Project Title : Generative Adversarial Network (GAN) for Handwritten Digit Generation (MNIST)

قسمت یک: هدف پروژه

۱. تعریف پروژه

هدف اصلی این پروژه، پیاده‌سازی و آموزش (Generative Adversarial Network – GAN) برای تولید تصاویر واقع‌گرایانه از ارقام دست‌نویس (0 تا 9) است. این مدل‌ها از مجموعه داده‌ی معروف MNIST استفاده می‌کنند که شامل تصاویری از ارقام واقعی نوشته‌شده توسط انسان‌ها است.

۲.ساختار کلی GAN :

یک GAN از دو شبکه‌ی عمیق عصبی تشکیل شده است:

- Generator : یک شبکه‌ی مولد که یک بردار نویز تصادفی را به یک تصویر تولیدی تبدیل می‌کند. هدف آن تولید تصویری است که به‌قدری واقعی به‌نظر برسد که توسط تمییزدهنده قابل تشخیص نباشد.
- Discriminator : یک شبکه‌ی تفکیک کننده که تلاش می‌کند تشخیص دهد یک تصویر داده‌شده واقعی است (از دیتاست MNIST) یا جعلی است (تولید شده توسط مولد).

این دو شبکه در یک بازی رقابتی (Min-Max Game) در برابر یکدیگر آموزش می‌بینند:

- Generator سعی در گمراه‌کردن تمییزدهنده دارد.
- Discriminator کننده سعی دارد در تشخیص تصاویر واقعی از جعلی دقیق باشد.

۳. اهداف عملیاتی پروژه

- درک عمیق از چگونگی تبدیل نویز به تصویر معنادار
- بررسی نقش لایه‌های مختلف در شبکه‌ی مولد برای استخراج ویژگی‌های تصویری
- تحلیل دینامیک رقابت بین Generator و Discriminator در طول آموزش
- ارزیابی کیفیت تصاویر تولیدشده با معیارهای کمی و کیفی نظیر:
- FID (Frechet Inception Distance)
- Inception Score (IS)
- بررسی بصری (Visual Inspection)
- رسم نمودارهای Loss

قسمت دو: مجموعه داده

مجموعه داده‌ی MNIST یکی از شناخته‌شده‌ترین دیتاست‌ها در حوزه یادگیری ماشین و بینایی ماشین است که شامل تصاویر سیاه‌وسفید از ارقام دست‌نویس 0 تا 9 می‌باشد و در این پروژه استفاده شده است.

۱. پیش‌پردازش و آماده‌سازی داده‌ها

برای آماده‌سازی تصاویر جهت آموزش شبکه GAN ، پیش‌پردازش‌های زیر انجام شده است:

>> نرمال‌سازی به بازه [-1, 1]

برای افزایش پایداری آموزش در GAN ، مقدار پیکسل‌ها به بازه [-1 , 1] تبدیل شده‌اند. این کار با ترکیب توابع ToTensor() و Normalize() از PyTorch انجام شده است.

این نرمال‌سازی با خروجی تابع Tanh() در لایه نهایی Generator نیز هماهنگ است (که مقادیر در همین بازه تولید می‌کند).

>> بارگذاری داده

- داده‌ها با استفاده از کلاس torchvision.datasets.MNIST بارگذاری شده‌اند
- گزینه download=True تضمین می‌کند که در صورت عدم وجود، فایل‌ها به‌صورت خودکار دانلود شوند.
- داده‌ها به صورت تصادفی shuffle می‌شوند تا یادگیری بهتر انجام شود.
- اندازه batch با مقدار batch_size = 64 تنظیم شده است.

>> استفاده در فاز ارزیابی

در فایل utils.py ، بخشی از داده‌های واقعی برای محاسبه FID/IS ذخیره می‌شوند.

این مرحله تنها یک‌بار اجرا می‌شود (در evaluate.py)

از این تصاویر واقعی برای مقایسه با تصاویر تولیدی استفاده می‌شود.

قسمت سه: معماری شبکه GAN

شبکه GAN در این پروژه از دو بخش اصلی تشکیل شده است.

- Generator : تولید تصویر از نویز
- Discriminator : تمایز میان تصویر واقعی و جعلی

۱. Generator

تبدیل یک بردار نویز تصادفی ($z \sim N(0,1)$) به یک تصویر 28×28 خاکستری که مشابه تصاویر MNIST باشد.

بر اساس فایل تمرین، ساختار شبکه Generator به شکل زیر است:

Layer No.	Layer Type	Output Shape	Kernel Size	Stride	Number of Kernels	Activation Function	Batch Normalization
1	Dense	(7, 7, 128)	--	--	--	ReLU	Yes
2	Deconvolution	(14, 14, 64)	(4, 4)	2	128	ReLU	Yes
3	Deconvolution	(28, 28, 32)	(4, 4)	2	64	ReLU	Yes
4	Deconvolution	(28, 28, 1)	(4, 4)	1	32	Tanh	No

این معماری مطابق اصول DCGAN طراحی شده است. استفاده از Tanh در لایه خروجی برای تولید مقادیر در بازه [-1 , 1] با نرمال‌سازی ورودی هماهنگ است.

پیاده‌سازی در کد (generator.py):

```
self.model = nn.Sequential(
    nn.Linear(latent_dim, 128 * 7 * 7),
    nn.BatchNorm1d(128 * 7 * 7),
    nn.ReLU(True),
    nn.Unflatten(1, (128, 7, 7)),
    nn.ConvTranspose2d(128, 64, 4, 2, 1),
    nn.BatchNorm2d(64),
    nn.ReLU(True),
    nn.ConvTranspose2d(64, 32, 4, 2, 1),
    nn.BatchNorm2d(32),
    nn.ReLU(True),
    nn.ConvTranspose2d(32, 1, 4, 1, 1),
    nn.Tanh()
)
```

این پیاده‌سازی کاملاً با معماری پیشنهادی تمرین منطبق است.

۲. Discriminator

دریافت یک تصویر و تعیین اینکه آیا تصویر واقعی (از دیتاست MNIST) است یا جعلی (تولید شده توسط Generator).

بر اساس فایل تمرین، ساختار شبکه Discriminator به شکل زیر است:

Layer No.	Layer Type	Output Shape	Kernel Size	Stride	Number of Kernels	Activation Function	Batch Normalization
1	convolution	(14, 14, 64)	(4, 4)	2	64	LeakyReLU (0.2)	No
2	convolution	(7, 7, 128)	(4, 4)	2	128	LeakyReLU (0.2)	Yes
3	convolution	(3, 3, 256)	(4, 4)	2	256	LeakyReLU (0.2)	Yes
4	Flatten	(25633)	--	--	--	--	No
5	Dense	(1024)	--	--	--	--	Yes
6	Dense (Output)	(1)	--	--	--	--	No

پیاده‌سازی در کد (discriminator.py):

```
self.model = nn.Sequential(
    nn.Conv2d(1, 64, 4, 2, 1),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Conv2d(64, 128, 4, 2, 1),
    nn.BatchNorm2d(128),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Conv2d(128, 256, 4, 2, 1),
    nn.BatchNorm2d(256),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Flatten(),
    nn.Linear(256 * 3 * 3, 1024),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Linear(1024, 1),
    nn.Sigmoid()
)
```

این ساختار دقیقاً با ساختار خواسته‌شده در تمرین مطابق است.

قسمت چهار: آموزش مدل (Training Process)

۱. توابع خطا (Loss Functions):

برای آموزش مؤثر شبکه GAN، از Binary Cross Entropy Loss (BCE Loss) برای هر دو شبکه استفاده شده است:

۱.۱ برای برای تفکیک کننده (Discriminator):

هدف: افزایش احتمال تشخیص صحیح تصویر واقعی و کاهش احتمال پذیرش تصویر جعلی.

$$L_D = -\frac{1}{2}\left[\log(D(x)) + \log(1 - D(G(z)))\right]$$

۱.۲ برای مولد (Generator):

$$L_G = -\log(D(G(z)))$$

این فرمول‌ها در کد با استفاده از nn.BCELoss() پیاده‌سازی شده‌اند:

```
criterion = nn.BCELoss()
```

۲. مراحل آموزش:

آموزش هر دو شبکه در هر تکرار (batch) با الگتوریتم زیر انجام می‌شود:

مرحله اول: آموزش Generator

۱. تولید نویز تصادفی:

```
z = torch.randn(batch_size, latent_dim, device=device)
```

۲. تولید تصویر جعلی:

```
gen_images = generator(z)
```

۳. محاسبه loss مولد:

```
g_loss = criterion(discriminator(gen_images), valid)
```

۴. بروزرسانی وزن‌های مولد:

```
optimizer_G.zero_grad()
g_loss.backward()
optimizer_G.step()
```

مرحله دوم: آموزش Discriminator

۱. افزودن نویز گوسی کوچک به ورودی‌ها(برای پایداری آموزش):

```
noisy_real = real_images + 0.05 * torch.randn_like(real_images)
noisy_fake = gen_images.detach() + 0.05 * torch.randn_like(gen_images.detach())
```

۲. محاسبه loss برای تصاویر واقعی و جعلی:

```
real_loss = criterion(discriminator(noisy_real), valid)
fake_loss = criterion(discriminator(noisy_fake), fake)
d_loss = (real_loss + fake_loss) / 2
```

۳. بروزرسانی وزن‌های تمییزدهنده:

```
optimizer_D.zero_grad()
d_loss.backward()
optimizer_D.step()
```

مرحله سوم: تنظیمات بهینه سازی (Optimizer)

از **Adam optimizer** استفاده شده است، مطابق پیشنهاد تمرین و DCGAN :

پارامتر	مقدار
lr	0
β1	0.5
β2	1

```
optimizer_G = optim.Adam(generator.parameters(), lr=..., betas=(0.5, 0.999))
optimizer_D = optim.Adam(discriminator.parameters(), lr=..., betas=(0.5, 0.999))
```

این تنظیمات باعث پایداری بهتر در فرآیند آموزش GAN می‌شوند.

۳. ذخیره‌سازی و تحلیل خروجی‌ها

در دوره‌های مشخص (1، 25، 50، 100):

- ذخیر تصاویر تولید شده:

```
save_generated_images(...)
save_image_grid(...)
```

- ذخیر وزن‌های مدل:

```
torch.save(generator.state_dict(), ...)
```

- رسم نمودار loss برای بازه‌های 1–25، 26–50، 51–100 و نمودار کلی کل دوره‌ها
- محاسبه میانگین loss هر بازه و ذخیره در loss_report.txt

قسمت پنج: معیارهای ارزیابی

معیارهای ارزیابی (Evaluation Metrics)

ارزیابی GAN یک چالش کلیدی است، زیرا کیفیت خروجی صرفاً عددی نیست، بلکه بصری و ادراکی هم هست. در این پروژه، ارزیابی تصاویر تولیدی به‌صورت ترکیبی از روش‌های کمی و کیفی انجام شده است.

۱. ارزیابی بصری (Visual Inspection)

بررسی بصری تصاویر تولیدشده در دوره‌های خاص برای مشاهده:

- ظاهر اولیه نویزها
- ظهور تدریجی ساختارهای عددی
- رسیدن به ارقام کامل و قابل تشخیص

در پروژه:

در دوره‌های 1، 25، 50، 100 تصاویر تولیدشده در قالب grid ذخیره شده‌اند:

```
save_generated_images(gen_samples, epoch)
save_image_grid(fake_images, folder=f"outputs/fake/epoch_{epoch}", prefix="fake")
```

این تصاویر امکان بررسی کیفی کیفیت و تنوع تصاویر را فراهم می‌کنند.

۲. امتیاز Inception Score (IS)

Inception Score میزان کیفیت و تنوع تصاویر تولیدی را ارزیابی می‌کند.

مقدار بالاتر نشان‌دهنده‌ی خروجی با ساختار واضح و تنوع بالا بین کلاس‌های تولیدشده (0 تا 9) است.

در پروژه:

محاسبه با استفاده از torch_fidelity در تابع calculate_fid_is :

```
metrics = calculate_metrics(..., isc=True, fid=True, ...)
is_score = metrics['inception_score_mean']
```

۳. (FID – Frechet Inception Distance)

FID فاصله‌ی آماری بین ویژگی‌های توزیع‌شده‌ی تصاویر واقعی و تولیدشده را اندازه‌گیری می‌کند.

FID < 50 → قابل قبول

FID < 20 → بسیار خوب

FID < 10 → عالی

در پروژه:

```
fid = metrics['frechet_inception_distance']
```

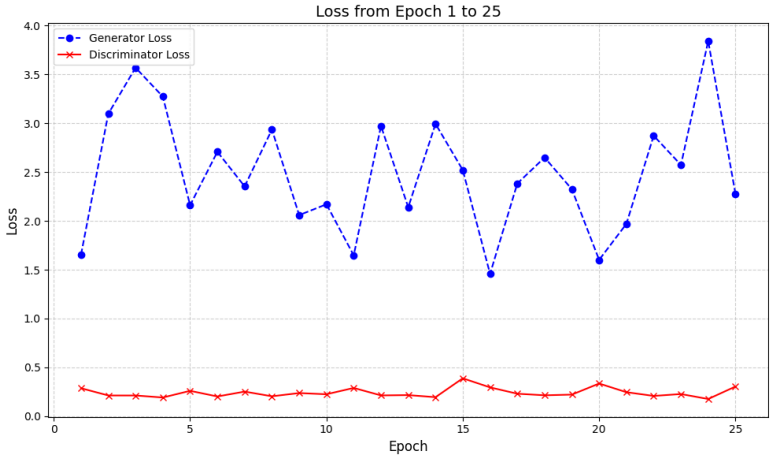
۴. نمودارهای Loss

- برای بررسی پایداری و پیشرفت آموزش:
  نمودارهای loss برای Generator و Discriminator در بازه‌های مشخص
- نمودار کلی از دوره 1 تا 100
- دوره های 25 , 50 , 100

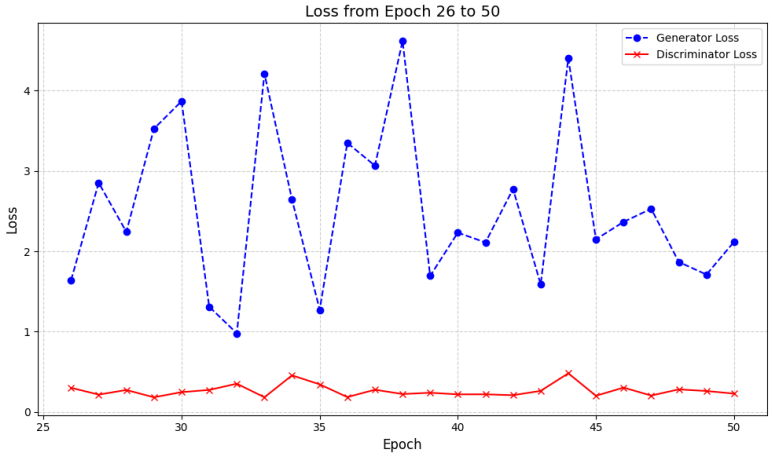
قسمت شش: خروجی‌های پروژه

۱. نمودارهای loss

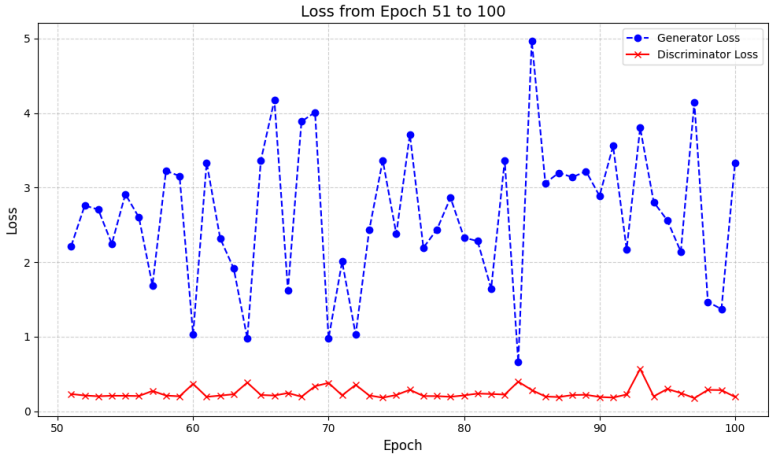
دوره ۱ تا ۲۵ :



دوره ۲۶ تا ۵۰ :



دوره ۵۱ تا ۱۰۰ :



نمودار کلی از دوره ۱ تا ۱۰۰:

۲. میانگین مقدار loss در دوره های مختلف:

- **Epochs 1–25:**

  Generator Loss: 2.4871

  Discriminator Loss: 0.2424

- **Epochs 26–50:**

  Generator Loss: 2.5247

  Discriminator Loss: 0.2661

- **Epochs 51–100:**

  Generator Loss: 2.6333

  Discriminator Loss: 0.2441

۳. خروجی تصاویر

فایل real :

این پوشه شامل تصاویر واقعی از دیتاست MNIST است که برای مقایسه با تصاویر تولیدی استفاده می‌شوند (در محاسبه FID و IS).

چند نمونه از تصاویر real :

فایل fake :

در این پوشه، تصاویر جعلی تولید شده توسط Generator در دوره‌های خاص (1، 25، 50، 100) ذخیره می‌شوند.

برای هر دوره (۱ ، ۲۵ ،۵۰ ، ۱۰۰) یک زیرپوشه ایجاد می‌شود.

کاربرد : مقایسه تصویری با real و تحلیل کیفیت و تنوع خروجی Generator در طول آموزش و ورودی به FID/IS .

چند نمونه از دوره‌های مختلف:

epoch 1:

epoch 25:

epoch 50:

epoch 100:

فایل samples :

فایل در این پوشه، تصاویر تولیدی Generator در دوره‌های خاص به‌صورت تک فایل grid ذخیره می‌شوند (برای بررسی کلی و سریع پیشرفت مدل). هر فایل یک grid از 64 تصویر تولیدشده است.

epoch 1 **epoch 25** **epoch 50** **epoch 100**

۴. تحلیل کیفی تصاویر تولیدی در دوره‌های مختلف آموزش

در این پروژه، تصاویر تولیدشده توسط شبکه Generator در دوره‌های مختلف آموزش (1، 25، 50، و 100) مورد ارزیابی بصری قرار گرفتند. تحلیل پیشرفت این تصاویر نشان می‌دهد که:

در دوره‌های ابتدایی (مثل Epoch 1):

- خروجی‌ها بیشتر شبیه به نویز هستند و فاقد ساختار عددی مشخص‌اند.
- مولد هنوز درک مناسبی از فضای ویژگی تصاویر MNIST ندارد.

در دوره‌های میانی (مثل Epoch 25 و 50):

- ویژگی‌های ساختاری ارقام آغاز به ظهور می‌کنند.
- تنوع عددی بیشتر شده و برخی ارقام قابل شناسایی‌اند.

در دوره 100:

- تصاویر از لحاظ وضوح و ساختار نسبتاً واقعی شده‌اند.
- بسیاری از خروجی‌ها به وضوح شبیه ارقام واقعی MNIST هستند.

با این حال، چرا هنوز مانند تصاویر واقعی نیستند و مقدار تابع زیان Generator هم زیاد است؟

۱. محدودیت در یادگیری توزیع کامل داده‌ها

شبکه مولد تلاش می‌کند توزیع تصاویر تولیدی $p_g(x)$ را به توزیع واقعی داده‌ها $p_{data}(x)$ نزدیک کند، اما به دلیل محدودیت‌های معماری و ظرفیت، این تطابق کامل حاصل نمی‌شود. در نتیجه تصاویر تولیدی ممکن است فقط به نواحی خاصی از فضای ویژگی داده‌های واقعی نزدیک شوند.

۲. احتمال بروز پدیده‌ی Mode Collapse

در برخی موارد، مولد ممکن است تنها چند کلاس عددی را به‌خوبی تولید کرده و کلاس‌های دیگر را نادیده بگیرد. این پدیده که با نام mode collapse شناخته می‌شود، موجب کاهش تنوع و شباهت به توزیع اصلی می‌شود.

۳. ناتوانی در بازسازی جزئیات انسانی

تصاویر MNIST شامل تنوعی از دست‌خط‌های انسانی است (ضخامت قلم، انحناها، سبک نوشتار).

Generator برای بازسازی چنین تفاوت‌های ظریفی به معماری‌های پیچیده‌تری نیاز دارد که فراتر از ظرفیت مدل پایه‌ی فعلی است.

۴. ناپایداری ذاتی در آموزش GAN

فرآیند رقابتی بین Generator و Discriminator اغلب باعث نوسانات در یادگیری می‌شود. هر عدم تعادل بین قدرت این دو شبکه ممکن است منجر به: عدم همگرایی کامل - گیر افتادن در حدهای محلی - تولید تصاویر یکنواخت یا ضعیف شود

بررسی FID و Inception Score (IS)

FID: 40.65

نشان‌دهنده آن است که تصاویر تولیدی دارای ویژگی‌های آماری قابل تشخیص از تصاویر واقعی هستند.

اگرچه ساختار کلی ارقام درست تولید شده، اما هنوز در جزئیات، تنوع، یا پراکندگی آماری با تصاویر دیتاست MNIST تفاوت دارند.

این عدد برای GAN پایه روی MNIST قابل قبول ولی نه ایده‌آل است؛ به‌ویژه اگر بدون تکنیک‌های پیشرفته‌ای مانند Feature Matching یا WGAN-GP آموزش داده شده باشد.

Inception Score (IS): 4.85

این مقدار نشان می‌دهد که تصاویر تولیدی عموماً قابل شناسایی هستند و تنوع مناسبی بین کلاس‌های عددی دیده می‌شود.

با این حال، هنوز میزان اطمینان مدل طبقه‌بند در پیش‌بینی آن‌ها به حد مطلوب نرسیده است (نشان‌دهنده‌ی کمی ابهام در خروجی‌هاست).

برای ارتقاء کیفیت و کاهش FID، می‌توان در از تکنیک‌های زیر استفاده کنید:

- **Conditional GAN (cGAN)** برای کنترل روی کلاس تولیدی
- **Wasserstein GAN with Gradient Penalty (WGAN-GP)** برای بهبود همگرایی
- **Feature Matching** برای افزایش همخوانی آماری
- افزایش ابعاد فضای نهفته یا عمق شبکه‌ها