

模式识别实验报告

专业：	吴健雄学院
学号：	61821313
年级：	大二
姓名：	王梓豪

签名：

时间：

实验一 数据降维与分类任务

1 问题描述

分别利用 PCA 和 LDA 降维技术对葡萄酒数据进行降维处理，在降维后的数据集上训练和测试 logistic 回归分类器，并比较降维技术前后分类器准确率的变化。

2 实现步骤与流程

PCA 降维

求解样本的散布矩阵

$$\mathbf{S}(\mathbf{x}) = \sum_{i=1}^n (\mathbf{x}_i - \boldsymbol{\mu}_x)(\mathbf{x}_i - \boldsymbol{\mu}_x)^T$$

求解散布矩阵最大的 β 个特征值对应的特征向量作为基向量进行投影

$$\mathbf{S}(\mathbf{x})\mathbf{w}_i = \lambda_i \mathbf{w}_i \quad i = 1, 2, \dots, k$$

投影后的样本特征向量

$$\mathbf{y} = \mathbf{W}^T \mathbf{x} \Rightarrow \mathbb{R}^\beta = \mathbb{R}^{\beta \times \alpha} \mathbb{R}^\alpha$$

其中变换矩阵 \mathbf{W} 的列向量即为散布矩阵的特征向量 \mathbf{w}_i

LDA 降维

求解类内散布矩阵

$$\mathbf{S}_w(\mathbf{x}) = \sum_{i=1}^c \mathbf{S}_i = \sum_{i=1}^c \sum_{\mathbf{x} \in \mathcal{X}_i} (\mathbf{x} - \boldsymbol{\mu}_i)(\mathbf{x} - \boldsymbol{\mu}_i)^T$$

求解类间散布矩阵

$$\mathbf{S}_b(\mathbf{x}) = \sum_{i=1}^c n_i (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T$$

如果类内散布矩阵可逆，投影的 β 个基向量满足

$$\mathbf{S}_w^{-1} \mathbf{S}_b \mathbf{w}_i = \lambda_i \mathbf{w}_i \quad i = 1, 2, \dots, k$$

并且对应分别对应最大的 β 个特征值 λ_i 。如果类内散布矩阵不可逆，可以将其替换为

$$\mathbf{S}_w \leftarrow \mathbf{S}_w + \epsilon \mathbf{I}_\beta, \quad \epsilon > 0$$

投影后的样本特征向量

$$\mathbf{y} = \mathbf{W}^T \mathbf{x} \Rightarrow \mathbb{R}^\beta = \mathbb{R}^{\beta \times \alpha} \mathbb{R}^\alpha$$

其中变换矩阵 \mathbf{W} 的列向量即为矩阵 $\mathbf{S}_w^{-1} \mathbf{S}_b$ 的特征向量 \mathbf{w}_i

logistic 回归

logistic 回归模型

$$\hat{y} = \sigma(z) = \sigma(\mathbf{w}^T \mathbf{x} + b)$$

其中 $\sigma(\cdot)$ 代表 sigmoid 函数，对于 logistic 回归可以使用两种损失函数

- 交叉熵损失

$$\ell_{cross-entropy} = - \sum_{i=1}^n \left[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \right]$$

- 负对数似然

$$\ell_{log-likelihood} = \sum_{i=1}^m \left[-y_i \hat{y}_i + \log(1 + e^{\hat{y}_i}) \right]$$

本次实验中采取前者进行实现。

3 实验结果与分析

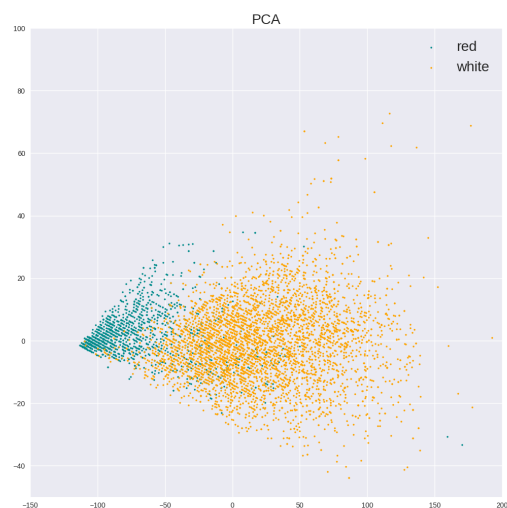
由于实验要求的 MindSpore 框架相关的资料较为匮乏，因此本次实验采取开源机器学习算法库 sklearn 与实验中的算法进行对比。sklearn 是一个开源的基于 python 语言的机器学习工具包。它通过 numpy, scipy 等 python 数值计算的库实现高效的算法应用，并且涵盖了几乎所有主流机器学习算法。

PCA 降维

经过实验算法以及 sklearn 算法的 PCA 降维后的样本如下图所示，可以看出，降维后的样本具有较高的散布程度。并且样本经过实验的 PCA 算法降维后与 sklearn 的 PCA 算法相比拥有相同的性态，但二者在某一特征维度上呈镜像对称。



(a) PCA (实验)



(b) PCA (sklearn)

图 1: PCA 降维效果图

LDA 降维

由于 LDA 限制降维维度不大于类别数减 1，为了达到可视化的效果，本次实验仍然将样本降维到二维经过实验算法降维后的样本如下图所示，可以看出，降维后的样本具有相对较好的可分性。但无法展示 sklearn 的 LDA 降维效果图



图 2: LDA 降维效果图

3.1 logistic 回归

实验中实现的 logistic 回归在原始数据集、PCA 降维数据集以及 LDA 降维数据集上的训练过程如后图所示，分别展示了模型训练过程中损失函数以及分类准确率的变化。

从图像中可以看出，原始数据集的模型准确率居中，PCA 降维后的数据集的模型准确率较差，LDA 降维后的数据集的模型准确率最佳。

相应地，sklearn 提供的 logistic 回归模型在各个数据集上的准确率如下图所示，与实验算法相同的是，LDA 的准确率最高，原始数据次之，PCA 最低。

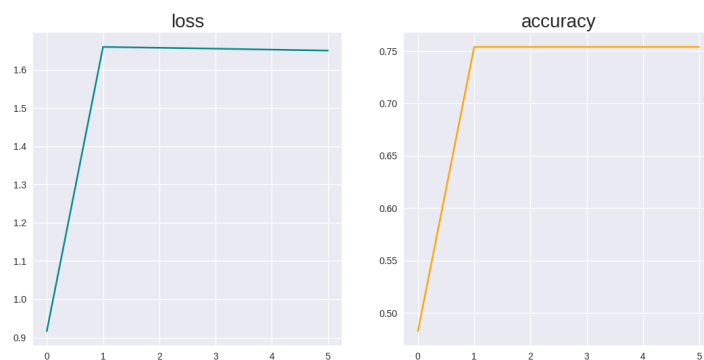
4 MindSpore 学习使用心得体会

由于本次实验并未采用 MindSpore，此部分用 sklearn 的学习使用心得体会来代替。

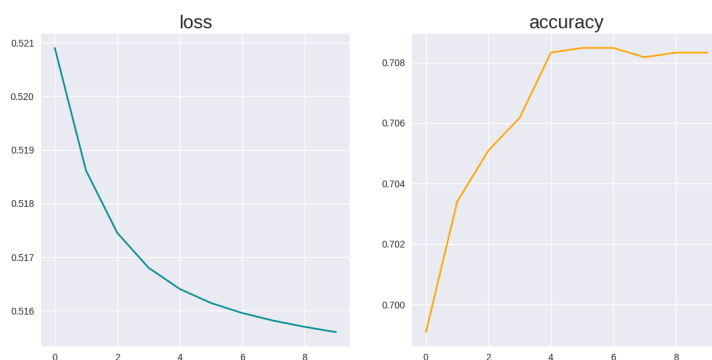
本次实验中调用了 sklearn 中的以下算法接口

- PCA (decomposition PCA)
- LDA (discriminant_analysis LinearDiscriminantAnalysis)
- logistic 回归 (linear_model LogisticRegression)

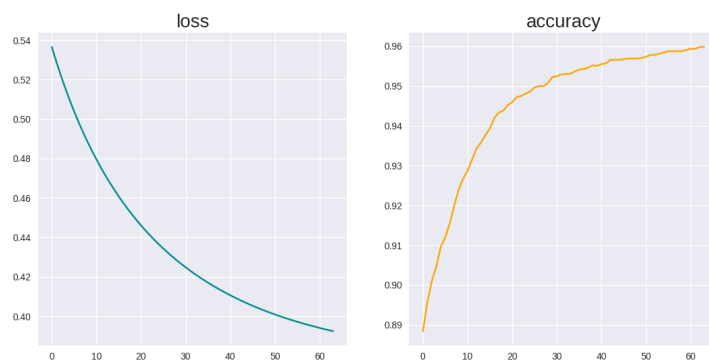
作为实验算法的对照算法，sklearn 提供的算法接口方便调用，并且执行效率较高，能够高效地实现实验的需求。



(a) 训练历史 (原始)



(b) 训练历史 (PCA)



(c) 训练历史 (LDA)

图 3: logistic 回归 (实验)

```

• (python) wzh@wzh:~/workspace/course/pattern recognition/实验/实验1$
origin data accuracy: 0.9795290133907958
PCA data accuracy: 0.9307372633523164
LDA data accuracy: 0.9947668154532862

```

图 4: logistic 回归 (sklearn)

5 代码附录

```
mine.py M X
实验1 > src > mine.py > ...
63     axs[0].plot(history['loss'], color='darkcyan')
64     axs[1].plot(history['acc'], color='orange')
65     plt.show()
66
67 if __name__ == '__main__':
68     import warnings
69     warnings.filterwarnings('ignore')
70
71     plt.style.use('seaborn')
72
73     # ***** origin ***** #
74
75     wine_data = MyWineData(args['dataroot'])
76     model, history = wine_data.classify(
77         lr=0.01,
78         decay=0.97,
79         iterations=32,
80         patience=5
81     )
82     wine_data.render_history(history)
83
84     # ***** PCA ***** #
85
86     wine_data = MyWineData(args['dataroot'])
87     wine_data.decompose(PCA, 2)
88     wine_data.render_data(
89         'PCA',
90         (-150, 200),
91         (-100, 50)
92     )
93     model, history = wine_data.classify(
94         lr=0.001,
95         decay=0.97,
96         iterations=64,
97         patience=5
98     )
99     wine_data.render_history(history)
100
101     # ***** LDA ***** #
102
103     wine_data = MyWineData(args['dataroot'])
104     wine_data.decompose(LDA, 2)
105     wine_data.render_data(
106         'LDA',
107         (-30, 20),
108         (-10, 5)
109     )
110     model, history = wine_data.classify(
111         lr=0.01,
112         decay=0.97,
113         iterations=64,
114         patience=5
115     )
116     wine_data.render_history(history)

logistic.py M X
实验1 > src > logistic.py > ...
5
6 class LogisticRegression(object):
7
8     def __init__(self, input_dim, scale=0.1) -> None:
9         self.w = scale * np.random.randn(input_dim)
10        self.b = scale * np.random.randn()
11
12    def predict(self, data, logit=True):
13        y_pred = np.matmul(data, self.w)
14        y_pred = y_pred + self.b
15        if logit:
16            y_pred = sigmoid(y_pred)
17        else:
18            y_pred = y_pred > 0
19        return y_pred
20
21    def fit(
22        self,
23        data,
24        label,
25        lr,
26        decay,
27        iterations,
28        patience,
29        epsilon=0.001
30    ):
31        history = dict()
32        history['loss'] = []
33        history['acc'] = []
34        n = data.shape[0]
35        best_acc = 0
36        tolerance = 0
37        for i in range(iterations):
38            y_pred = self.predict(data)
39            positive = label * (1 / (y_pred + epsilon))
40            negative = (1 - label) * (1 / (y_pred - 1 - epsilon))
41            grad = -(positive + negative)
42            grad = grad * y_pred * (1 - y_pred)
43            grad_w = np.matmul(data.T, grad) / n
44            grad_b = np.sum(grad) / n
45            self.w -= lr * grad_w
46            self.b -= lr * grad_b
47            positive = label * np.log(y_pred + epsilon)
48            negative = (1 - label) * np.log(1 - y_pred + epsilon)
49            loss = -np.sum(positive + negative) / n
50            y_pred = y_pred > 0.5
51            acc = np.sum(y_pred == label) / n
52            if acc > best_acc:
53                tolerance = 0
54                best_acc = acc
55            else:
56                tolerance += 1
57            if tolerance >= patience:
58                break
59            lr *= decay
```

图 5: 部分代码截图

实验二 KNN 分类任务

1 问题描述

通过 KNN 算法对 Iris 鸢尾花数据集中的测试集进行分类，分别使用欧式距离以及马氏距离作为度量实现，同时实现基于马氏距离的度量学习算法 NCA

2 实现步骤与流程

KNN 算法

KNN 算法选取训练集中与未见样本的度量最小的 k 个样本中类别众数作为预测值

NCA 算法

样本的散布矩阵

$$\mathbf{S} = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T$$

马氏距离

$$d_M(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{S}^{-1} (\mathbf{x}_i - \mathbf{x}_j)}$$

可学习的马氏距离（伪马氏距离）

$$d_M(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j)}$$

其中， \mathbf{M} 是半正定矩阵，重写以上距离数学形式

$$d_M(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{A}^T \mathbf{A} (\mathbf{x}_i - \mathbf{x}_j)}$$

其中变换矩阵 $\mathbf{A} \in \mathbb{R}^{k \times d}$ ，并且 $k \geq \text{rank}(\mathbf{M})$ 。设样本 \mathbf{x}_i 的近邻分布为

$$p_{ij} = \exp\{-d_M^2(\mathbf{x}_i, \mathbf{x}_j)\} / Z_i$$

$$Z_i = \sum_{k \neq i} \exp\{-d_M^2(\mathbf{x}_i, \mathbf{x}_k)\}$$

代表样本 \mathbf{x}_j 与样本 \mathbf{x}_i 属于相同类别的概率。针对变换矩阵 \mathbf{A} 最大化同类样本相似程度

$$\max_{\mathbf{A}} f(\mathbf{A}) = \sum_{i=1}^n \sum_{j \in \Omega_i} p_{ij}$$

其中 Ω_i 表示和样本 \mathbf{x}_i 属于同一类别的其它样本的集合。令

$$d_{ij} = d_M^2(\mathbf{x}_i, \mathbf{x}_j)$$

准则函数对变换矩阵 \mathbf{A} 求偏导数得

$$\frac{\partial p_{ij}}{\partial d_{ik}} = \begin{cases} -p_{ij}(1 - p_{ij}) & k = j \\ p_{ij}p_{ik} & k \neq j \end{cases}$$

$$\frac{\partial d_{ik}}{\partial \mathbf{A}} = 2\mathbf{A}(\mathbf{x}_i - \mathbf{x}_k)(\mathbf{x}_i - \mathbf{x}_k)^T$$

$$\frac{\partial p_{ij}}{\partial \mathbf{A}} = 2p_{ij}\mathbf{A} \left[\sum_{k \neq i} p_{ik}(\mathbf{x}_i - \mathbf{x}_k)(\mathbf{x}_i - \mathbf{x}_k)^T - (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T \right]$$

$$\frac{\partial f}{\partial \mathbf{A}} = \sum_{i=1}^n \sum_{j \in \Omega_i} \frac{\partial p_{ij}}{\partial \mathbf{A}}$$

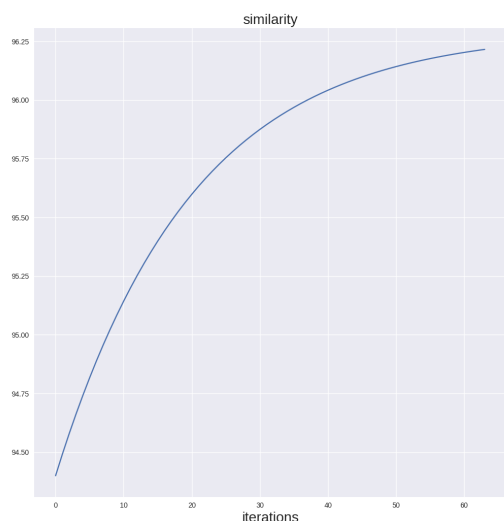
利用梯度下降法即可学习到一个较优的变换矩阵，使得变换后的同类样本距离，异类样本远离。但注意到该优化问题非凸，意味着最终收敛得到的解可能是局部最优解，并且与迭代初始值密切相关，实际实验中需要多次尝试才能得到满足要求的解。

3 实验结果与分析

由于实验要求的 MindSpore 框架相关的资料较为匮乏，因此本次实验采取开源机器学习算法库 sklearn 与实验中的算法进行对比。sklearn 是一个开源的基于 python 语言的机器学习工具包。它通过 numpy, scipy 等 python 数值计算的库实现高效的算法应用，并且涵盖了几乎所有主流机器学习算法。

NCA 算法

以下是 NCA 算法的度量学习历史以及最终收敛的模型的变换矩阵对样本进行降维后的分布图



(a) 度量学习历史



(b) 降维样本分布

图 1: NCA 算法

观察可得，随着模型的迭代，同类样本的相似度逐渐提高，这点也可以从降维样本上看出

KNN 算法

以下是实验 KNN 算法（欧式距离、马氏距离）与 sklearn 的 KNN 算法的在验证集上的准确率对比

```

● (python) wzh@wzh:~/workspace/course/pattern recognition/实验/实验2$
euclidean accuracy: 1.0
100%|████████████████████████████████████████████████████████████████████████████████
mahalanobis accuracy: 1.0
  
```

(a) KNN (实验)

```

● (python) wzh@wzh:~/workspace/course/pattern recognition/实验/实验2$
euclidean accuracy: 1.0
  
```

(b) KNN (sklearn)

图 2: KNN 算法准确率

经过实验，采用 5 近邻在验证集上可以达到最高的准确率 100%，但是由于样本数量较少，因此结果只能在一定程度上作为参考

4 MindSpore 学习使用心得体会

由于本次实验并未采用 MindSpore，此部分用 sklearn 的学习使用心得体会来代替。

本次实验中调用了 sklearn 中的以下算法接口

- KNN (neighbors KNeighborsClassifier)

作为实验算法的对照算法，sklearn 提供的算法接口方便调用，并且执行效率较高，能够高效地实现实验的需求。

5 代码附录

```
mine.py  knn.py  ...  scikit-learn.py  X
实验2 > src > knn.py > ...
1 import numpy as np
2 from tqdm import tqdm
3
4 class KNN(object):
5
6     def __init__(self, solver:str):
7         self.solver = solver
8
9     def fit(
10         self,
11         data,
12         label,
13         **kwargs
14     ):
15         self.data = data
16         self.label = label
17         self.c = np.max(label) + 1
18         if self.solver == 'euclidean':
19             return
20         index_split = [
21             np.where(label == i)[0]
22             for i in range(self.c)
23         ]
24         n = data.shape[0]
25         input_dim = data.shape[-1]
26         output_dim = kwargs['transform_dim']
27         self.A = 0.75 * np.random.randn(
28             input_dim,
29             output_dim
30         )
31         iterations = kwargs['iterations']
32         lr = kwargs['lr']
33         decay = kwargs['decay']
34         history = []
35         for iteration in tqdm(range(iterations)):
36             tmp = np.matmul(data, self.A)
37             tmp = tmp[:, np.newaxis, :] - \
38                 tmp[np.newaxis, :, :]
39             d = np.sum(tmp ** 2, axis=-1)
40             z = np.sum(np.exp(-d), axis=-1, keepdims=True)
41             z = z - 1
42             p = np.exp(-d) / (z + 0.001)
43             grad = np.zeros_like(self.A)
44             similarity = 0.
45             for i in range(n):
46                 index_i = index_split[label[i]]
47                 tmp = np.zeros((input_dim, input_dim))
48                 for k in range(n):
49                     if k == i: continue
50                     tmp += p[i][k] * np.outer(
51                         data[i] - data[k],
52                         data[i] - data[k]
53                     )
54                 for j in index_i:
55                     similarity += n[i][j]
```

```
实验2 > src > scikit-learn.py > ...
1 import numpy as np
2 import pandas as pd
3 from sklearn.neighbors import \
4     KNeighborsClassifier as KNN
5 from data import IrisData
6
7 args = dict()
8 args['dataroot'] = r'./data/'
9 args['predict'] = r'./prediction/'
10
11 class SciIrisData(IrisData):
12
13     def __init__(self, dataroot):
14         super().__init__(dataroot)
15
16     def euclidean(self, k):
17         model = KNN(k)
18         data = self.train[:, :-1]
19         label = self.train[:, -1].astype(np.int32)
20         model.fit(data, label)
21         data = self.val[:, :-1]
22         label = self.val[:, -1].astype(np.int32)
23         y_pred = model.predict(data)
24         acc = np.sum(y_pred == label)
25         acc /= data.shape[0]
26         return model, acc
27
28 if __name__ == '__main__':
29     import warnings
30     warnings.filterwarnings('ignore')
31
32     iris = SciIrisData(args['dataroot'])
33     k = 5
34
35     model, acc = iris.euclidean(k)
36     print('euclidean accuracy: {}'.format(acc))
37     y_pred = model.predict(iris.test)
38     pd.DataFrame(y_pred).to_csv(
39         args['predict'] + 'task3_test_prediction.csv',
40         index=False, header=['label']
41     )
42
```

图 3: 部分代码截图

实验三 神经网络

1 问题描述

利用神经网络算法对 MNIST 数据集进行分类

2 实现步骤与流程

BP 算法通过链式法则将误差对参数的梯度逐级传播，并通过梯度下降算法对参数进行优化

加载数据集

实验提供的 MNIST 数据集文件为.idx3-ubyte 和.idx1-ubyte 格式，需要对读取到的二进制序列进行处理，转化为 numpy 可以进行运算的 ndarray 的格式

定义基础模块

神经网络可能由多个模块构成，为了简化网络模型的构建过程，需要对神经网络中的模块概念进行封装，并定义模块的前向传播以及反向传播机制，本次实验实现了多种基本模块，包括全连接层（Linear）、标准化层（LayerNormalization）以及常见的激活函数 relu 层、sigmoid 层、tanh 层以及 softmax 层，并提供了简单的序列神经网络模块（Sequence）用于堆叠各种模块形成网络

定义损失函数

为了对网络中的参数进行优化，还需要定义损失函数，包括损失函数的计算以及梯度的回传。本次实验提供了交叉熵损失函数（CrossEntropy）用于多分类任务的优化

构建训练网络

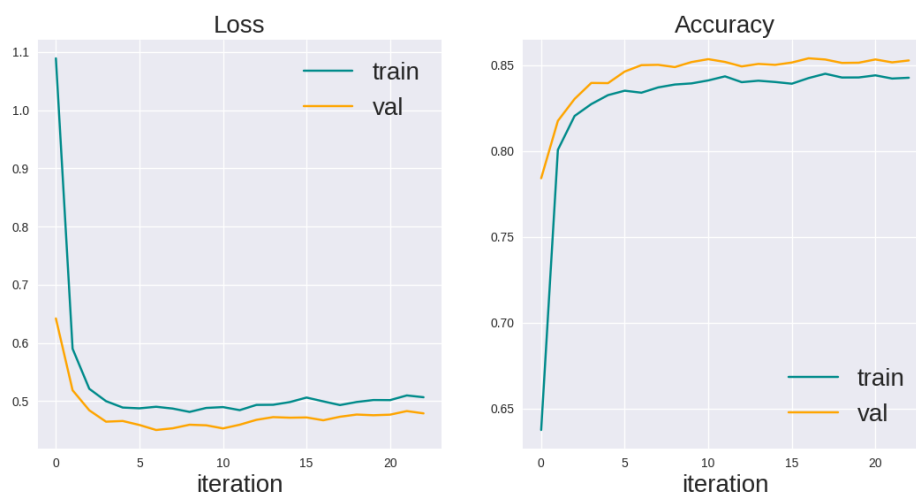
在初始化数据集以及构建好神经网络后，即可对网络中的参数进行学习训练。本次实验采用小批次梯度下降算法，为了加快网络的收敛速度的同时保证网络收敛的稳定性，实验中采用了学习率的指数衰减策略，并且

为了减少网络过拟合的可能，实验中实现了网络的提前体制（early-stop）。在训练过程中记录网络模型在训练集以及验证集上的平均损失函数以及准确率的历史数据，训练结束后可视化历史数据，观察网络的训练过程

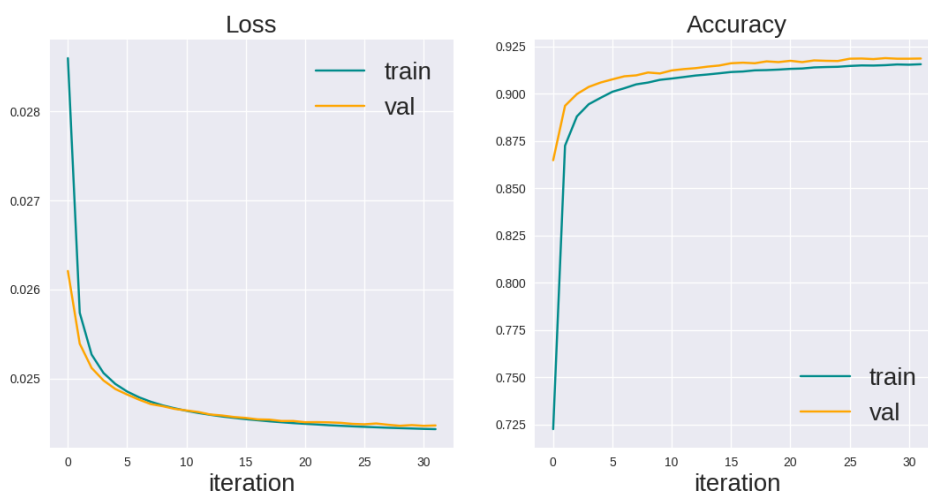
3 实验结果与分析

由于实验要求的 MindSpore 框架相关的资料较为匮乏，因此本次实验采取深度学习框架 pytorch 与实验中的算法进行对比。pytorch 是一个开源的 python 深度学习库，可以用于构建能够自动求导的神经网络。

以下是实验中构建的神经网络的训练历史和相同结构的网络在 pytorch 下的训练历史



(a) 历史数据（实验）



(b) 历史数据（pytorch）

图 1: 神经网络训练历史

可以看出，随着训练轮数的增加，网络模型在分类的准确率上不断上升最终达到收敛。与 pytorch 的算法

相比，实验的算法最终的收敛准确率较低，在 85% 左右，而 pytorch 最终的准确率可以达到接近 92%，并且在实验算法每个轮次需要约 1.6s 的运行时间，而 pytorch 每个轮次需要 1.2s 的运行时间，在时间效率上更高。

经过实际测试，本次实验采用的单层网络结构在训练效率以及准确率上的综合表现较好。

4 MindSpore 学习使用心得体会

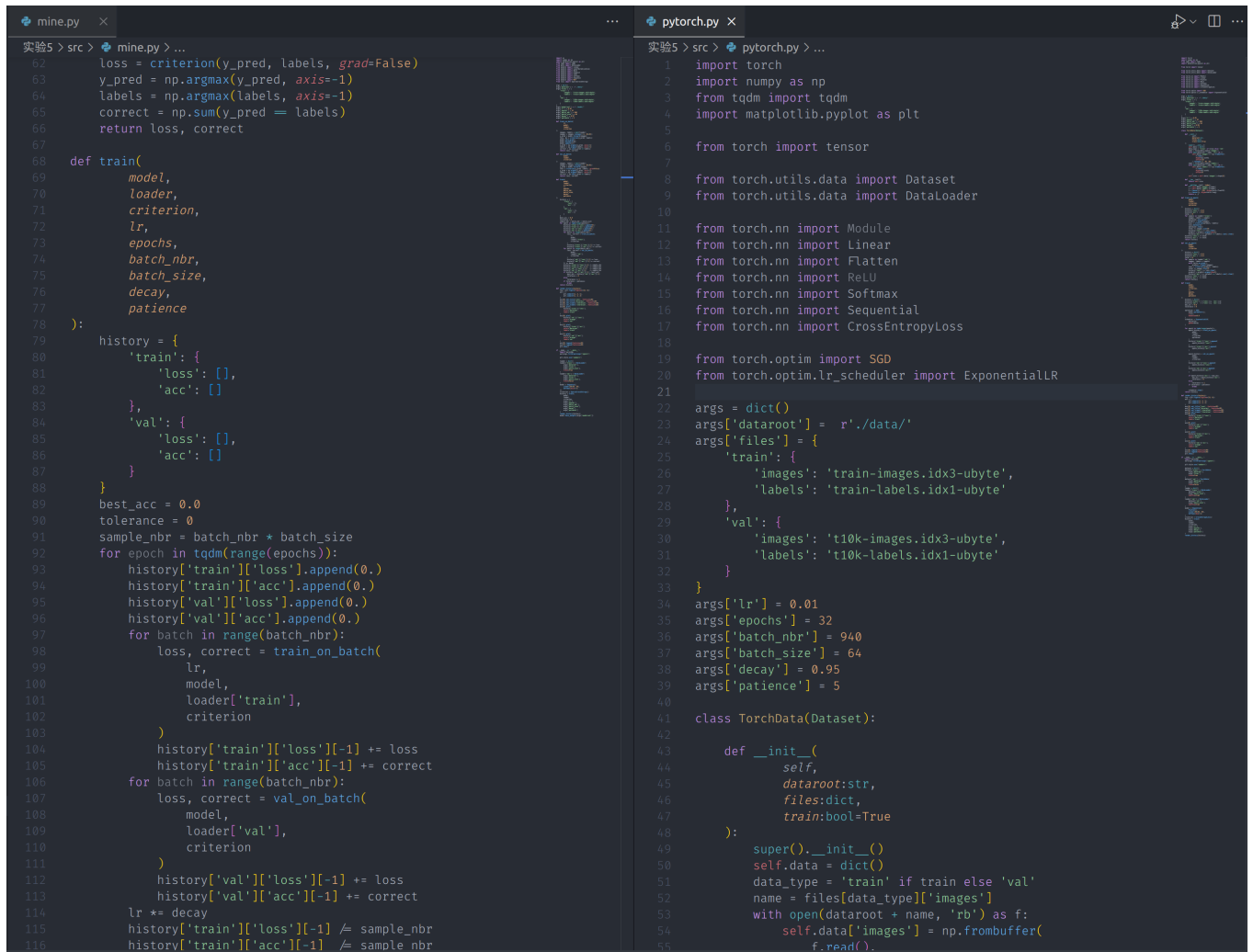
由于本次实验并未采用 MindSpore，此部分用 pytorch 的学习使用心得体会来代替。

本次实验中调用了 pytorch 中的以下算法接口

- 神经网络模块 (nn)
- 数据集接口 (utils data Dataset、DataLoader)
- SGD 优化器 (optim SGD)
- 学习率指数衰减 (optim lr_scheduler ExponentialLR)

作为实验算法的对照算法，pytorch 提供的算法接口方便调用，并且执行效率较高，能够高效地实现实验的需求。

5 代码附录



```
实验5 > src > mine.py > ...
62     loss = criterion(y_pred, labels, grad=False)
63     y_pred = np.argmax(y_pred, axis=-1)
64     labels = np.argmax(labels, axis=-1)
65     correct = np.sum(y_pred == labels)
66     return loss, correct
67
68 def train(
69     model,
70     loader,
71     criterion,
72     lr,
73     epochs,
74     batch_nbr,
75     batch_size,
76     decay,
77     patience
78 ):
79     history = {
80         'train': {
81             'loss': [],
82             'acc': []
83         },
84         'val': {
85             'loss': [],
86             'acc': []
87         }
88     }
89     best_acc = 0.0
90     tolerance = 0
91     sample_nbr = batch_nbr * batch_size
92     for epoch in tqdm(range(epochs)):
93         history['train']['loss'].append(0.)
94         history['train']['acc'].append(0.)
95         history['val']['loss'].append(0.)
96         history['val']['acc'].append(0.)
97         for batch in range(batch_nbr):
98             loss, correct = train_on_batch(
99                 lr,
100                 model,
101                 loader['train'],
102                 criterion
103             )
104             history['train']['loss'][-1] += loss
105             history['train']['acc'][-1] += correct
106         for batch in range(batch_nbr):
107             loss, correct = val_on_batch(
108                 model,
109                 loader['val'],
110                 criterion
111             )
112             history['val']['loss'][-1] += loss
113             history['val']['acc'][-1] += correct
114         lr *= decay
115         history['train']['loss'][-1] /= sample_nbr
116         history['train']['acc'][-1] /= sample_nbr
117
118     return history, best_acc, tolerance
```

```
实验5 > src > pytorch.py > ...
1  import torch
2  import numpy as np
3  from tqdm import tqdm
4  import matplotlib.pyplot as plt
5
6  from torch import tensor
7
8  from torch.utils.data import Dataset
9  from torch.utils.data import DataLoader
10
11  from torch.nn import Module
12  from torch.nn import Linear
13  from torch.nn import Flatten
14  from torch.nn import ReLU
15  from torch.nn import Softmax
16  from torch.nn import Sequential
17  from torch.nn import CrossEntropyLoss
18
19  from torch.optim import SGD
20  from torch.optim.lr_scheduler import ExponentialLR
21
22  args = dict()
23  args['dataroot'] = 'r'./data/'
24  args['files'] = {
25      'train': {
26          'images': 'train-images.idx3-ubyte',
27          'labels': 'train-labels.idx1-ubyte'
28      },
29      'val': {
30          'images': 't10k-images.idx3-ubyte',
31          'labels': 't10k-labels.idx1-ubyte'
32      }
33  }
34  args['lr'] = 0.01
35  args['epochs'] = 32
36  args['batch_nbr'] = 940
37  args['batch_size'] = 64
38  args['decay'] = 0.95
39  args['patience'] = 5
40
41  class TorchData(Dataset):
42
43      def __init__(
44          self,
45          dataroot:str,
46          files:dict,
47          train:bool=True
48      ):
49          super().__init__()
50          self.data = dict()
51          data_type = 'train' if train else 'val'
52          name = files[data_type]['images']
53          with open(dataroot + name, 'rb') as f:
54              self.data['images'] = np.frombuffer(
55                  f.read(),
```

图 2: 部分代码截图

心得体会

本次实验增进了我对模式识别课程的知识理解,同时让我更加熟悉基于 python 的机器学习库 scikit-learn 以及深度学习库 pytorch 的使用,并且提升了我的编程能力。

值得注意的是,本次实验的实验二——KNN 分类任务中关于 NCA 算法的阐释上存在严重的错误,原实验说明文档中提到的是最小化同类样本相似程度,但实际上应当是最大化,具体的推导过程见实验报告中实验二部分

实验内容

- 利用欧式距离作为 KNN 算法的度量函数,对测试集进行分类。实验报告中,要求在验证集上分析近邻数 K 对 KNN 算法分类精度的影响。
- 利用马氏距离作为 KNN 算法的度量函数,对测试集进行分类。在马氏距离中, M 为半正定矩阵,正交基 A 使得 $M = AA^T$ 成立。给定以下目标函数,在训练集上利用梯度下降法对马氏距离进行学习:

$$f(A) = \min_A \sum_{i=1}^N \sum_{j \in \Omega_i} p_{ij},$$

其中, Ω_i 表示与 x_i 属于相同类别的样本的下标的集合, p_{ij} 定义为:

$$p_{ij} = \begin{cases} \frac{\exp(-d_M(x_i, x_j)^2)}{\sum_{k \neq i} \exp(-d_M(x_i, x_k)^2)} & j \neq i, \\ 0 & j = i \end{cases},$$

d_M 为:

$$d_M(x_i, x_j) = \|Ax_i - Ax_j\|_2.$$

实验中,矩阵 A 的维度 e 可任意设置为一个合适值,例如 $e = 2$ 。实验报告中请对优化过程的梯度计算公式进行推导,即给出 $\frac{df}{dA}$ 的计算公式。

图 1: 实验文档中对于 NCA 算法的表述