print(generated)

**OUTPUT:**

```
romeo: but that kill my heart!

king henry vi:
why, are you so brief?

second murderer:
soft! was ever man so wontmen!

benvolio:
tut, then, i hope, sir, my mistaking sorrow on the sight.

juliet:
o, sir, your cartisfy!

ablisti:
let them call upon you alive,
who in a birthmen to marry warwick as meet,
to
```

**RESULT:** The LSTM model successfully learned character-level language structure and generated Shakespeare-like text with realistic word formations and dialogue styles.

| EXP NO: 10<br>DATE: 11/10/2025 | **Generative Adversarial Network** |
|---|---|

**AIM:** To train a Generative Adversarial Network (GAN) to generate new images from a dataset. Evaluate the quality of the images generated using visual inspection and a quantitative metric like the Inception Score (IS) or Fréchet Inception Distance (FID).

**ALGORITHM:**

- Import required libraries and set hyperparameters.
- Load and preprocess the FashionMNIST dataset (resize and normalize images).
- Define the Generator network using transposed convolutions to create fake images from random noise.
- Define the Discriminator network to distinguish real and fake images.
- Use Binary Cross Entropy loss and Adam optimizer for training.
- Alternately train Discriminator and Generator for each batch.
- After training, generate and save new images from random noise.

**CODE:**

```
import torch, torch.nn as nn, torch.optim as optim

from torchvision import datasets, transforms, utils

from torch.utils.data import DataLoader

from tqdm import tqdm


device = 'cuda' if torch.cuda.is_available() else 'cpu'

latent_dim = 100

batch_size = 128

epochs = 10

img_channels = 1


transform = transforms.Compose([

   transforms.Resize(64),
```

```
    transforms.CenterCrop(64),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])
])

dataset = datasets.FashionMNIST(root='./data', train=True, transform=transform,
download=True)
loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

class Generator(nn.Module):
    def __init__(self, z_dim, img_channels):
        super().__init__()
        self.model = nn.Sequential(
            nn.ConvTranspose2d(z_dim, 512, 4, 1, 0, bias=False),
            nn.BatchNorm2d(512), nn.ReLU(True),
            nn.ConvTranspose2d(512, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256), nn.ReLU(True),
            nn.ConvTranspose2d(256, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128), nn.ReLU(True),
            nn.ConvTranspose2d(128, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64), nn.ReLU(True),
            nn.ConvTranspose2d(64, img_channels, 4, 2, 1, bias=False),
            nn.Tanh()
        )
    def forward(self, z): return self.model(z)

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

```
        super().__init__()
        self.model = nn.Sequential(
            nn.Conv2d(img_channels, 64, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128), nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(128, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256), nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(256, 512, 4, 2, 1, bias=False),
            nn.BatchNorm2d(512), nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(512, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )
    def forward(self, img): return self.model(img).view(-1)

G = Generator(latent_dim, img_channels).to(device)
D = Discriminator(img_channels).to(device)
criterion = nn.BCELoss()
opt_G = optim.Adam(G.parameters(), lr=0.0002, betas=(0.5, 0.999))
opt_D = optim.Adam(D.parameters(), lr=0.0002, betas=(0.5, 0.999))

for epoch in range(epochs):
    for real, _ in tqdm(loader, desc=f"Epoch {epoch+1}/{epochs}"):
        bs = real.size(0)
        real = real.to(device)
        z = torch.randn(bs, latent_dim, 1, 1, device=device)
        fake = G(z)
```