模式识别与机器学习 -- 实验2

本实验包含以下部分：

- softmax （50%）
- svm （50%）

softmax

1 手动实现 Softmax 函数 （15%）

```
代码
max_values = logits.max(dim=-1, keepdim=True).values
exp_logits = torch.exp(logits - max_values)
sum_exp = exp_logits.sum(dim=-1, keepdim=True)
probs = exp_logits / sum_exp
return probs
```

说明

找出每一行的最大值；

用原始值减去每一行的最大值后，计算每一行每一个值的exp；

再计算每一行exp之后的和；

将每行exp之后的值，除以对应行exp后和的值，得到结果。

2 创建自定义 Softmax 层 （15%）

```
代码
def forward(self,x):
    return my_softmax(x)

self.network = nn.Sequential(
    nn.Linear(28 * 28, hidden1),
    nn.ReLU(),
    nn.Linear(hidden1, hidden2),
    nn.ReLU(),
    nn.Linear(hidden2, 10),
    MySoftmax()
)

def forward(self, x):
    x = self.flatten(x)
    probs = self.network(x)
    return probs
```

说明

在层封装中，定义前向传播逻辑为自己实现的softmax函数。

在自定义神经网络中，参考标准模型，写出一个三层全连接层的神经网络，在最后加入Softmax层。

在前向传播计算中，参考标准模型对数据进行处理，之后使用自己定义的神经网络进行计算，返回概率。

3 参数调优实验（无需给出代码）

通过参考“训练最佳模型”的代码，我们可以得到不同训练任务的代码模板。

我们需要做的就是在观测某一参数对于训练的影响时，先固定住其他参数，再将相应参数赋值为for循环中变化的变量，就可以得到该参数的不同取值对于训练的影响。

可以通过画图来具象化。

4 提交实验结果，只需截图最后的实验结果汇总和最佳模型的配置即可（20%）

```
实验结果汇总:
       Experiment Parameter Best Val Acc Final Val Acc Final Val Loss
0   Learning Rate    0.0001       0.1006        0.1006         2.2435
1   Learning Rate     0.001       0.6928        0.6928         0.8353
2   Learning Rate      0.01       0.8402        0.8402         0.4523
3   Learning Rate       0.1       0.8818        0.8818         0.3337
4      Batch Size        16       0.8733        0.8576         0.3798
5      Batch Size        32       0.8828        0.8706         0.3528
6      Batch Size        64       0.8796        0.8796         0.3374
7      Batch Size       128       0.8728        0.8728         0.3454
8    Architecture       小网络       0.8780        0.8774         0.3429
9    Architecture       中网络       0.8805        0.8805         0.3315
10   Architecture       大网络       0.8798        0.8795         0.3405
11      Optimizer       SGD       0.8390        0.8368         0.4648
12      Optimizer      Adam       0.8843        0.8758         0.3558

结果已保存到 experiment_summary.csv
```

```
...
    ======================================================================
    训练最佳模型
    ======================================================================

    最佳配置:
      学习率: 0.001
      批次大小: 64
      网络结构: (512, 256)
      优化器: Adam

...
    早停于 epoch 14, step 13500

    最终验证准确率: 0.8937
    模型已保存到 best_model.pth
```

svm

一、损失和梯度的计算

1, 循环实现中的梯度计算（10%）

补全 fduml.linear_svm import中的svm_loss_naive函数

在这里写代码，并简单说明（注意，没有说明会适当扣分）

代码

```
for i in range(num_train):
    scores = X[i].dot(W)
    correct_class_score = scores[y[i]]
    for j in range(num_classes):
        if j == y[i]:
            continue
        margin = scores[j] - correct_class_score + 1  # note delta = 1
        if margin > 0:
            dw[:, j] += X[i]
            dw[:, y[i]] -= X[i]

dW /= num_train
dW += 2 * reg * W
```

说明

对于每一张训练样本i，首先计算其在所有类别上的得分scores = X[i] · W，然后取出正确类得分。对于每一个非正确类别j，计算margin = s_j - s_yi + 1。若margin > 0，说明该错误类违反了安全间隔要求，需要进行梯度更新，具体地：将损失函数对于权重矩阵W求导后发现，对于每一个样本i，其中每一个违反间隔的错误类j，对于对应权重列的梯度贡献为+X[i]（dW[:, j] += X[i]）；对于正确类权重列的梯度贡献为-X[i]（dW[:, y[i]] -= X[i]）。

遍历完所有样本与所有类别后，对梯度取平均并加上L2正则项的导数2·reg·W，即得到最终梯度。

2, 向量实现中的损失计算和梯度计算（15%）

补全 fduml.linear_svm import中的svm_loss_vectorized函数

在这里写代码，并简单说明

代码

损失计算：

```
    scores = X.dot(W)

    num_train = X.shape[0]
    correct_scores = scores[np.arange(num_train), y]
    correct_scores = correct_scores[:, np.newaxis]

    margins = scores - correct_scores + 1
    margins[np.arange(num_train), y] = 0
    margins = np.maximum(0, margins)

    loss = np.sum(margins) / num_train
```

```
    loss += reg * np.sum(W * W)
```

梯度计算：

```
    scores = X.dot(W)

    num_train = X.shape[0]
    correct_scores = scores[np.arange(num_train), y]
    correct_scores = correct_scores[:, np.newaxis]

    margins = scores - correct_scores + 1
    margins[np.arange(num_train), y] = 0
    margins = np.maximum(0, margins)

    binary = (margins > 0).astype(float)
    row_sum = np.sum(binary, axis=1)
    binary[np.arange(num_train), y] = -row_sum

    dW = X.T.dot(binary) / num_train
    dW += 2 * reg * W
```

说明

对于损失计算，矩阵乘法一次性计算所有样本在所有类别上的得分，通过高级索引取出每行对应的正确类分数，并广播相减，得到所有margin初步值。将正确类位置的margin强制置0（不惩罚自己），再执行max(0, margins)将所有违反间隔的类保留，对所有剩余错误类地margin求和后除以N并加上L2正则项，即得最终损失。

对于梯度计算，构建一个投票矩阵binary，初始时将违反间隔的类别设为1，其余设为0，之后通过求和得到每一个样本中所有违反类的总数，将正确类处减去该总数。这样，binary[i,j]就统计出了dw[p,j]需要变化的X[i,p]的数量，又因为dw初始化为全0，则dw[p,j] = binary[i,j] * X[i,p] for i = 0 to N,恰好符合矩阵乘法定义，则dW = X.T @ binary。最后除以样本数N并加上正则化梯度2·reg·W，得到最终的梯度。

3, 在这里提交ipynb中的相关检查结果（不占额外分数，但这是判断上面的实现是否正确的重要依据

朴素方法梯度计算检查：

```
numerical: 34.072127 analytic: 34.078060, relative error: 8.705712e-05
numerical: -5.975755 analytic: -5.975755, relative error: 7.060030e-11
numerical: 5.995035 analytic: 5.995035, relative error: 5.140901e-11
numerical: 9.907292 analytic: 9.908156, relative error: 4.361374e-05
numerical: 5.826547 analytic: 5.830336, relative error: 3.250939e-04
numerical: -26.476294 analytic: -26.476294, relative error: 3.902843e-11
numerical: -11.161025 analytic: -11.161025, relative error: 1.483257e-11
numerical: -53.350381 analytic: -53.332142, relative error: 1.709649e-04
numerical: -3.071166 analytic: -3.071166, relative error: 4.748555e-10
numerical: 12.656534 analytic: 12.652825, relative error: 1.465565e-04
numerical: 18.307093 analytic: 18.307093, relative error: 4.177052e-11
numerical: 20.636195 analytic: 20.632329, relative error: 9.366816e-05
numerical: -7.164993 analytic: -7.171831, relative error: 4.769199e-04
numerical: 10.697625 analytic: 10.697625, relative error: 1.175761e-11
numerical: -44.344063 analytic: -44.344063, relative error: 5.399326e-12
numerical: -3.174007 analytic: -3.174007, relative error: 1.481318e-10
numerical: 19.059625 analytic: 19.056726, relative error: 7.607028e-05
numerical: 26.577599 analytic: 26.581626, relative error: 7.575402e-05
numerical: -27.089163 analytic: -27.089163, relative error: 1.794163e-11
numerical: 10.030195 analytic: 10.033607, relative error: 1.700816e-04
```

向量化损失计算检查：

```
Naive loss: 8.949716e+00 computed in 0.560514s
Vectorized loss: 8.949716e+00 computed in 0.013702s
difference: 0.000000
```

向量化梯度计算检查：

```
Naive loss and gradient: computed in 0.573740s
Vectorized loss and gradient: computed in 0.013498s
difference: 0.000000
```

二、实现SGD

代码+简单说明（10%）

代码

采样：

```
    indices = np.random.choice(num_train, batch_size, replace=True)
    X_batch = X[indices]
    y_batch = y[indices]
```

参数更新：

```
    self.W -= learning_rate * grad
```

预测标签：

```
    scores = X.dot(self.W)
    y_pred = np.argmax(scores, axis=1)
```

说明

对于采样，使用提示中的函数生成索引，从训练数据中采样并存储到对应的数组中。

对于参数更新，按照梯度下降的公式W = W - learning rate * grad对W进行更新。

对于预测标签，每次都计算全部样本在所有类别上的得分，对每一个样本取最大值的索引，即为预测类别值。

三、利用验证集做超参数调优

表格记录（5*5），可以自己尝试不同的组合 （10%）

学习率 \ 正则化强度	10000	25000	50000	75000	100000
0.00000001	{0.178}	{0.174}	{0.189}	{0.173}	{0.22}
0.00000005	{0.244}	{0.296}	{0.338}	{0.347}	{0.348}
0.0000001	{0.296}	{0.363}	{0.367}	{0.349}	{0.342}
0.0000005	{0.345}	{0.345}	{0.341}	{0.328}	{0.319}
0.000001	{0.299}	{0.333}	{0.277}	{0.263}	{0.252}

最优的结果的loss曲线截图和正确率截图（5%）