《模式识别与机器学习A》实验报告

实验题目： 基于生成模型的图像生成

班级： 2203601

学号： 2022113416

姓名： 刘子康

**实验报告内容**

**一、实验目的**

·掌握生成式神经网络的基本原理与结构，掌握搭建和训练生成式神经网络的方法；

·采用任意一种课程中介绍过的或者其它生成式神经网络模型（如变分自编码器等）用于解决图像生成问题，并加深对生成模型的认识；

·理解不同激活函数、dropout比例、数据量和超参数对模型性能的影响。

**二、实验内容**

·参照变分自编码器模型，基于现有框架Pytorch构建一个生成式神经网络，实现数据样本的分类和预测。

·选择MNIST数据集进行训练和测试，该数据集为手写体数字图像标准数据集，包含60000个训练样本和10000个测试样本，每个样本为单通道28\*28像素灰度图像。

·尝试选择不同激活函数，使用dropout等技巧，分析实验结果和可能原因。

·使用不同数据量，不同超参数（如学习率和批次大小），比较实验效果，并给出截图和分析。

**三、实验环境**

·操作系统：Windows 11

·编程语言：Python 3.10

·第三方库：PyTorch 2.4.0+cu118，torchvision0.19.0，Numpy 1.23.4，Matplotlib 3.8.2

·IDE：Pycharm 2022社区版

**四、实验过程、结果及分析**

（包括代码截图、运行结果截图及必要的理论支撑等）

**4.1实验原理**

生成式神经网络可以分为两个主要步骤：

·训练：假设𝑥表示样本，模型从训练集中学习得到的样本数据分布为，以尽可能接近训练数据中的真实分布；

·生成：从中通过采样生成新的样本。

自编码器是一类在半监督学习和非监督学习中使用的人工神经网络，其功能是通过将输入信息作为学习目标，对输入信息进行表征学习。自编码器通常分为两个步骤：

·编码器(encoder) 𝝓：𝑥→z。即将输入样本𝑥转换为压缩的隐含特征表示z；

·解码器(decoder) ψ：𝑧→𝑥’。即将隐含的特征表示z重构为样本𝑥’，𝑥’的维度和𝑥的维度相同，以使𝑥’尽量与原始输入样本𝑥相同。

变分自编码器（Variational auto-encoder，VAE）继承了传统自动编码器的架构，并使用它来学习数据生成分布，这允许我们从潜在空间中随机抽取样本，然后使用解码器网络对这些随机样本进行解码，以生成具有与训练网络的特征类似的特征的独特图像。在传统自编码器中，编码器的输出是一个固定的潜在向量，而VAE引入概率分布的思想，使得潜在空间的表示不再是一个确定的点，而是一个概率分布。

变分自编码器采用一种“重采样”的技巧，即利用标准正态分布来完成采样过程：，从而得到解码器的输入向量z。该公式在反向传播时可以对均值、方差求偏导，这里的ε是从标准正态分布N(0, 1)中采样得到的随机数向量。

**4.2实验过程**

4.2.1数据集预处理

如图4-1，使用Numpy库的random.choice随机选择样本索引，使用torch.utils.data.dataset.Subset创建子数据集，以便设置不同的训练集数据量。

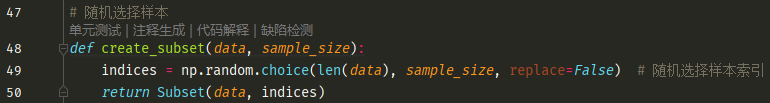


图4-1 随机选取样本

如图4-2，使用Torchvision库的torchvision.datasets.MNIST下载MNIST数据集，并通过torchvision.transform模块进行将图像转换为Tensor格式、展平为一维张量等预处理，以便符合VAE的输入格式。使用PyTorch库的torch.utils.data.DataLoader作为数据集加载器，并设置批次大小。

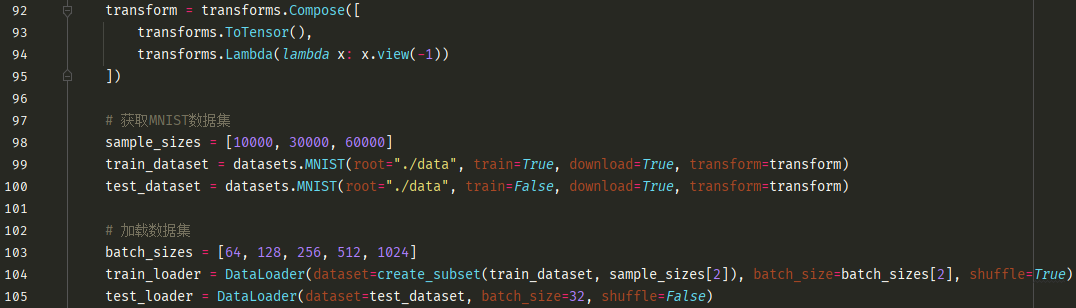


图4-2 获取和加载MNIST数据集

4.2.2 VAE模型搭建

定义一个VAE类，以torch.nn.Module作为基类。

（1）初始化：接收参数为激活函数和dropout比例，按照VAE模型的网络结构依次定义潜在空间维度、编码器和解码器的各个层。



图4-3 VAE模型初始化

（2）编码操作：将输入数据通过全连接层fc1的线性变换和激活函数，映射到隐藏层h1（28\*28=784维降至400维），并进一步经全连接层fc2计算出潜在空间的中间特征表示z的分布均值和方差（对数形式）。

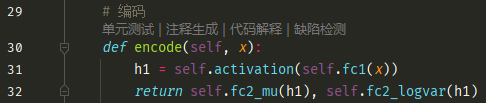


图4-4 编码操作

（3）重参数化操作：计算标准差（），并使用torch.randn\_like()对其添加高斯噪声，返回重参数化后的样本数据分布。



图4-5 重参数化操作

（4）解码操作：将编码的潜在表示z通过全连接层fc3和激活函数，映射到隐藏层h3，并进一步经全连接层fc4的线性变换和Sigmoid函数，映射回原始输入空间（400维升至28\*28=784维），生成新样本。

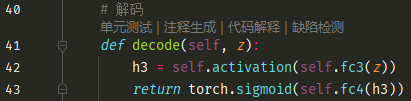


图4-6 解码操作

（5）前向传播：按照编码→重采样→解码的顺序，迭代一轮并输出结果。

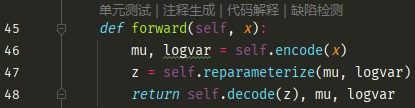


图4-6 前向传播

4.2.3模型训练

如图4-7，分批次进行训练，首先初始化优化器，然后前向传播一次并保存训练结果，最后计算loss并反向传播更新参数。每50个批次输出一次训练进度和loss。

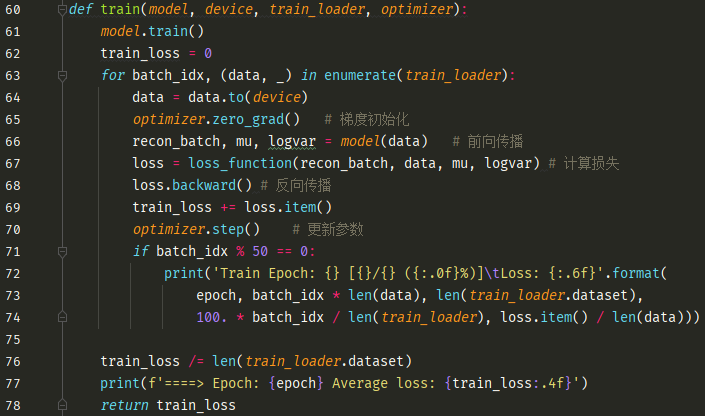


图4-7 模型训练

VAE的损失函数包括重建误差和KL散度。如图4-8，重建误差用于度量生成数据与输入数据的相似度（这里使用交叉熵损失函数），KL散度用于度量z的近似后验分布与先验分布的相似度。

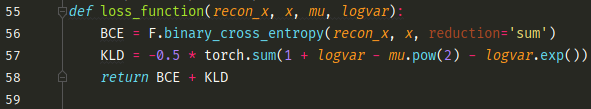


图4-8 损失函数

4.2.4生成图像并绘制曲线

如图4-9，每训练10个epoch，从设定好的先验标准正态分布中采样，利用解码器生成n个新样本图像并保存。训练全部完成后，绘制训练过程的平均loss值变化曲线。

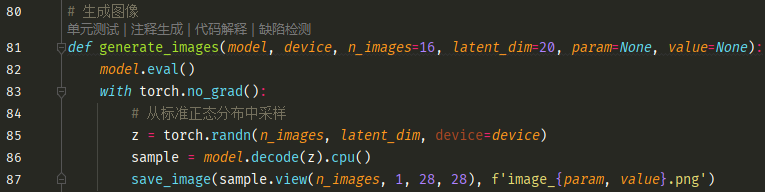


图4-9 生成图像

**4.3实验结果及分析**

4.3.1不同激活函数

分别设置激活函数为ReLU函数、Sigmoid函数、Tanh函数，训练过程loss变化如图4-10所示，不同激活函数生成图像如表4-1所示，根据生成图像清晰和准确程度可以看出最佳激活函数为RuLU。

Sigmoid函数优化稳定，但指数运算的计算复杂度较高，且在深层网络中易出现梯度消失的问题，因此适用于较为简单的网络结构；Tanh函数相比Sigmoid函数优点在于均值为0，不会对梯度产生影响，但仍存在梯度饱和与指数计算的问题；ReLU函数收敛速度更快，计算简单，且不会出现梯度饱和或消失的问题，但可能导致“神经元坏死”。

表4-1 不同激活函数生成图像

|  |  |
| --- | --- |
| 激活函数 | 生成图像 |
| ReLU函数 |  |
| Sigmoid函数 |  |
| Tanh函数 |  |

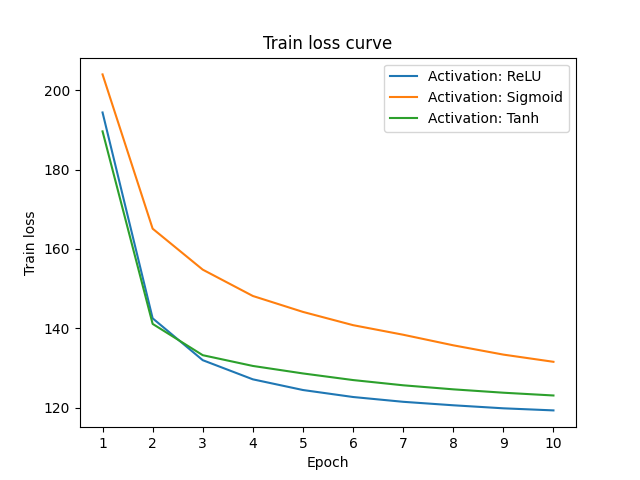


图4-10 不同激活函数训练过程

4.3.2不同dropout比例

分别设置dropout比例为0、0.1、0.2、0.3、0.4、0.5，训练过程loss变化如图4-11所示，不同dropout比例生成图像如表4-2所示，根据生成图像清晰和准确程度可以看到不使用dropout技巧时效果最佳。

dropout技巧主要用于防止模型过拟合，而对应该数据集分类任务，当dropout比例逐渐增大时，loss值逐渐增大。猜测可能是模型并未出现明显过拟合现象，较高的dropout比例反而影响了模型的特征学习。

表4-2 不同dropout比例生成图像

|  |  |
| --- | --- |
| dropout比例 | 生成图像 |
| 0 |  |
| 0.1 |  |
| 0.2 |  |
| 0.3 |  |
| 0.4 |  |
| 0.5 |  |

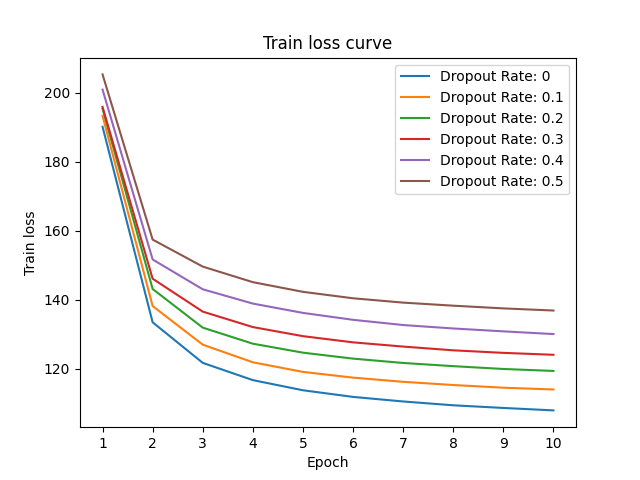


图4-11 不同dropout比例训练过程

4.3.3不同数据量

分别设置训练集数据量为10000、30000、60000，训练过程loss变化如图4-12所示，不同数据量生成图像如表4-3所示，根据生成图像清晰和准确程度可以看到最佳数据量为60000。

在模型提取特征能力足够的前提下，更多的数据样本可以帮助模型更好地学习数据分布和多样化的特征，减少过拟合的风险，并提高模型的泛化能力。

表4-3 不同数据量生成图像

|  |  |
| --- | --- |
| 数据量 | 生成图像 |
| 10000 |  |
| 30000 |  |
| 60000 |  |

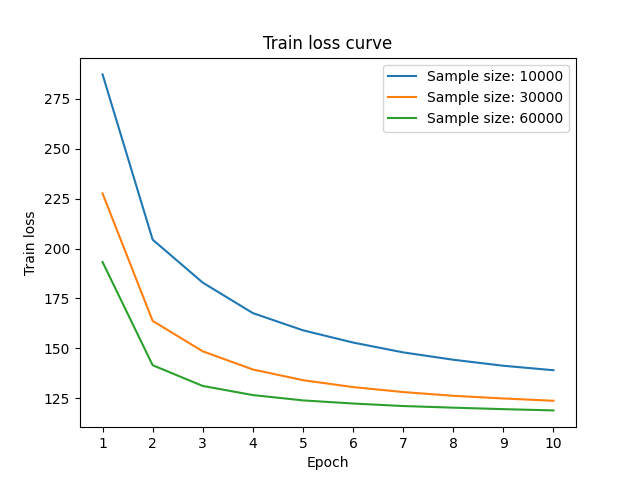


图4-12 不同数据量训练过程

4.3.4不同学习率

分别设置学习率为1e-5、1e-4、1e-3、1e-2、1e-1，训练过程loss变化如图4-13所示，不同学习率生成图像如表4-4所示，根据生成图像清晰和准确程度可以看到最佳学习率为1e-2。

当学习率较低时（如1e-5），模型收敛速度很慢，不能很好地学习到数据特征，故loss值整体较高，且epoch=0时尚未收敛；当学习率为1e-4~1e-2时，随epoch增加，loss值先是迅速下降，之后保持平稳，模型收敛；当学习率为1e-1时，同样先是迅速下降，之后保持平稳，但初始和最终loss值都很大，可能是学习率过大导致参数更新波动较大，模型无法完全收敛。

表4-4 不同学习率生成图像

|  |  |
| --- | --- |
| 学习率 | 生成图像 |
| 1e-5 |  |
| 1e-4 |  |
| 1e-3 |  |
| 1e-2 |  |
| 1e-1 |  |

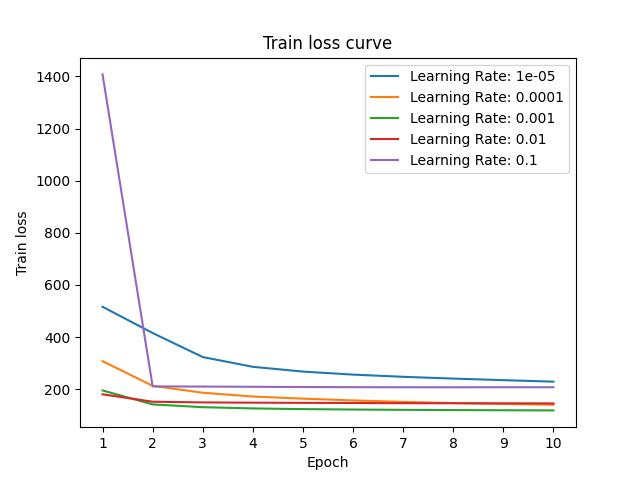


图4-13 不同学习率训练过程

4.3.5不同批次大小

分别设置批次大小为64、128、256、512、1024，训练过程loss变化如图4-14所示，不同批次大小生成图像如表4-5所示，根据生成图像清晰和准确程度可以看到最佳批次大小为128或256。

较小的Batch Size可以加快每轮训练的速度，更好地拟合复杂的数据分布，提高模型精度，且使得训练更加随机化，有助于跳出局部极小值，从而提高最终模型的泛化能力，但由于每次更新都是基于少量样本，也存在着梯度波动较大，收敛速度变慢的问题。

较大的Batch Size可以充分利用现代GPU的强大并行计算能力，加速整体训练过程，且可以获得更稳定的梯度估计，优化过程更加直接地朝向全局极值前进，收敛速度更快，但过大可能会导致陷入局部极小值，影响最终的模型性能。

表4-5 不同批次大小生成图像

|  |  |
| --- | --- |
| 批次大小 | 生成图像 |
| 64 |  |
| 128 |  |
| 256 |  |
| 512 |  |
| 1024 |  |

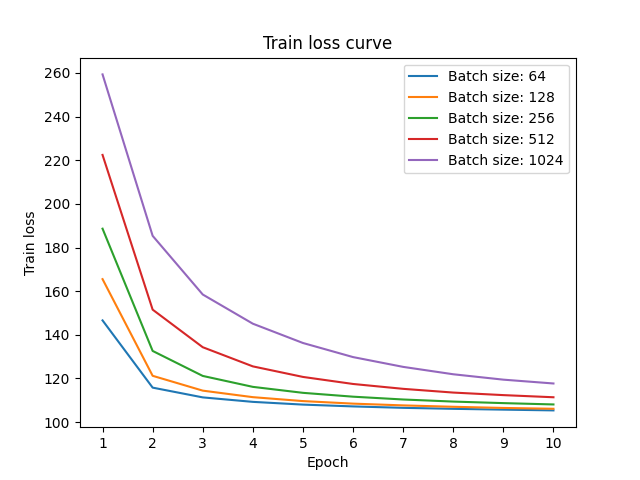


图4-14 不同批次大小训练过程

**五、实验总体结论**

此次实验参照VAE模型，基于PyTorch搭建了一个生成式神经网络，并实现了MNIST数据集的训练和图像生成，数据集的训练和测试，达到了较好的性能和效果。并且使用不同激活函数、dropout比例、数据量和超参数进行训练，分析了其对模型性能的影响。

由实验结果可以得到，当激活函数为ReLU，数据量为60000，学习率为1e-2，批次大小为128或256时，模型达到最佳性能。

常见的生成式神经网络模型包括两大类，即变分自编码器（VAE）和生成式对抗网络（GAN）。VAE通过编码器和解码器的训练过程，学习数据的概率分布，能够进行变分推断，但也存在生成的样本质量较低、训练过程较慢的缺点；而GAN通过生成器和判别器的竞争过程进行训练，生成与真实数据相似的样本，且生成的样本质量较高，缺点是训练过程不稳定，容易出现模型崩溃。

**六、完整实验代码**

1. import torch
2. import torch.nn as nn
3. import torch.optim as optim
4. import torch.nn.functional as F
5. from torch.utils.data import DataLoader
6. from torch.utils.data.dataset import Subset
7. from torchvision import datasets, transforms
8. from torchvision.utils import save\_image
9. import numpy as np
10. import matplotlib.pyplot as plt
11. import time
12. class VAE(nn.Module):
13. def \_\_init\_\_(self, latent\_dim=20, activation=nn.ReLU(), p=0.1):
14. super(VAE, self).\_\_init\_\_()
15. self.latent\_dim = latent\_dim    *# 潜在空间维度*
16. self.activation = activation
17. self.dropout = nn.Dropout(p)
18. *# 编码器*
19. self.fc1 = nn.Linear(28 \* 28, 400)
20. self.fc2\_mu = nn.Linear(400, latent\_dim)
21. self.fc2\_logvar = nn.Linear(400, latent\_dim)
22. *# 解码器*
23. self.fc3 = nn.Linear(latent\_dim, 400)
24. self.fc4 = nn.Linear(400, 28 \* 28)
25. *# 编码*
26. def encode(self, x):
27. h1 = self.activation(self.fc1(x))
28. h1 = self.dropout(h1)
29. return self.fc2\_mu(h1), self.fc2\_logvar(h1)
30. *# 重参数化*
31. def reparameterize(self, mu, logvar):
32. std = torch.exp(0.5 \* logvar)   *# 标准差*
33. eps = torch.randn\_like(std)     *# 高斯噪声*
34. return mu + eps \* std
35. *# 解码*
36. def decode(self, z):
37. h3 = self.activation(self.fc3(z))
38. h3 = self.dropout(h3)
39. return torch.sigmoid(self.fc4(h3))
40. def forward(self, x):
41. mu, logvar = self.encode(x)
42. z = self.reparameterize(mu, logvar)
43. return self.decode(z), mu, logvar
44. *# 随机选择样本*
45. def create\_subset(data, sample\_size):
46. indices = np.random.choice(len(data), sample\_size, replace=False)  *# 随机选择样本索引*
47. return Subset(data, indices)
48. def loss\_function(recon\_x, x, mu, logvar):
49. BCE = F.binary\_cross\_entropy(recon\_x, x, reduction='sum')
50. KLD = -0.5 \* torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
51. return BCE + KLD
52. def train(model, device, train\_loader, optimizer, epoch):
53. model.train()
54. train\_loss = 0
55. for batch\_idx, (data, \_) in enumerate(train\_loader):
56. data = data.to(device)
57. optimizer.zero\_grad()   *# 梯度初始化*
58. recon\_batch, mu, logvar = model(data)   *# 前向传播*
59. loss = loss\_function(recon\_batch, data, mu, logvar) *# 计算损失*
60. loss.backward() *# 反向传播*
61. train\_loss += loss.item()
62. optimizer.step()    *# 更新参数*
63. if batch\_idx % 50 == 0:
64. print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.4f}'.format(
65. epoch, batch\_idx \* len(data), len(train\_loader.dataset),
66. 100. \* batch\_idx / len(train\_loader), loss.item() / len(data)))
67. train\_loss /= len(train\_loader.dataset)
68. print(f'====> Epoch: {epoch} Average loss: {train\_loss:.4f}')
69. return train\_loss
70. *# 生成图像*
71. def generate\_images(model, device, n\_images=16, latent\_dim=20, param=None, value=None):
72. model.eval()
73. with torch.no\_grad():
74. *# 从标准正态分布中采样*
75. z = torch.randn(n\_images, latent\_dim, device=device)
76. sample = model.decode(z).cpu()
77. save\_image(sample.view(n\_images, 1, 28, 28), f'image\_{param, value}.png')
78. if \_\_name\_\_ == "\_\_main\_\_":
79. epoch = 10
80. Loss = []
81. transform = transforms.Compose([
82. transforms.ToTensor(),
83. transforms.Lambda(lambda x: x.view(-1))
84. ])
85. *# 获取MNIST数据集*
86. sample\_sizes = [10000, 30000, 60000]
87. train\_dataset = datasets.MNIST(root="./data", train=True, download=True, transform=transform)
88. *# 加载数据集*
89. batch\_sizes = [64, 128, 256, 512, 1024]
90. train\_loader = DataLoader(dataset=create\_subset(train\_dataset, sample\_sizes[2]), batch\_size=batch\_sizes[1], shuffle=True)
91. *# 部署GPU，创建VAE模型实例*
92. activation = [nn.ReLU(), nn.Sigmoid(), nn.Tanh()]
93. dropout\_p = [0, 0.1, 0.2, 0.3, 0.4, 0.5]  *# 不同dropout比例*
94. device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
95. model = VAE(activation=activation[0], p=dropout\_p[0]).to(device)
96. *# 创建优化器*
97. lrs = [1e-5, 1e-4, 1e-3, 0.01, 0.1]  *# 不同学习率*
98. optimizer = optim.Adam(model.parameters(), lr=lrs[3])
99. *# 模型训练*
100. train\_losses = []
101. print(f"Start Time: {time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())}")
102. for i in range(epoch):
103. train\_loss = train(model, device, train\_loader, optimizer, i)
104. train\_losses.append(train\_loss)
105. *# 生成图像*
106. generate\_images(model, device, n\_images=16, param='sample')
107. print(f"End Time: {time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())}\n")
108. Loss.append(train\_losses)
109. *# 绘制训练损失曲线*
110. x\_ticks = np.arange(1, epoch + 1)
111. plt.plot(x\_ticks, Loss[0])
112. plt.xticks(x\_ticks)
113. plt.xlabel('Epoch')
114. plt.ylabel('Train loss')
115. plt.title('Train loss curve')
116. *# plt.legend()*
117. plt.show()

**七、参考文献**

[1] 刘远超. 深度学习基础: 高等教育出版社, 2023.