可能会导致学习率过早下降。

- RMSprop（均方根反向传播）：改进的 AdaGrad 算法，引入了一个衰减系数来控制历史梯度的影响，避免学习率过早下降。
- Adam（自适应矩估计）：结合了动量优化器和 RMSprop 的优点，同时具备较快的收敛速度和较好的参数更新效果。Adam 是目前深度学习中最常用的优化器之一。

这些优化器都有各自的特点和适用场景。选择合适的优化器取决于数据集、模型架构和训练任务的性质。通常情况下，Adam 优化器是一个较好的默认选择，但在某些情况下，其他优化器也可能获得更好的结果。因此，选择合适的优化器需要在实践中进行实验和调整。

7.6 应用实例：客户信用评级——防范金融风险


应用实例

金融系统是国家经济的核心组成部分，金融对国家的重要性体现在促进经济发展、提高生活水平、创造就业机会、保障金融稳定、提供金融服务等多个方面。它是国家经济体系中不可或缺的组成部分，对于国家和人民的繁荣和福祉具有深远影响。

金融安全是确保经济稳定和可持续发展的基石，当今世界大的经济波动往往由金融问题引发。1997 年亚洲金融危机、2008 年国际金融危机提醒我们，对金融风险不能不未雨绸缪，对维护金融安全必须高度重视。

习近平总书记指出“金融安全是国家安全的重要组成部分，是经济平稳健康发展的重要基础，维护金融安全，是关系我国经济社会发展全局的一件带有战略性、根本性的大事”。党的二十大报告中也提出“加强和完善现代金融监管，强化金融稳定保障体系，依法将各类金融活动全部纳入监管，守住不发生系统性风险底线”。维护金融安全的重要性不言而喻。

其中，金融欺诈检测在防范金融风险中起着重要的作用。它能够及早发现潜在的欺诈行为，提供实时监测和预警功能，加强身份验证和访问控制，提高客户信任和保护品牌声誉，同时满足合规和监管要求。通过有效的欺诈检测，金融机构可以降低风险、减少损失，并确保金融体系的安全和稳定运行。

本案例利用全连接神经网络，通过对贷款申请客户的多项特征进行分析，实现风险检测。本案例使用的数据集存储在文件“application.csv”中，我们首先导入数据集，代码如下。

代码7-2　读取并显示数据集

```
import pandas as pd
```

```
#读取文件
data = pd.read_csv("application.csv")
data.head()
```

以上代码的输出结果如图 7-12 所示，其中，SK_ID_CURR 为样本编号，TARGET 为分类标签，1 表示该样本客户存在付款困难，有风险，0 表示正常，我们通过特征来进行分类。数据集共有 7 个特征，分别为：INCOME_TOTAL（客户收入），CREDIT（贷款授信金额），ANNUITY（贷款年金），GOODS_PRICE（申请消费贷款购买商品的价格），INCOME_TYPE（客户的收入类型，共 8 种），HOUR_APPR（客户申请贷款大概的时间），GRGANIZATION_TYPE（客户工作的组织类型，共 58 种）。

	SK_ID_CURR	TARGET	INCOME_TOTAL	CREDIT	ANNUITY	GOODS_PRICE	INCOME_TYPE	HOUR_APPR	ORGANIZATION_TYPE
0	100002	1	202500.0	406597.5	24700.5	351000.0	Working	10	Business Entity Type 3
1	100003	0	270000.0	1293502.5	35698.5	1129500.0	State servant	11	School
2	100004	0	67500.0	135000.0	6750.0	135000.0	Working	9	Government
3	100006	0	135000.0	312682.5	29686.5	297000.0	Working	17	Business Entity Type 3
4	100007	0	121500.0	513000.0	21865.5	513000.0	Working	11	Religion

图7-12 数据集信息

接下来，删除数据集中包含空值的样本，代码如下。

代码7-3　读取并显示数据集

```
# 统计空值
null_counts = data.isnull().sum()
print(null_counts)

# 删除包含空值的行
data = data.dropna()

# 打印删除空值后的数据集信息
print(data.info())
```

以上代码的输出结果如图 7-13 所示，原数据集的 ANNUITY 和 GOODS_TYPE 分别包含 12 条和 278 条空值，删除含空值的样本后，数据集中包含 307221 条样本。

```
SK_ID_CURR           0
TARGET               0
INCOME_TOTAL         0
CREDIT               0
ANNUITY              0
GOODS_PRICE          0
INCOME_TYPE          0
HOUR_APPR            0
ORGANIZATION_TYPE    0
dtype: int64
<class 'pandas.core.frame.DataFrame'>
Int64Index: 307221 entries, 0 to 307510
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   SK_ID_CURR         307221 non-null  int64
 1   TARGET             307221 non-null  int64
 2   INCOME_TOTAL       307221 non-null  float64
 3   CREDIT             307221 non-null  float64
 4   ANNUITY            307221 non-null  float64
 5   GOODS_PRICE        307221 non-null  float64
 6   INCOME_TYPE        307221 non-null  object
 7   HOUR_APPR          307221 non-null  int64
 8   ORGANIZATION_TYPE  307221 non-null  object
dtypes: float64(4), int64(3), object(2)
memory usage: 23.4+ MB
None
```

图7-13 删除空值后数据集信息

从图 7-12 和图 7-13 均可以看到，INCOME_TYPE 和 ORGANIZATION 两个特征的类型为文本型，称为标签编码，必须将它们转换为数值型才能作为输入，代码如下。

代码7-4　标签编码

```
from sklearn import preprocessing

# 创建 LabelEncoder 对象，用于对标签进行编码
label_encoder = preprocessing.LabelEncoder()

# 将 'INCOME_TYPE' 字段的标签进行编码
data['INCOME_TYPE'] = label_encoder.fit_transform(data['INCOME_TYPE'])

# 将 'ORGANIZATION_TYPE' 字段的标签进行编码
data['ORGANIZATION_TYPE'] =
label_encoder.fit_transform(data['ORGANIZATION_TYPE'])

data.head()
```

以上代码的输出结果如图 7-14 所示，使用 preprocessing.LabelEncoder 对象对数据集中的两个字段进行标签编码。

	SK_ID_CURR	TARGET	INCOME_TOTAL	CREDIT	ANNUITY	GOODS_PRICE	INCOME_TYPE	HOUR_APPR	ORGANIZATION_TYPE
0	100002	1	202500.0	406597.5	24700.5	351000.0	7	10	5
1	100003	0	270000.0	1293502.5	35698.5	1129500.0	4	11	39
2	100004	0	67500.0	135000.0	6750.0	135000.0	7	9	11
3	100006	0	135000.0	312682.5	29686.5	297000.0	7	17	5
4	100007	0	121500.0	513000.0	21865.5	513000.0	7	11	37

图7-14 标签编码后的数据集

数据预处理完毕后，将数据集划分为训练集和测试集，进行标准化并转换为张量，代码如下。

代码7-5 标签编码

```
import torch
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# 分割特征和标签
X = data.drop(columns=["TARGET", "SK_ID_CURR"]).values
y = data["TARGET"].values.reshape(-1, 1)
X.shape

# 数据标准化
scaler = StandardScaler()
X = scaler.fit_transform(X)

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# 转换为张量
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32)
```

接下来，我们构建全连接神经网络，代码如下。

代码7-6 构建网络

```
import torch.nn as nn

# 定义神经网络模型
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(7, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 2)  # 输出层为 2，对应两个类别

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# 初始化模型
model = Net()
```

以上代码构建的模型包含三层全连接层，输入个数为 7，即 7 个特征，输出为 2（对应类别数）。接下来，开始训练网络模型，代码如下。

代码7-7　训练模型

```
import torch.optim as optim

# 定义损失函数和优化器
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 训练模型
num_epochs = 15
train_losses = []
train_accs = []
test_accs = []
for epoch in range(num_epochs):
    # 前向传播
    outputs = model(X_train)
    loss = criterion(outputs, y_train.squeeze().long())

    # 反向传播和优化
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # 计算训练集准确率
    _, predicted = torch.max(outputs, 1)
    train_acc = (predicted == y_train.squeeze().long()).sum().item() /
y_train.shape[0]

    # 计算测试集准确率
    with torch.no_grad():
        test_outputs = model(X_test)
        _, predicted = torch.max(test_outputs, 1)
        test_acc = (predicted == y_test.squeeze().long()).sum().item() /
y_test.shape[0]

    # 保存损失和准确率
    train_losses.append(loss.item())
    train_accs.append(train_acc)
    test_accs.append(test_acc)

    # 输出训练信息
    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {loss.item():.4f}, Train
Acc: {train_acc:.4f}, Test Acc: {test_acc:.4f}")
```

以上代码的输出结果如图 7-15 所示。本案例为分类问题，因此使用交叉熵损失函数 nn.CrossEntropyLoss，设置 Adam 优化器，学习率为 0.001，迭代次数为 15。从输出结果可见，随着训练周期的增加，模型的性能逐渐提升，损失逐渐减少，训练集和测试集的准确率也随之增加。从第 8 次迭代开始，准确率不再提升，模型收敛。

```
Epoch 1/15, Loss: 0.7179, Train Acc: 0.2181, Test Acc: 0.4197
Epoch 2/15, Loss: 0.6994, Train Acc: 0.4189, Test Acc: 0.6608
Epoch 3/15, Loss: 0.6816, Train Acc: 0.6612, Test Acc: 0.7805
Epoch 4/15, Loss: 0.6644, Train Acc: 0.7804, Test Acc: 0.8438
Epoch 5/15, Loss: 0.6478, Train Acc: 0.8442, Test Acc: 0.8826
Epoch 6/15, Loss: 0.6318, Train Acc: 0.8849, Test Acc: 0.9046
Epoch 7/15, Loss: 0.6164, Train Acc: 0.9066, Test Acc: 0.9159
Epoch 8/15, Loss: 0.6014, Train Acc: 0.9179, Test Acc: 0.9177
Epoch 9/15, Loss: 0.5869, Train Acc: 0.9199, Test Acc: 0.9177
Epoch 10/15, Loss: 0.5727, Train Acc: 0.9199, Test Acc: 0.9177
Epoch 11/15, Loss: 0.5590, Train Acc: 0.9199, Test Acc: 0.9177
Epoch 12/15, Loss: 0.5455, Train Acc: 0.9199, Test Acc: 0.9177
Epoch 13/15, Loss: 0.5325, Train Acc: 0.9199, Test Acc: 0.9177
Epoch 14/15, Loss: 0.5197, Train Acc: 0.9199, Test Acc: 0.9177
Epoch 15/15, Loss: 0.5072, Train Acc: 0.9199, Test Acc: 0.9177
```

图7-15 模型训练结果

为了更加清晰地了解训练过程，我们对训练过程进行可视化，代码如下。

代码7-8　训练过程可视化

```
import matplotlib.pyplot as plt

# 绘制损失函数和准确率曲线
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(train_losses)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training Loss")

plt.subplot(1, 2, 2)
plt.plot(train_accs, label="Train")
plt.plot(test_accs, label="Test")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Training and Test Accuracy")
plt.legend()

plt.tight_layout()
plt.show()
```

以上代码的运行结果如图 7-16 所示，左图为损失的变化趋势，右图为模型准确率的变化趋势。

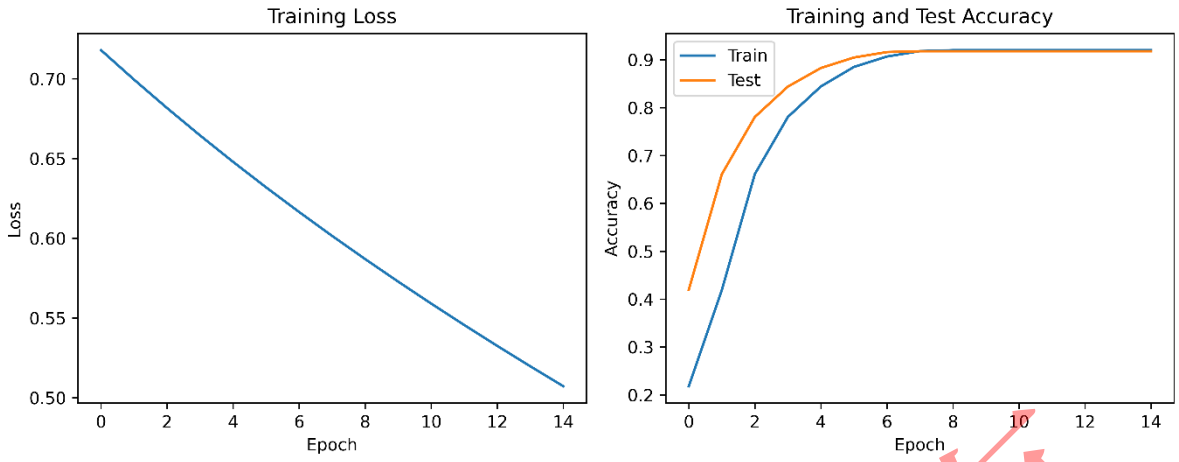

图7-16 训练过程可视化

读者可以调整网络的结构（如层数、每层神经元的个数等）、学习率、迭代次数等，观察训练的效果。这些调整，统称为超参数调整。深度学习中的超参数是指那些需要手动设置的参数，而不是通过训练过程自动学习的模型参数。这些超参数对于模型的性能和训练过程的效果起着重要的影响。

7.7 本章小结

本章首先介绍了深度学习与传统机器学习的区别，以及深度学习的历史和应用领域。接着，我们学习了神经网络的基础，包括生物神经元模型和前馈神经网络。

然后，我们深入讨论了神经网络的训练过程，包括前向传播、激活函数、损失函数、反向传播和优化算法。我们还介绍了常用的深度学习框架，如飞桨（PaddlePaddle）、TensorFlow 和 PyTorch。

最后，我们使用 PyTorch 构建了一个客户信用评级的应用实例，以防范金融风险。

本章的内容涵盖了深度学习的关键概念和实践技巧，帮助读者入门深度学习，并为进一步探索和应用提供了基础。通过本章的学习，读者将能够理解深度学习的基本原理和应用场景，为未来的学习打下坚实的基础。

7.8 课后习题

一、单项选择题

1. 深度学习是人工智能领域的哪个分支？（　）