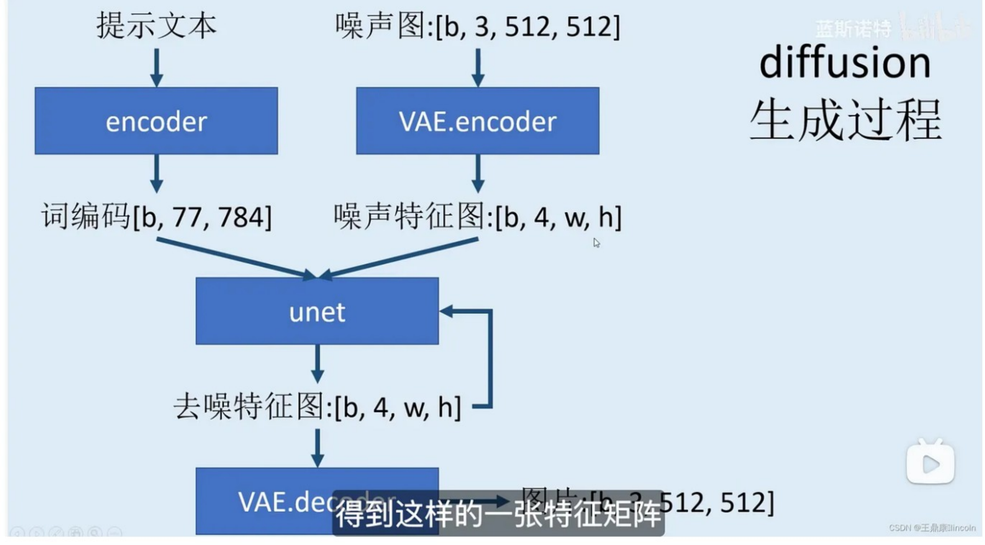
Stable diffusion from Scratch

noplaxochia · Follow
5 min read · Dec 23, 2023

A PyTorch Implementation of Stable Diffusion

> *References:https://www.bilibili.com/video/BV1Mm4y117Ci?p=3&vd_source=5e8e2495f48b394b4e747ad13ae12772*



Structure

```
encoder.requires_grad_(False)
vae.requires_grad_(False)
unet.requires_grad_(True)

encoder.eval()
vae.eval()
unet.train()
```

(Scheduler and train)

```
optimizer = torch.optim.AdamW(unet.parameters(),
                              lr=1e-5,
                              betas=(0.9, 0.999),
                              weight_decay=0.01,
                              eps=1e-8)
criterion = torch.nn.MSELoss()


def train():
    loss_sum = 0
    for epoch in range(400):
        for i, data in enumerate(loader):
            loss = get_loss(data) / 4
            loss.backward()
            loss_sum += loss.item()

            if (epoch * len(loader) + i) % 4 == 0:
                torch.nn.utils.clip_grad_norm_(unet.parameters(), 1.0)
                optimizer.step()
                optimizer.zero_grad()

        if epoch % 10 == 0:
            print(epoch, loss_sum)
            loss_sum = 0

    #torch.save(unet.to('cpu'), 'saves/unet.model')


train()
```

Text Encoder

> *https://github.com/dingkwang/Diffusion_From_Scratch/blob/main/1.encoder.ipynb*

```
encoder = torch.nn.Sequential(
    Embed(),
    ClipEncoder(), # Text Encoder
    ClipEncoder(),
    ClipEncoder(),
    ClipEncoder(),
    ClipEncoder(),
    ClipEncoder(),
    ClipEncoder(),
    ClipEncoder(),
    ClipEncoder(),
    ClipEncoder(),
    ClipEncoder(),
    ClipEncoder(),
    torch.nn.LayerNorm(768),
)
```

Embed layer

- Encode text.
- Reduce dimension
- Position embed.

```
self.embed = torch.nn.Embedding(49408, 768)
self.pos_embed = torch.nn.Embedding(77, 768)
```

ClipTextEncoder

```
__init__():
self.s1 = torch.nn.Sequential(
    torch.nn.LayerNorm(768),
    Atten(),
)

self.s2 = torch.nn.Sequential(
    torch.nn.LayerNorm(768),
    torch.nn.Linear(768, 3072),
)

self.s3 = torch.nn.Linear(3072, 768)
```

Multi-head Self Attention Layer

1. **线性变换**：输入 `x` 经过三个线性层（`self.q`，`self.k`，`self.v`），分别生成查询（query）、键（key）、值（value）。
2. **注意力计算**：计算查询和键的点积，然后应用 softmax 函数来获取注意力权重。这些权重随后用于加权值。
3. **输出**：最后，注意力加权的值通过另一个线性层（`self.out`）生成最终输出。

```
class Atten(torch.nn.Module):

    def __init__(self):
        super().__init__()
        self.q = torch.nn.Linear(768, 768)
        self.k = torch.nn.Linear(768, 768)
        self.v = torch.nn.Linear(768, 768)
        self.out = torch.nn.Linear(768, 768)

    def forward(self, x):
        #x -> [b, 77, 768]

        b = x.shape[0]

        #维度不变
        #[b, 77, 768]
        q = self.q(x) * 0.125
        k = self.k(x)
        v = self.v(x)

        #拆分注意力头
        #[b, 77, 768] -> [b, 77, 12, 64] -> [b, 12, 77, 64] -> [b*12, 77, 64]
        q = q.reshape(b, 77, 12, 64).transpose(1, 2).reshape(b * 12, 77, 64)
        k = k.reshape(b, 77, 12, 64).transpose(1, 2).reshape(b * 12, 77, 64)
        v = v.reshape(b, 77, 12, 64).transpose(1, 2).reshape(b * 12, 77, 64)

        #计算qk乘积
        #[b*12, 77, 64] * [b*12, 64, 77] -> [b*12, 77, 77]
        attn = torch.bmm(q, k.transpose(1, 2))

        #[b*12, 77, 77] -> [b, 12, 77, 77]
        attn = attn.reshape(b, 12, 77, 77)

        #覆盖mask
        def get_mask(b):
            mask = torch.empty(b, 77, 77)
            #上三角的部分置为负无穷
            mask.fill_(-float('inf'))
            #对角线和以下的位置为0
            mask.triu_(1)
            return mask.unsqueeze(1)

        #[b, 12, 77, 77] + [b, 1, 77, 77] -> [b, 12, 77, 77]
        attn = attn + get_mask(attn.shape[0]).to(attn.device)
```

```
        #[b, 12, 77, 77] -> [b*12, 77, 77]
        attn = attn.reshape(b * 12, 77, 77)

        #计算softmax,被mask的部分值为0
        attn = attn.softmax(dim=-1)

        #计算和v的乘积
        #[b*12, 77, 77] * [b*12, 77, 64] -> [b*12, 77, 64]
        attn = torch.bmm(attn, v)

        #[b*12, 77, 64] -> [b, 12, 77, 64] -> [b, 77, 12, 64] -> [b, 77, 768]
        attn = attn.reshape(b, 12, 77, 64).transpose(1, 2).reshape(b, 77, 768)

        #线性输出,维度不变
        #[b, 77, 768]
        return self.out(attn)


Atten()(torch.randn(2, 77, 768)).shape
```

Image encoder/decoder VAE

Image-space Encoder

- Input: Conv
- Down: Resnet
- Mid: Resnet Atten Resnet
- Out: Norm + Conv

Gaussian Sampler

Decoder to Image-space

```
class VAE(torch.nn.Module):

    def __init__(self):
        super().__init__()

        self.encoder = torch.nn.Sequential(
            #in
            torch.nn.Conv2d(3, 128, kernel_size=3, stride=1, padding=1),
            #down
            torch.nn.Sequential(
                Resnet(128, 128),
                Resnet(128, 128),
                torch.nn.Sequential(
                    Pad(),
                    torch.nn.Conv2d(128, 128, 3, stride=2, padding=0),
                ),
            ),
            torch.nn.Sequential(
                Resnet(128, 256),
                Resnet(256, 256),
                torch.nn.Sequential(
                    Pad(),
                    torch.nn.Conv2d(256, 256, 3, stride=2, padding=0),
                ),
            ),
            torch.nn.Sequential(
                Resnet(256, 512),
                Resnet(512, 512),
                torch.nn.Sequential(
                    Pad(),
                    torch.nn.Conv2d(512, 512, 3, stride=2, padding=0),
                ),
            ),
            torch.nn.Sequential(
                Resnet(512, 512),
                Resnet(512, 512),
            ),

            #mid
            torch.nn.Sequential(
                Resnet(512, 512),
                Atten(),
                Resnet(512, 512),
            ),

            #out
            torch.nn.Sequential(
                torch.nn.GroupNorm(num_channels=512, num_groups=32, eps=1e-6),
                torch.nn.SiLU(),
                torch.nn.Conv2d(512, 8, 3, padding=1),
            ),

            #正态分布层
            torch.nn.Conv2d(8, 8, 1),
        )

        self.decoder = torch.nn.Sequential(
            #正态分布层
            torch.nn.Conv2d(4, 4, 1),

            #in
            torch.nn.Conv2d(4, 512, kernel_size=3, stride=1, padding=1),

            #middle
            torch.nn.Sequential(Resnet(512, 512), Atten(), Resnet(512, 512)),

            #up
            torch.nn.Sequential(
                Resnet(512, 512),
                Resnet(512, 512),
                Resnet(512, 512),
                torch.nn.Upsample(scale_factor=2.0, mode='nearest'),
                torch.nn.Conv2d(512, 512, kernel_size=3, padding=1),
            ),
            torch.nn.Sequential(
                Resnet(512, 512),
                Resnet(512, 512),
                Resnet(512, 512),
```

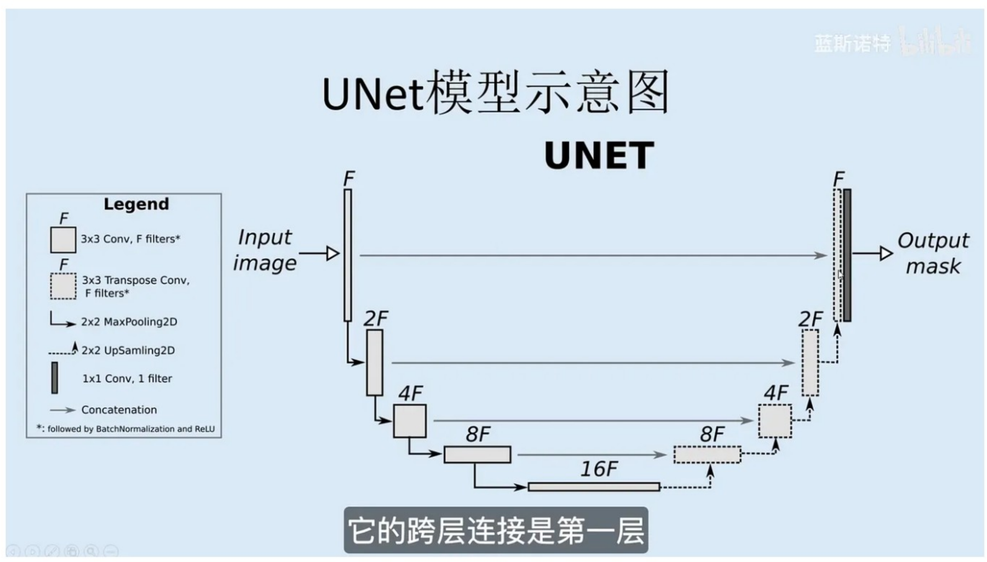
```
                torch.nn.Upsample(scale_factor=2.0, mode='nearest'),
                torch.nn.Conv2d(512, 512, kernel_size=3, padding=1),
            ),
            torch.nn.Sequential(
                Resnet(512, 256),
                Resnet(256, 256),
                Resnet(256, 256),
                torch.nn.Upsample(scale_factor=2.0, mode='nearest'),
                torch.nn.Conv2d(256, 256, kernel_size=3, padding=1),
            ),
            torch.nn.Sequential(
                Resnet(256, 128),
                Resnet(128, 128),
                Resnet(128, 128),
            ),

            #out
            torch.nn.Sequential(
                torch.nn.GroupNorm(num_channels=128, num_groups=32, eps=1e-6),
                torch.nn.SiLU(),
                torch.nn.Conv2d(128, 3, 3, padding=1),
            ),
        )

    def sample(self, h):
        #h -> [1, 8, 64, 64]

        #[1, 4, 64, 64]
        mean = h[:, :4]
        logvar = h[:, 4:]
        std = logvar.exp()**0.5

        #[1, 4, 64, 64]
        h = torch.randn(mean.shape, device=mean.device)
        h = mean + std * h

        return h

    def forward(self, x):
        #x -> [1, 3, 512, 512]

        #[1, 3, 512, 512] -> [1, 8, 64, 64]
        h = self.encoder(x)

        #[1, 8, 64, 64] -> [1, 4, 64, 64]
        h = self.sample(h)

        #[1, 4, 64, 64] -> [1, 3, 512, 512]
        h = self.decoder(h)

        return h
```

Attention layer

- Norm
- Single head self-attention
- Softmax
- V
- atten + residue

UNet

- In
- Down
- Mid
- Up



Text(kv)-image(q) cross Attention

```
(
  #attens
  layer_norm,
  atten,
  layer_norm,
  atten,

  #act,
  layer_norm,
  linear,
  GELU,

  #out
)
```

cross attention

```
class CrossAttention(torch.nn.Module):

    def __init__(self, dim_q, dim_kv):
        #dim_q -> 320
        #dim_kv -> 768

        super().__init__()

        self.dim_q = dim_q

        self.q = torch.nn.Linear(dim_q, dim_q, bias=False)
        self.k = torch.nn.Linear(dim_kv, dim_q, bias=False)
        self.v = torch.nn.Linear(dim_kv, dim_q, bias=False)

        self.out = torch.nn.Linear(dim_q, dim_q)

    def forward(self, q, kv):
        #x -> [1, 4096, 320]
        #kv -> [1, 77, 768]

        #[1, 4096, 320] -> [1, 4096, 320]
        q = self.q(q)
        #[1, 77, 768] -> [1, 77, 320]
        k = self.k(kv)
        #[1, 77, 768] -> [1, 77, 320]
        v = self.v(kv)

        def reshape(x):
            #x -> [1, 4096, 320]
            b, lens, dim = x.shape

            #[1, 4096, 320] -> [1, 4096, 8, 40]
            x = x.reshape(b, lens, 8, dim // 8)

            #[1, 4096, 8, 40] -> [1, 8, 4096, 40]
            x = x.transpose(1, 2)

            #[1, 8, 4096, 40] -> [8, 4096, 40]
            x = x.reshape(b * 8, lens, dim // 8)

            return x

        #[1, 4096, 320] -> [8, 4096, 40]
        q = reshape(q)
        #[1, 77, 320] -> [8, 77, 40]
        k = reshape(k)
        #[1, 77, 320] -> [8, 77, 40]
        v = reshape(v)

        #[8, 4096, 40] * [8, 40, 77] -> [8, 4096, 77]
        #atten = q.bmm(k.transpose(1, 2)) * (self.dim_q // 8)**-0.5

        #从数学上是等价的,但是在实际计算时会产生很小的误差
        atten = torch.baddbmm(
            torch.empty(q.shape[0], q.shape[1], k.shape[1], device=q.device),
            q,
            k.transpose(1, 2),
            beta=0,
            alpha=(self.dim_q // 8)**-0.5,
        )

        atten = atten.softmax(dim=-1)

        #[8, 4096, 77] * [8, 77, 40] -> [8, 4096, 40]
        atten = atten.bmm(v)

        def reshape(x):
            #x -> [8, 4096, 40]
            b, lens, dim = x.shape

            #[8, 4096, 40] -> [1, 8, 4096, 40]
            x = x.reshape(b // 8, 8, lens, dim)

            #[1, 8, 4096, 40] -> [1, 4096, 8, 40]
            x = x.transpose(1, 2)

            #[1, 4096, 320]
            x = x.reshape(b // 8, lens, dim * 8)

            return x

        #[8, 4096, 40] -> [1, 4096, 320]
        atten = reshape(atten)

        #[1, 4096, 320] -> [1, 4096, 320]
        atten = self.out(atten)

        return atten


CrossAttention(320, 768)(torch.randn(1, 4096, 320), torch.randn(1, 77,
                                                                768)).shape
```

Stable Diffusion

Leo

The Journey from Latent Diffusion to Stable Diffusion 3.0

Takeaways

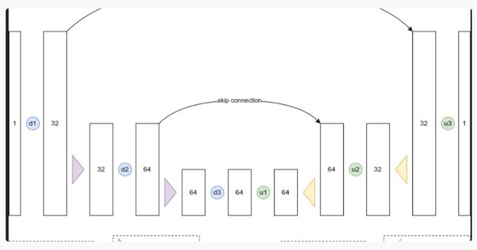
Mar 15 1

Emanuele in Kinomoto.Mag AI

Kolors: A Diffusion Model for Photorealism and Complex Prom…

Kwai-Kolors: discover this new model to generate realistic AI images with unique…

Jul 17 51

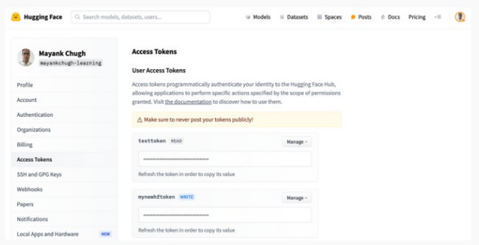


noplaxochia

Diffusion model class

Diffuser's Unet Over Basic unet

Jun 14



Mayankchugh Jobathk

Text-to-Image using Stable-Diffusion-3-Medium: Step-by-Ste…

Stable Diffusion 3 Medium is an advanced text-to-image model known as a Multimodal…

Jul 9

See more recommendations