

实验报告：人工智能算法平台搭建

一、实验目的

- 掌握常用人工智能算法的基本概念、基本技术和基本方法。
- 熟悉 Numpy 等常用 AI 库的使用。
- 尝试通过经典 AI 算法（如线性回归）解决实际问题。
- 理解机器学习模型的构建、训练和优化过程。

二、实验原理

线性回归是一种经典的监督学习算法，用于建立自变量与因变量之间的线性关系模型。本实验通过构建简单的一元线性回归模型 $y = ax + b$ ，使用梯度下降法优化参数 a （斜率）和 b （截距），使模型预测值与实际值之间的均方误差最小化。核心公式包括：

- 模型预测： $predict = a \times x + b$
- 损失函数（均方误差）： $loss = \frac{0.5}{n} \sum_{i=1}^n (predict_i - y_i)^2$
- 参数更新（梯度下降）：
 - $da = \frac{1}{n} \sum_{i=1}^n (predict_i - y_i) \times x_i$
 - $db = \frac{1}{n} \sum_{i=1}^n (predict_i - y_i)$
 - $a = a - Lr \times da$
 - $b = b - Lr \times db$ （其中 Lr 为学习率）

三、实验步骤与内容

1. 环境准备

安装并导入实验所需的 Python 库：

- Numpy**：用于数值计算和数组操作。
- Matplotlib**：用于数据可视化。

2. 数据准备与可视化

```
python
# 导入所需模块
```

```
import numpy as np
import matplotlib.pyplot as plt

# 定义数据, 将列表转换为数组
x = [3,21,22,34,54,34,55,67,89,99]
x = np.array(x)
y = [1,10,14,34,44,36,22,67,79,90]
y = np.array(y)

# 显示散点图
plt.scatter(x, y)
```

3. 模型构建

```
python
# 定义线性回归模型
def model(a, b, x):
    return a * x + b

# 定义损失函数 (均方误差)
def loss_function(a, b, x, y):
    num = len(x)
    predict = model(a, b, x)
    return (0.5 / num) * (np.square(predict - y)).sum()

# 定义优化函数 (梯度下降)
def optimize(a, b, x, y):
    num = len(x)
    predict = model(a, b, x)
    da = (1.0 / num) * ((predict - y) * x).sum()
    db = (1.0 / num) * ((predict - y).sum())
    a = a - Lr * da
    b = b - Lr * db
    return a, b

# 定义迭代训练函数
def iterate(a, b, x, y, times):
    for i in range(times):
        a, b = optimize(a, b, x, y)
    return a, b
```

4. 模型训练与结果可视化

```
python
# 初始化参数
a = np.random.rand(1) # 随机初始化斜率
b = np.random.rand(1) # 随机初始化截距
Lr = 1e-4 # 学习率

# 多次迭代训练并可视化结果
# 1 次迭代
a, b = iterate(a, b, x, y, 1)
prediction = model(a, b, x)
loss = loss_function(a, b, x, y)
print(f"1 次迭代: a={a}, b={b}, loss={loss}")
plt.scatter(x, y)
plt.plot(x, prediction)

# 2 次迭代 (累计 3 次)
a, b = iterate(a, b, x, y, 2)
prediction = model(a, b, x)
loss = loss_function(a, b, x, y)
print(f"3 次迭代: a={a}, b={b}, loss={loss}")
plt.scatter(x, y)
plt.plot(x, prediction)

# 3 次迭代 (累计 6 次)
a, b = iterate(a, b, x, y, 3)
prediction = model(a, b, x)
loss = loss_function(a, b, x, y)
print(f"6 次迭代: a={a}, b={b}, loss={loss}")
plt.scatter(x, y)
plt.plot(x, prediction)

# 4 次迭代 (累计 10 次)
a, b = iterate(a, b, x, y, 4)
prediction = model(a, b, x)
loss = loss_function(a, b, x, y)
print(f"10 次迭代: a={a}, b={b}, loss={loss}")
plt.scatter(x, y)
plt.plot(x, prediction)

# 1000 次迭代 (累计 1010 次)
a, b = iterate(a, b, x, y, 1000)
```

```
prediction = model(a, b, x)
loss = loss_function(a, b, x, y)
print(f"1010 次迭代: a={a}, b={b}, loss={loss}")
plt.scatter(x, y)
plt.plot(x, prediction)

plt.show()
```

四、实验结果与分析

1. **参数变化趋势：** 随着迭代次数的增加，模型参数 a 和 b 逐渐收敛到最优值，损失函数值不断减小并趋于稳定。
2. **模型拟合效果：** 从可视化结果可以观察到，随着训练迭代次数的增加，拟合直线越来越接近散点分布的趋势，说明模型对数据的拟合效果不断提升。
3. **算法有效性：** 通过梯度下降算法，模型能够自动调整参数以最小化损失函数，验证了线性回归算法在处理线性相关数据时的有效性。

五、实验总结与思考

1. 本次实验成功搭建了基于 Numpy 和 Matplotlib 的人工智能算法平台，实现了简单的线性回归模型。
2. 通过手动实现线性回归的损失函数和梯度下降优化过程，加深了对机器学习算法原理的理解。
3. 学习率和迭代次数是影响模型训练效果的重要超参数，需要根据实际情况进行调整以获得更好的结果。
4. 后续可以尝试使用 Scikit-Learn 库中的 LinearRegression 模块实现相同功能，并对比手动实现与库函数实现的差异。
5. 对于非线性数据，可以考虑引入多项式特征或使用更复杂的模型（如神经网络）进行拟合。

六、问题与展望

1. 本次实验使用的学习率是固定的，未来可以尝试自适应学习率策略以加快模型收敛速度。
2. 实验数据量较小，后续可以使用更大规模的数据集进行实验，验证模型的泛化能力。
3. 可进一步学习 PyTorch 等深度学习框架，实现更复杂的神经网络模型，解决更具挑战性的 AI 任务。