每周研究进展阶段汇报

汇报人：杨凯冰
电　邮：tjuykb3022234232@163.com
时间段：2025 年 1 月 25 日 (周六) 至 2025 年 2 月 7 日 (周五)

一、本周工作：
1. 学习 python、pytorch 基础，Softmax 回归和多层感知机基础知识，使用 pytorch 实现交叉熵、梯度下降等基础训练过程，并在代码层面使用 FashionMNIST 和 cifar10 数据集实现多层感知机和 Softmax 回归训练
2. 阅读多模态综述'Visual Generation' 部分
3. 和文轩师兄讨论数据集选择，并协助文轩师兄进行实验。
二、思考总结：

Part 1.

在多分类问题中，采用大余量方法进行分类（选择更置信的识别正确类），但此时输出的 o 是一个向量而不是结果置信度，因此引入 Softmax 分类对输出向量进行处理：

$$\hat{\mathbf{y}} = \mathrm{softmax}(o) \tag{1}$$

$$\hat{y}_i = \frac{\exp(o_i)}{\sum_k \exp(o_k)} \tag{2}$$

将 y 和 $\hat{y}$ 的差值作为交叉熵中的 p 和 q 进行训练:

$$H(\mathbf{p}, \mathbf{q}) = \sum_i -p_i \log(q_i) \tag{3}$$

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log \hat{y}_i = -\log \hat{y}_y \tag{4}$$

$$\partial_{o_i} l(\mathbf{y}, \hat{\mathbf{y}}) = \mathrm{softmax}(\mathbf{o})_i - y_i \tag{5}$$

在思考为什么抛弃传统的 L2 Loss 而使用交叉熵作为 Softmax 损失后，发现在 Softmax 能够将原本全连接层的输出从负无穷到正无穷转换到 0-1 之间，从而能够构建对应的置信度，而且相加和为 1，使得结果更加均分分布。这使得使用对错误惩罚更大（能够获得能大置信度）的交叉熵在比较概率分布比 L2 Loss 直接比较数值差异有着更好的优势。同时由于 L2 Loss 是 one-hot 编码，导致最后的需要多一步归一化，无法直接利用概率分布的特性，无法与 Softmax 协同作用，所以引入 Softmax 回归后，使用 CrossEntropy Loss 作为新的损失函数。

在实现代码过程中，我深切感受到了简单模型训练的基本流程，介于手搓 Softmax 代码较长，放到了 github 上，下面讲解部分重点代码。

```
def softmax(X):
  X_exp = torch.exp(X)
  partition = X_exp.sum(1. keepdim=True)
  return X_exp/partition #使用了广播机制，每一行除以每一行的和

def net(X):
  return softmax(torch.matmul(X.reshape((-1, W.shape[0])), W) +b)
def train_epoch_ch3(net, train_iter, loss, updater):
  if isinstance(net, torch.nn.Module):
    net.train()#开始训练模式
  metric = Accumulator(3)#长度为3的累加器
  for X, y in train_iter:
    y_hat = net(X)
    l = loss(y_hat, y)
    #如果updater是对应的优化器的话
    if isinstance(updater, toch.optim.Optimizer):
      updater.zero_grad()
      #计算梯度
      l.backward()
      #自更新
```

```
            updater.setep()
            metric.add(
                    float(l) * len(y), accuracy(y_hat, y),
                    y.size().numel())
        else:
            l.sum().backward()
            updater(X.shape[0])
            metric.data(float(l.sum()),accuracy(y_hat,y),y.size().numel())
    return metric[0]/metric[2], metric[1]/mertic[2]

    def train(net, train_iter, test_iter, loss, num_epochs, updater):
  #可视化的部分
  animator = Animator(xlabel='epoch', xlim=[1, num_epochs], ylim =[0.3, num_epochs[,
                      legend =['train loss', 'train_acc','test acc'])
  for epoch in range(num_epochs):
    trian_metrics = train_epoch_ch3(net, train_iter, loss, updater)
    test _acc = aevaluate_accuracy(net, test_iter)

    def predict(net, test_iter, n=6):
      #拿出一个样本
      for X,y in test_iter:
        break
      trues = d2l.get_fashion_mnist_labels(y)
      preds = d2l_get_fashion_mnist_labels(net(X).argmax(axis=1))

    predict_ch3(net, test_iter)

    ##############################Softmax通过API优化设计#####################
    #由于Softmax不会直接调整数据形状，所以需要先将数据转化为2D张量的形式
    net = nn.Sequential(nn.Flatten(), nn.Linear(784,10))

    def init_weights(m):
      if type(m) == nn.Linear:
        nn.init.normal_(m.weight, std=0.01)

    net.apply(init_weights)
    loss = nn.CrossEntropyLoss()
    trainer = torch.optim.SCD(net.parameters(), lr=0.1)
    num_epochs =10
    d2l.train(net, train_iter, test_iter, loss, num_epoches,trainer)
```

感知机则是较为朴素久远但重要的二分类结构。初始的单层感知机通过不断学习更新权重和偏置（算法一），能够做到较为简单的二分类问题，这与 Softmax 的 n 分类有着显著不同。

Algorithm 1: 感知机训练过程

Input: 训练集 $(x_1, y_1), \ldots, (x_n, y_n)$
Output: 权重向量 w, 偏置 b
初始化 $w \leftarrow 0, b \leftarrow 0$;
repeat
  for $i = 1$ **to** n **do**
    if $y_i(w^\top x_i + b) \leq 0$ **then**
      $w \leftarrow w + y_i x_i$;
      $b \leftarrow b + y_i$;
    end
  end
until 所有样本正确分类;

从上面的训练算法可以看出，感知机在分类错误的时候会更新权重和偏置（在学习偏置和权重），结合感知机是序列扫描（并行度较差），等于每次处理一个数据，即 $batch1$, 其在更新学习的时候，等价于使用批量大小为 1 的梯度下降，且使用损失函数如下：

$$\ell(y, \mathbf{x}, \mathbf{w}) = \max(0, -y\langle \mathbf{w}, \mathbf{x} \rangle) \tag{6}$$

且其梯度下降不是随机的，而是有序的。

同时，观察结束条件，能够发现其并只有在所有的类分类正确的时候才会结束。这就引发一个思考，能够做到所有的类都分类正确吗？为此查阅感知机有收敛定理：

设所有的数据均在半径 r 内，所有的余量 ρ 分类为两类：

$$y(\mathbf{x}^\top \mathbf{w} + b) \geq \rho \text{对于} \|\mathbf{w}\|^2 + b^2 \leq 1 \tag{7}$$

则感知机能够在 $\frac{r^2+1}{\rho^2}$ 步内收敛。但在学习完该定理后，发现实际中求取数据中的 r 和 ρ 并不实际，即在实际中用时，对于统计上的定理往往不需要关注如何求取，只用了解其能够在有限步内收敛即可，即在代码实现过程中，如果训练步数不高（计算需求不高）时，往往关注结果而不关注精确解。

然而单层感知机由于只是产生线性分割面–无法解决非线性问题如 XOR 问题，故引入多层感知机（将每个感知机的结果进行交互产生新的结果）。且在学习过程中，发现多层感知机相比于单层的通过引入隐藏层、增加激活函数来学习任务中的非线性问题。

输入层: $\mathbf{x} \in \mathbb{R}^n$	输入层: $\mathbf{x} \in \mathbb{R}^n$
隐藏层: $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$	隐藏层: $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
输出层: $\mathbf{w}_2 \in \mathbb{R}^m, b_2 \in \mathbb{R}$	输出层: $\mathbf{W}_2 \in \mathbb{R}^{m \times k}, \mathbf{b}_2 \in \mathbb{R}^k$
$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$	$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$
$o = \mathbf{w}_2^T \mathbf{h} + b_2$	$o = \mathbf{W}_2^T \mathbf{h} + \mathbf{b}_2$
	$y = \text{softmax}(o)$

多层感知机在单层的基础上添加了多个隐藏层（上表中红色部分），例如可以设置以下隐藏层

$$\begin{aligned} \mathbf{h}_1 &= \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \\ \mathbf{h}_2 &= \sigma(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2) \\ \mathbf{h}_3 &= \sigma(\mathbf{W}_3 \mathbf{h}_2 + \mathbf{b}_3) \\ \mathrm{o} &= \mathbf{W}_4 \mathbf{h}_3 + \mathbf{b}_4 \end{aligned} \tag{8}$$

通过设置激活函数（常用 Sigmoid，ReLU，Tanh），可以避免层数塌陷：无论经过多少层，网络仍然等效于一个单层线性层。

具体实现代码过程中，其中权重的设置是随机的。经过尝试，如果一开始全 0 初始化，MNIST 准确率 <0.2，而随机初始化，准确率可以提到 0.8 以上。查阅资料发现，随机初始化能够打破对称性：是的神经元在前行传播计算时有相同的输出，反向传播的时候也有相同的梯度，导致所有神经元同步而导致多层架构失效；同时随机的权重能够让网络学习到不同的特征，能够使梯度流动优化，即能够保证其在适当的范围内波动而不会出现太大太小，保证前向传播的时候方差稳定。而偏置初始化可以为 0 是因为偏置对对称性的影响较小，能够保证 ReLU 神经元在初始的时候就处于激活状态，且对梯度影响不大。

```
num_inputs, num_outputs, num_hiddens = 784, 10, 256
#权重W1和W2的值为随机值，b1和b2的值为0
#第一层输入和隐藏层，则w1有inputs*hiddens个
W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens, requires_grad=True) * 0.01)
#第一层的偏置是对于隐藏层的偏置，所以只用设hiddens个
b1 = nn.Parameter(torch.zeros(num_hiddens, requires_grad=True))
#第二次连接隐藏层和最后一层，所以个数是hiddens*outputs个
W2 = nn.Parameter(torch.randn(num_hiddens, num_outputs, requires_grad=True) * 0.01)
#最后一层的偏置是结果的偏置
b2 = nn.Parameter(torch.zeros(num_outputs, requires_grad=True))
# 输出层偏置
params = [W1, b1, W2, b2]

def relu(X):
    a = torch.zeros_like(X)
    return torch.max(X, a)  # ReLU激活函数实现

def net(X):
    X = X.reshape((-1, num_inputs))          # 展平输入图像
    H = relu(X @ W1 + b1)                    # 隐藏层计算（矩阵乘法用@运算符）
    return (H @ W2 + b2)                     # 输出层计算
```

```
22
23      # 定义损失函数和优化器
24      loss = nn.CrossEntropyLoss()                # 交叉熵损失
25      num_epochs, lr = 10, 0.1
26      updater = torch.optim.SGD(params, lr=lr)    # 随机梯度下降优化器
27
28      # 训练过程（保持与softmax相同接口）
29      d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs, updater)
30
31      ############################多层感知机通过API优化设计####################
32      #和MLP的结果一样
33      #和后面许多的转化更容易
34      import torch
35      from torch import nn
36      from d2l import torch as d2l
37
38      net = nn.Sequential(
39            nn.Flatten(), #输入作为三维
40            nn.Linear(784,256), #第一层是线性层，输入为784，输出位256--暗含第一次隐藏层是
                  256
41            nn.ReLU(), nn.Linear(256,10))#激活函数式ReLU，最后一层256对10--输出10
42
43      #初始化权重
44      #如果是线性层，则对其归一化
45      def init_weights(m):
46          if type(m) == nn.Linear
47            nn.init.normal_(m.weight, std=0)
48
49      net.apply(init_weights);
50
51      #训练过程
52      num_epochs, lr ,batch_size= 10 , 0.1, 256
53      loss = nn.CrossEntropyLoss()
54      trainer = torch.optim.SGD(net.parameters(), lr=lr)
55      train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size, num_epoch))
56      d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs,updater)
```

总结以上如下：

- Softmax 回归
  1. 是一个多分类模型；
  2. 使用 Softmax 将结果转化为每一个类的置信度
  3. 使用交叉熵来衡量预测和实际的区别

- MLP
  1. 二分类模型
  2. 单层感知机训练等价使用批量为 1 的梯度下降模型训练，且无法拟合 XOR 等非线性问题
  3. 多层感知机通过使用隐藏层和激活函数来处理非线性问题，常用激活函数 ReLU，可以通过结合 Softmax 来处理多分类问题
  4.MLP 需要设定隐藏层层数和每层神经元个数超参数，且深的模型比胖的模型更好，更易解决问题

在此学习过程中遇到的疑问并查找资料总结如下：

- 网络模型中，一层视作包含激活函数的一层，即能够学习的层数（包含 w 和 b 的层数）
- 感知机未推广是由于超参数定义不方便，优化不方便（不好调参数），SVM（数学和调节好）和 MLP 的效果差不多一样下选择用数学更好的 SVM
- 在选择层数和层神经元个数时，尽可能选择层数较深的–可视作深度学习（浅的–层数少，单层神经元个数多的网络容易过拟合）深度比宽度好，且模型架构没有最优解
- ReLU 不是线性函数（虽然分开看是线性的，但整体上是非线性的），可以视作是引入了非线性–选择不同的激活函数本质一样，只对梯度计算有影响
- 训练完后参数是固定的，不能是动态的–可能会导致有抖动，使得分类不准确

Part 2.

在视觉生成（Visual Generation）部分，确保生成的视觉内容和人类意图严格对其是十分关键的，而由于开放式文本描述在某些情况下不够精确，尤其是空间参考方面，因此实现视觉对其主要由以下四个关键领域：

- 空间可控 T2I
  结合文本输入和其他条件（如 bounding box）来指导图像生成过程，是用户能够更精确的控制图像中对象的位置和空间排列
- 基于文本的图像编辑
  通过局部修改和全局调整风格来编辑现有图像，以达到对其人类意图的目的
- 文本提示遵循
  提高模型的文本遵循能力
- 视觉概念自定义
  将特定的视觉概念整合到文本输入中，以生产具有特定细节的图像

以 Stable Diffusion 模型作为 T2I 示例，该模型能够通过从随机噪声开始并逐步去噪来生成图像。总结该部分，主要讲述了视觉生成和人类意图对齐的重要性和多种方法来增强 T2I 模型在该方面性能。
Part 3.

恒辉师兄帮忙在 165 服务器上创建账号，配置好环境，并协助文轩师兄跑了两个更改 loss 的实验；和文轩师兄讨论了数据集的选择
三、下周规划：
1. 继续学习 pytorch 的用法，跟着 up 小土堆视频，看完神经网络搭建；
2. 学习卷积神经网络，并用代码实现；
3. 阅读冬暖学长论文列表中大模型综述中'Unified Vision Models' 部分，并进行论文迭代；
3. 和文轩师兄讨论实验部分，并协助跑几个相关实验。

References