人工智能基础 实验二

PB19071501 李平治

I. 实验环境

- 2GHz 4vIntel Core i5
- macOS 12.31
- Python 3.6.2, requirements:
  - PyTorch 1.8.1
  - TorchVision 0.9.1
  - TorchAudio 0.8.1
  - NumPy 1.22.3

II. 传统机器学习

2.1 决策树

1. 实验结果

```
DecisionTree acc: 75.00%
```

2. 代码说明

- 决策树节点类

```python
class DecisionNode():
    def __init__(self, feature, value, true_branch, false_branch, threshold):
        self.feature_idx = feature
        self.value = value
        self.true_branch = true_branch
        self.false_branch = false_branch
        self.threshold = threshold
```

- 决策树节点划分metric: 信息增益

```python
def calc_info_gain(l, l1, l2):
    def calc_entropy(y):
        unique_labels = np.unique(y)
        entropy = 0
        for label in unique_labels:
            _p = len(l[l == label]) / len(l)
            entropy += - _p * log2(_p)
        return entropy

    p = len(l1) / len(l)
    info_gain = calc_entropy(l) - p * calc_entropy(l1) - (1 - p) * calc_entropy(l2)
    return info_gain
```

用交叉熵来计算l分裂成l1和l2的信息增益

- 决策树分裂:

```python
for feature_idx in range(num_features):
    feature_vals = np.expand_dims(features[:, feature_idx], axis=1)
    unique_vals = np.unique(feature_vals)
    for threshold in unique_vals:
        fl1 = np.array([s for s in features_labels if s[feature_idx] >= threshold])
        fl2 = np.array([s for s in features_labels if s[feature_idx] < threshold])
        if len(fl1) > 0 and len(fl2) > 0:
            l1 = fl1[:, num_features:]
            l2 = fl2[:, num_features:]
            info_gain = calc_info_gain(labels, l1, l2)
            if info_gain > largest_info_gain:
                largest_info_gain = info_gain
                best_feature_idx = feature_idx
                best_threshold = threshold
                best_sets = (fl1, fl2)
```

自顶向下递归地建立决策树。当递归到某个节点时，遍历该节点处剩余的每个feature，计算分裂后的信息增益，选取其中的最大值作为该节点的分裂方式。

2.2 支持向量机

1. 实验结果

```
SVM(Linear kernel) acc: 63.33%
SVM(Poly kernel) acc: 93.33%
SVM(Gauss kernel) acc: 86.67%
```

2. 代码说明

- 计算Kernel矩阵

```python
num_samples, num_features = train_data.shape
kernel_matrix = np.zeros((num_samples, num_samples))
# computing kernel matrix
for i in range(num_samples):
for j in range(num_samples):
  kernel_matrix[i, j] = self.KERNEL(train_data[i], train_data[j])
```

- 构造二次规划矩阵，利用cvxopt进行凸优化

```python
P = cvxopt.matrix(np.outer(train_label, train_label) * kernel_matrix, tc='d')
q = cvxopt.matrix(np.ones(num_samples) * -1)
A = cvxopt.matrix(train_label, (1, num_samples), tc='d')
b = cvxopt.matrix(0, tc='d')
G = cvxopt.matrix(np.vstack((np.identity(num_samples) * -1,
np.identity(num_samples))))
h = cvxopt.matrix(
     np.vstack((cvxopt.matrix(np.zeros(num_samples)),
cvxopt.matrix(np.ones(num_samples) * self.C))))
cvxopt.solvers.options['show_progress'] = False
minimization = cvxopt.solvers.qp(P, q, G, h, A, b)
```

- 利用优化结果计算Support Vector

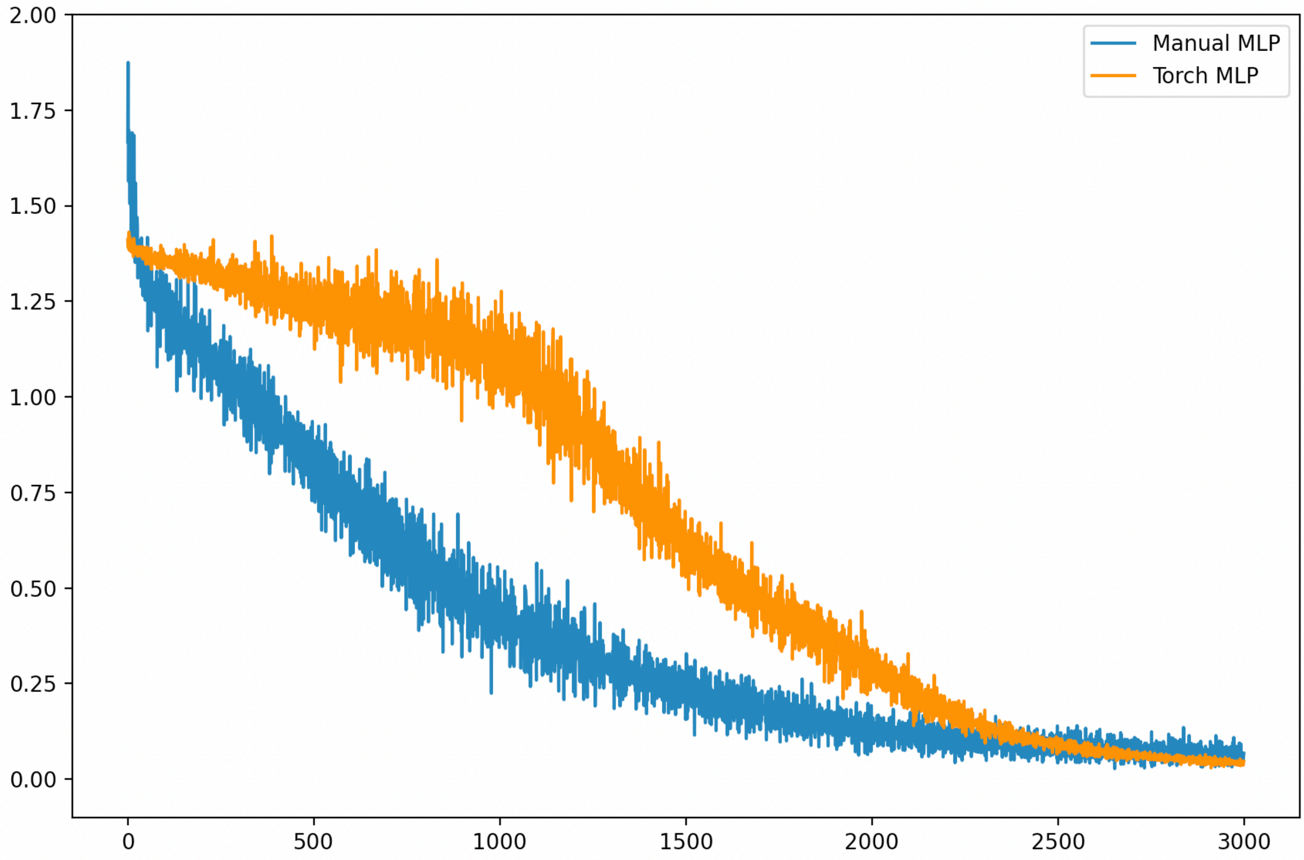
```python
self.SV = train_data[idx]
self.SV_alpha = x[idx]
self.SV_label = train_label[idx]

self.b = self.SV_label[0]
for i in range(len(self.SV_alpha)):
    self.b -= self.SV_alpha[i] * self.SV_label[i] * self.KERNEL(self.SV[i],
self.SV[0])
```

III. 深度学习

1. 实验结果

- 手动梯度下降MLP和PyTorch SGD优化的训练loss随时间变化:



- 最终Weight和Bias矩阵:

```
Layer 0 weights:
[[ 0.71373035 -0.14031091 -1.22833925 -1.05971415  0.23197974 -1.36548472
   1.10723941  0.01184267  0.2762316  -0.17720831]
 [ 1.07201889 -0.90048664 -0.2797492  -0.0362148   0.55110194 -0.8250343
  -0.6621626  -0.52860441 -0.01509795  0.71106389]
 [-0.0314159   1.29494303  1.29345379  0.06860662  0.25708299 -0.65133536
  -0.6999699  -0.01662199  0.00876799  0.43711671]
 [-0.41538996 -0.19637986 -1.39046448 -0.68142634 -0.83639086 -0.77620864
  -1.13811798  0.3352585   0.73443531  0.23509683]
 [-0.59145627  0.19022218 -0.55830424  1.18653668  0.1267212  -0.00432356
   0.03854422  0.97213278  0.53651101  1.35447032]
 [ 0.60170082 -0.01275607 -0.77999961  0.20334991 -0.49158491  0.16005377
   0.54048761  0.99605436 -0.3465038   0.79611359]
 [-0.27738149  0.32076467  0.23958467  0.3860778   0.31143597  0.09639461
   0.62998939  0.88811348  0.95130668 -0.27157077]
 [-1.10201143  0.06848426 -0.45724747  0.2224386   0.06824301 -1.08968259
   0.04681725  0.57139095  0.49725576 -0.1757426 ]
 [-0.2170296  -0.13048129  0.34254864  0.31899847  0.7007105  -0.48353367
   0.46204869  0.41091423 -0.02232778  0.24134772]
```

```
 [ 0.38564192  0.48573589  0.06783026 -0.74310775  0.37566666  0.09160413
   0.12785705 -0.12785469 -0.5541416   0.58659853]]
Layer 0 bias:
[[ 0.70967959 -0.34486275 -0.18745722 -0.30144904 -0.11190529 -0.13014961
  -0.3090695  -0.03084662 -0.25859487  0.61736607]]
Layer 1 weights:
[[-0.395834    0.45729931  1.37017437  0.5254712  -0.55148535  0.07887536
   0.28752868 -0.55885535]
 [-0.65436311  0.04535324 -0.94552995 -0.34587684  0.38720653 -0.34922973
   0.24859145 -1.0281667 ]
 [ 0.7345047  -1.04769039  1.31484343 -0.23989031  0.22150873 -0.38116993
   1.0232681   0.74546457]
 [-1.11854796  1.45644972  0.84507112  0.00449973 -0.31333921  0.50082672
   0.20777519 -0.33771897]
 [-1.08956433  0.65844698  0.2772536  -0.44407401  0.47201683 -0.69539268
   0.95736914 -0.39632955]
 [ 0.50162812 -0.05927347  1.27281565  0.57783057  0.38447288  0.35104057
   0.62987503  0.31858833]
 [ 0.28146659 -0.23324366 -1.1482497  -1.73069808  0.73883151  1.21319605
   0.91751322  0.28450752]
 [-0.08760837  0.24089345 -0.03321411 -1.40818747 -0.52533527 -0.16178151
   0.01591185 -0.47043789]
 [ 0.22888259 -0.83745442  0.37730192  0.14901383  1.15403613 -1.07303555
  -0.49026357  0.40601436]
 [ 1.5085927  -0.67580446 -0.79954812 -0.09358566  0.87978643 -0.60332018
  -0.06302326  0.04477567]]
Layer 1 bias:
[[-0.07928198 -0.1103097  -0.50769125  0.03037476 -0.21251321 -0.21864107
   0.0121647  -0.52367457]]
Layer 2 weights:
[[-0.94403443  1.23029462 -0.86753894  1.29832689 -0.66562278 -0.41991861
   0.8919153   0.91926763]
 [ 0.35730557 -0.71551597  1.22194264 -0.42213453  1.56335366  0.58432913
   0.23875559 -1.33761593]
 [ 0.52775366 -0.12524943  0.96464407  1.24644082  0.0754051   0.40156549
  -1.44996109  2.24571438]
 [-0.62207318 -1.19745006 -1.17400298 -1.06207725 -0.55421339  1.31262698
  -0.08830852 -1.28693551]
 [ 0.21223018 -0.04579365 -1.23276717  0.88640074 -0.022396   -0.4062726
   1.03852099  0.65678288]
 [ 1.03846011  0.33186102  1.0352534  -0.24728323 -0.89653311  1.32076883
  -0.07065747  0.15209968]
 [-0.52565519  0.04154617 -0.50348028 -0.10605509  0.79172268  0.66163628
  -0.14218047 -0.94024049]
 [-0.90064422  0.18113216 -1.34070867  0.81115339 -0.14728617  0.14306045
  -0.12744113 -0.4968613 ]]
Layer 2 bias:
[[-0.08997688  0.12861772  0.01979176  0.07590009 -0.23034299 -0.40386966
   0.15902587  0.27343204]]
Layer 3 weights:
[[-0.82483127  1.01402963  0.34427833 -0.53909552]
 [ 1.76691092 -0.84279838 -1.10451799  0.14489597]
```

```
 [-2.11402814  1.7305078   1.10187896 -1.4852654 ]
 [ 2.68570098  0.51783799 -1.24341607 -0.7304413 ]
 [-0.02752349 -1.64392124  0.70468014  0.46732968]
 [ 0.670869   -0.69234533  0.53733591  0.42579972]
 [-0.67206958  0.13297239 -0.2989807   1.79937365]
 [ 1.38864963  2.00707777 -1.47585715 -1.97054739]]
Layer 3 bias:
[[-0.06864368  0.30088591 -0.54513421  0.31289198]]
```

2. 代码说明

- 定义线性层、激活层和损失函数

```python
class LinearLayer:
    """
    Linear layer of MLP network
    """
    def __init__(self, input_size, output_size):
        self.input_size = input_size
        self.output_size = output_size
        self.last_input = None
        self.weights = np.random.randn(input_size, output_size) * np.sqrt(2/input_size)
        self.bias = np.zeros((1, self.output_size))
        self.training = False

    def forward(self, x):
        if self.training:
            self.last_input = x
        return np.dot(x, self.weights) + self.bias

    def backward(self, grad_y):
        if self.last_input is None:
            raise RuntimeError("No input to back-propagate through")
        grad_bias = np.sum(grad_y, axis=0, keepdims=True)
        grad_weights = np.matmul(self.last_input.T, grad_y)
        grad_x = np.matmul(grad_y, self.weights.T)
        return grad_x, self.weights, grad_weights, self.bias, grad_bias

    def train(self):
        self.training = True

    def eval(self):
        self.training = False
```

```python
class Tanh:
    """
    Thah layer of MLP network
    """
```

```
    def __init__(self):
        self.last_input = None
        self.is_training = False

    def forward(self, x):
        if self.is_training:
            self.last_input = x
        return np.tanh(x)

    def backward(self, grad_y):
        return (1 - np.power(np.tanh(self.last_input), 2)) * grad_y, None, None, None, None

    def train(self):
        self.is_training = True

    def eval(self):
        self.is_training = False
```

```
class SoftmaxCrossEntropy:
    """
    Softmax cross entropy loss layer of MLP network
    """
    def __init__(self):
        pass

    def __call__(self, scores, labels):
        scores = np.exp(scores - np.max(scores, axis=1, keepdims=True))
        scores /= np.sum(scores, axis=1, keepdims=True)
        positive_scores = scores[np.arange(batch_size), labels]
        loss = np.mean(-np.log(positive_scores))

        one_hot = np.zeros_like(scores)
        one_hot[np.arange(batch_size), labels] = 1.0
        grad = (scores - one_hot) / batch_size
        return loss, grad
```

其中forward()用于前向传播，输入为上一层的输入值，输出为经过该网络层运算后的值；backward()用于反向传播梯度，输入为后一层计算出的梯度，利用self.last_input储存的上一次前向传播通过的输出，来计算本层的梯度和损失值。

- MLP网络的反向传播

```
class MLP:
    # ......
    def backward(self, input, label, lr):  # 自行确定参数表
        # 反向传播
        weights_list = []
        grad_weights_list = []
        bias_list = []
        grad_bias_list = []
```

```python
        pred = self.forward(input)
        loss, grad = self.loss(pred, label)
        for l in reversed(self.layers):
            grad, weights, grad_weights, bias, grad_bias = l.backward(grad)
            if weights is not None:
                weights_list.append(weights)
                grad_weights_list.append(grad_weights)
                bias_list.append(bias)
                grad_bias_list.append(grad_bias)
        for w, gw, b, gb in zip(weights_list, grad_weights_list, bias_list, grad_bias_list):
            w -= gw * lr
            b -= gb * lr
        return loss
    # ......
```

利用前向传播计算出预测结果，并用SoftmaxCrossEntropy计算出损失和最后一次层梯度，之后从后往前迭代计算梯度，最后进行更新

3.2 CNN

1. 实验结果

```
Train Epoch: 0/5 [0/50000]      Loss: 2.300163
Train Epoch: 0/5 [12800/50000]  Loss: 2.210129
Train Epoch: 0/5 [25600/50000]  Loss: 2.050327
Train Epoch: 0/5 [38400/50000]  Loss: 2.011160
Train Epoch: 1/5 [0/50000]      Loss: 1.777975
Train Epoch: 1/5 [12800/50000]  Loss: 1.912966
Train Epoch: 1/5 [25600/50000]  Loss: 1.880835
Train Epoch: 1/5 [38400/50000]  Loss: 1.880602
Train Epoch: 2/5 [0/50000]      Loss: 1.716251
Train Epoch: 2/5 [12800/50000]  Loss: 1.868620
Train Epoch: 2/5 [25600/50000]  Loss: 1.674133
Train Epoch: 2/5 [38400/50000]  Loss: 1.879710
Train Epoch: 3/5 [0/50000]      Loss: 1.695350
Train Epoch: 3/5 [12800/50000]  Loss: 1.565757
Train Epoch: 3/5 [25600/50000]  Loss: 1.747636
Train Epoch: 3/5 [38400/50000]  Loss: 1.783090
Train Epoch: 4/5 [0/50000]      Loss: 1.685986
Train Epoch: 4/5 [12800/50000]  Loss: 1.579239
Train Epoch: 4/5 [25600/50000]  Loss: 1.760924
Train Epoch: 4/5 [38400/50000]  Loss: 1.692889
Finished Training
Test set: Average loss: 1.6487   Acc 0.44
Test set: Average loss: 1.6913   Acc 0.43
```

2. 代码说明

- 网络结构

```python
class MyNet(nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        self.conv_layer = nn.Sequential(nn.Conv2d(3, 8, (9, 9)),
                                        nn.ReLU(),
                                        nn.MaxPool2d(2),
                                        nn.Conv2d(8, 4, (3, 3)),
                                        nn.ReLU(),
                                        nn.MaxPool2d(2))
        self.cls_layer = nn.Sequential(nn.Linear(100, 108),
                                       nn.ReLU(),
                                       nn.Linear(108, 72),
                                       nn.ReLU(),
                                       nn.Linear(72, 10),
                                       nn.ReLU())

    def forward(self, x):
        x = self.conv_layer(x)
```

```python
        x = x.view(x.size(0), -1)
        x = self.cls_layer(x)
        return x
```

总结

- 通过本次实验，具体实现了决策树、SVM传统机器学习模型和MLP、CNN深度学习模型，加深了对这四种模型的理解，对模型训练中的参数有了更深刻的认识，同时也训练了Python编码能力和对PyTorch框架的使用。
- 本次实验用时：
  - 决策树: 2h
  - SVM: 3h
  - MLP: 1.5h
  - CNN: 0.5h
  - 实验报告: 1h