).shape
```

```
torch.Size([2, 20, 8, 8])
```

```
n_D = 64
net_D = nn.Sequential(
    D_block(n_D),  # Output: (64, 32, 32)
    D_block(in_channels=n_D, out_channels=n_D*2),  # Output: (64 * 2, 16, 16)
    D_block(in_channels=n_D*2, out_channels=n_D*4),  # Output: (64 * 4, 8, 8)
    D_block(in_channels=n_D*4, out_channels=n_D*8),  # Output: (64 * 8, 4, 4)
    nn.Conv2d(in_channels=n_D*8, out_channels=1,
              kernel_size=4, bias=False))  # Output: (1, 1, 1)
```

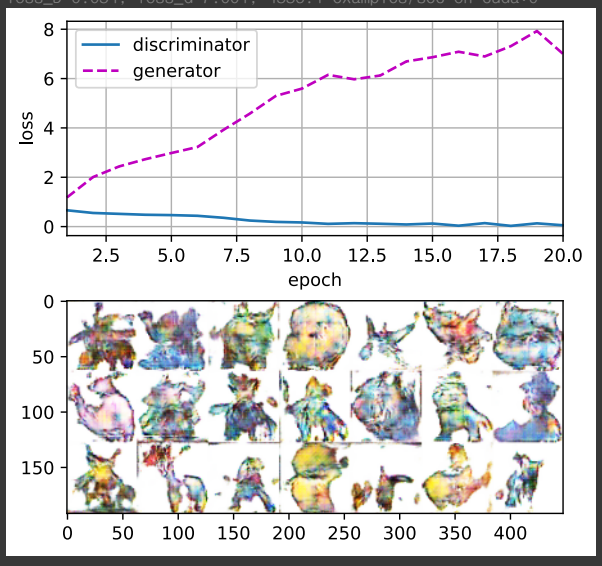
```
x = torch.zeros((1, 3, 64, 64))
net_D(x).shape
```

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

```
def train(net_D, net_G, data_iter, num_epochs, lr, latent_dim,
          device=d2l.try_gpu()):
    loss = nn.BCEWithLogitsLoss(reduction='sum')
    for w in net_D.parameters():
        nn.init.normal_(w, 0, 0.02)
    for w in net_G.parameters():
        nn.init.normal_(w, 0, 0.02)
    net_D, net_G = net_D.to(device), net_G.to(device)
    trainer_hp = {'lr': lr, 'betas': [0.5,0.999]}
    trainer_D = torch.optim.Adam(net_D.parameters(), **trainer_hp)
    trainer_G = torch.optim.Adam(net_G.parameters(), **trainer_hp)
    animator = d2l.Animator(xlabel='epoch', ylabel='loss',
                            xlim=[1, num_epochs], nrows=2, figsize=(5, 5),
                            legend=['discriminator', 'generator'])
    animator.fig.subplots_adjust(hspace=0.3)
    for epoch in range(1, num_epochs + 1):
        # Train one epoch
        timer = d2l.Timer()
        metric = d2l.Accumulator(3)  # loss_D, loss_G, num_examples
        for X, _ in data_iter:
            batch_size = X.shape[0]
            Z = torch.normal(0, 1, size=(batch_size, latent_dim, 1, 1))
            X, Z = X.to(device), Z.to(device)
            metric.add(d2l.update_D(X, Z, net_D, net_G, loss, trainer_D),
                       d2l.update_G(Z, net_D, net_G, loss, trainer_G),
                       batch_size)
        # Show generated examples
        Z = torch.normal(0, 1, size=(21, latent_dim, 1, 1), device=device)
        # Normalize the synthetic data to N(0, 1)
        fake_x = net_G(Z).permute(0, 2, 3, 1) / 2 + 0.5
        imgs = torch.cat(
            [torch.cat([
                fake_x[i * 7 + j].cpu().detach() for j in range(7)], dim=1)
             for i in range(len(fake_x)//7)], dim=0)
        animator.axes[1].cla()
        animator.axes[1].imshow(imgs)
        # Show the losses
        loss_D, loss_G = metric[0] / metric[2], metric[1] / metric[2]
        animator.add(epoch, (loss_D, loss_G))
    print(f'loss_D {loss_D:.3f}, loss_G {loss_G:.3f}, '
          f'{metric[2] / timer.stop():.1f} examples/sec on {str(device)}')

latent_dim, lr, num_epochs = 100, 0.005, 20
train(net_D, net_G, data_iter, num_epochs, lr, latent_dim)
```

```
loss_D 0.054, loss_G 7.001, 4556.1 examples/sec on cuda:0
```



[Readings]

- DCGAN architecture는 Discriminator를 위해 4개의 convolutional layer를 가지고, Generator를 위해서는 4개의 fractionally-strided convolutional layer를 가진다.