读书报告——线性神经网络

——计算机2101

杜建宇

1.引言

随着计算技术的飞速发展和大数据的普及应用，神经网络已经从最初的概念迅速演化为深度学习的核心组成部分。在过去的十年里，深度学习模型，如卷积神经网络和循环神经网络，通过在图像识别、自然语言处理等领域的卓越表现，引领了人工智能的发展潮流。然而，在这个丰富多彩的神经网络家族中，线性神经网络以其简单而直观的特性依然占据着特殊的地位。作为神经网络的基础模型，线性神经网络具有清晰的数学表达和易于理解的训练过程，为理解深度学习提供了坚实的基础。在本报告中，我们将深入研究线性神经网络的基本原理、算法细节以及在实际应用中的价值，同时通过Python代码演示，使读者能够亲身体验线性神经网络的工作机制及其在数据建模中的实际效果。

2.线性神经网络基础

线性神经网络结构和原理：

线性神经网络是一种基本的前馈神经网络结构，由输入层、输出层和可能的隐藏层组成。每个神经元与上一层的所有神经元连接，每个连接都有一个权重，同时每个神经元都有一个偏置。通过对输入进行加权求和，并应用激活函数，线性神经网络产生输出。用矩阵表示，如果我们将输入表示为向量 X，权重表示为矩阵 W，偏置表示为向量 b，输出表示为向量 Y，则前向传播可以表示为 $Y = WX + b$

线性激活函数的特性：

线性激活函数 $f(x) = x$ 在神经网络中表现为简单的比例放大或缩小输入。尽管它具有数学上的简单性，但线性激活函数的局限性在于它无法引入非线性关系。多层线性激活函数的组合等于单层线性激活函数，这限制了神经网络的表达能力。为了解决这一问题，非线性激活函数，如 $ReLU$（$RectifiedLinearUnit$）等，被引入以赋予神经网络更强大的表示能力。

训练数据：

训练数据是线性神经网络成功学习的基石。在处理图像数据时，我们通常对像素值进行标准化，将其缩放到[0, 1]的范围内。对于文本数据，可能会进行词嵌入或者TF-IDF处理。此外，数据的划分为**训练集、确认集和测试集**，这是我们课堂上讲过的内容，训练集是用于训练模型的数据集，占了总数据的绝大部分比例；确认集是用于调优模型参数、选择合适的模型或进行超参数调整的数据集，在训练过程中，模型在确认集上进行验证，根据确认集的性能调整模型的参数，确认集的主要作用是防止模型过度拟合训练集，即模型在训练集上表现很好但在从未见过的数据上表现不佳的情况；测试集是用于评估模型最终性能的数据集。一旦模型通过训练和验证，就会在测试集上进行测试，以评估其对未见过的数据的泛化能力，测试集是模型性能的最终评估标准，可以提供对模型在真实场景中的表现有多好的估计。

损失函数和优化算法：

损失函数是模型预测输出与实际标签之间差异的度量。常用的损失函数之一是均方误差（Mean Squared Error），定义为预测值与实际值之间差异的平方的平均值。梯度下降是一种基于梯度的优化算法，通过计算损失函数关于权重和偏置的梯度，沿着梯度的反方向更新参数，逐步减小损失。学习率是一个控制步进大小的超参数，过大可能导致震荡，过小可能导致收敛速度过慢。

3.算法细节

线性模型

线性假设是指目标可以表示为特征的加权和。为每一个特征添加一个权重，权重决定了每个特征对我们预测值的影响。还会添加一个偏置，偏置是指当所有特征都取值为0时，预测值应该为多少。虽然有时候考虑实际情况，不应该有偏置（或者说偏置为0），但是我们仍然需要偏置项。如果没有偏置，我们模型的表达能力将受到限制。

损失函数

在我们开始考虑如何用模型*拟合*（fit）数据之前，我们需要确定一个拟合程度的度量。 *损失函数*（loss function）能够量化目标的*实际*值与*预测*值之间的差距。 通常我们会选择非负数作为损失，且数值越小表示损失越小，完美预测时的损失为0。 回归问题中最常用的损失函数是平方误差函数。

解析解

线性回归刚好是一个很简单的优化问题。 与我们将在本书中所讲到的其他大部分模型不同，线性回归的解可以用一个公式简单地表达出来， 这类解叫作解析解（analytical solution）。

像线性回归这样的简单问题存在解析解，但并不是所有的问题都存在解析解。 解析解可以进行很好的数学分析，但解析解对问题的限制很严格，导致它无法广泛应用在深度学习里。

随机梯度下降

即使在我们无法得到解析解的情况下，我们仍然可以有效地训练模型。 在许多任务上，那些难以优化的模型效果要更好。 因此，弄清楚如何训练这些难以优化的模型是非常重要的。

本书中我们用到一种名为*梯度下降*（gradient descent）的方法， 这种方法几乎可以优化所有深度学习模型。 它通过不断地在损失函数递减的方向上更新参数来降低误差。

梯度下降最简单的用法是计算损失函数（数据集中所有样本的损失均值） 关于模型参数的导数（在这里也可以称为梯度）。 但实际中的执行可能会非常慢：因为在每一次更新参数之前，我们必须遍历整个数据集。 因此，我们通常会在每次需要计算更新的时候随机抽取一小批样本， 这种变体叫做*小批量随机梯度下降*（minibatch stochastic gradient descent）。

总结一下，算法的步骤如下： （1）初始化模型参数的值，如随机初始化； （2）从数据集中随机抽取小批量样本且在负梯度的方向上更新参数，并不断迭代这一步骤。

可以调整但不在训练过程中更新的参数称为*超参数*（hyperparameter）。 *调参*（hyperparameter tuning）是选择超参数的过程。 超参数通常是我们根据训练迭代结果来调整的， 而训练迭代结果是在独立的*验证数据集*（validation dataset）上评估得到的。

线性回归恰好是一个在整个域中只有一个最小值的学习问题。 但是对像深度神经网络这样复杂的模型来说，损失平面上通常包含多个最小值。 深度学习实践者很少会去花费大力气寻找这样一组参数，使得在*训练集*上的损失达到最小。 事实上，更难做到的是找到一组参数，这组参数能够在我们从未见过的数据上实现较低的损失， 这一挑战被称为*泛化*（generalization）。

4.线性神经网络的应用

分类应用：

线性神经网络在二元分类问题中表现出色，例如在医学图像中检测肿瘤是否恶性。通过学习权重和偏置，线性神经网络能够从图像中提取特征，使得对于恶性和良性肿瘤的分类更为准确。

回归应用：

在金融领域，线性神经网络被广泛用于预测股票价格。通过学习历史股价数据，线性神经网络可以捕捉到股价与各种经济指标之间的线性关系，从而提供未来股价的预测。

模式识别应用：

在自动驾驶技术中，线性神经网络可以用于识别道路上的标志和交通标识。通过学习不同标志的特征，线性神经网络能够准确地进行模式识别，从而实现车辆的智能导航。

实际案例和研究成果：

研究表明，线性神经网络在医学图像中的病灶检测方面表现出色，例如乳腺癌的早期诊断。其能够通过学习图像中的特征，辅助医生进行更准确的诊断，提高诊断效率。

5.Python代码实现

从零开始实现整个方法，包括数据流水线、模型、损失函数和小批量随机梯度下降优化器。

首先导入需要的库

```
%matplotlib inline
import random
import torch
from d2l import torch as d2l
```

生成数据集

为了简单起见，我们将根据带有噪声的线性模型构造一个人造数据集。 我们的任务是使用这个有限样本的数据集来恢复这个模型的参数。 我们将使用低维数据，这样可以很容易地将其可视化。 在下面的代码中，我们生成一个包含1000个样本的数据集， 每个样本包含从标准正态分布中采样的2个特征。 我们的合成数据集是一个矩阵$X\in R^{1000\times 2}$。

我们使用线性模型参数 w=[2,-3.4]^T、b=4.2和噪声项 ϵ ,生成数据集及其标签：$y = Xw + b + \epsilon$

ϵ 可以视为模型预测和标签时的潜在观测误差。 在这里我们认为标准假设成立，即 ϵ 服从均值为0的正态分布。 为了简化问题，我们将标准差设为0.01。 下面的代码生成合成数据集。

```
def synthetic_data(w, b, num_examples):  #@save
    """生成y=Xw+b+噪声"""
    X = torch.normal(0, 1, (num_examples, len(w)))
    y = torch.matmul(X, w) + b
    y += torch.normal(0, 0.01, y.shape)
    return X, y.reshape((-1, 1))

true_w = torch.tensor([2, -3.4])
true_b = 4.2
features, labels = synthetic_data(true_w, true_b, 1000)
```

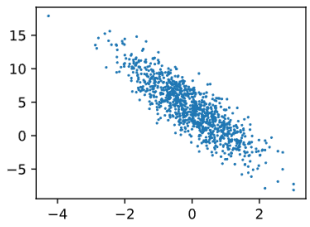
[`features`**中的每一行都包含一个二维数据样本，** `labels`**中的每一行都包含一维标签值（一个标量）**]。

```
[5]: print('features:', features[0],'\nlabel:', labels[0])

features: tensor([0.0538, 0.9438])
label: tensor([1.0949])
```

通过生成第二个特征`features[:, 1]`和`labels`的散点图， 可以直观观察到两者之间的线性关系。

```
[7]: d2l.set_figsize()
     d2l.plt.scatter(features[:, (1)].detach().numpy(), labels.detach().numpy(), 1);
```



读取数据集

训练模型时要对数据集进行遍历，每次抽取一小批量样本，并使用它们来更新我们的模型。 由于这个过程是训练机器学习算法的基础，所以有必要定义一个函数， 该函数能打乱数据集中的样本并以小批量方式获取数据。

在下面的代码中，我们[**定义一个`data_iter`函数， 该函数接收批量大小、特征矩阵和标签向量作为输入，生成大小为`batch_size`的小批量**]。 每个小批量包含一组特征和标签。

```
def data_iter(batch_size, features, labels):
    num_examples = len(features)
    indices = list(range(num_examples))
    # 这些样本是随机读取的，没有特定的顺序
    random.shuffle(indices)
    for i in range(0, num_examples, batch_size):
        batch_indices = torch.tensor(
            indices[i: min(i + batch_size, num_examples)])
        yield features[batch_indices], labels[batch_indices]
```

初始化模型参数

[**在我们开始用小批量随机梯度下降优化我们的模型参数之前**]， (**我们需要先有一些参数**)。 在下面的代码中，我们通过从均值为0、标准差为0.01的正态分布中采样随机数来初始化权重， 并将偏置初始化为0。

在初始化参数之后，我们的任务是更新这些参数，直到这些参数足够拟合我们的数据。 每次更新都需要计算损失函数关于模型参数的梯度。 有了这个梯度，我们就可以向减小损失的方向更新每个参数。 因为手动计算梯度很枯燥而且容易出错，所以没有人会手动计算梯度。 我们使用 :numref:`sec_autograd`中引入的自动微分来计算梯度。

定义模型

接下来，我们必须[**定义模型，将模型的输入和参数同模型的输出关联起来。**] 回想一下，要计算线性模型的输出， 我们只需计算输入特征$\mathbf{X}$和模型权重$\mathbf{w}$的矩阵-向量乘法后加上偏置b。 注意，上面的$\mathbf{X}\mathbf{w}$是一个向量，而b是一个标量。 我们利用:numref:`subsec_broadcasting`中描述的广播机制： 当我们用一个向量加一个标量时，标量会被加到向量的每个分量上。

```
def linreg(X, w, b):  #@save
    """线性回归模型"""
    return torch.matmul(X, w) + b
```

定义损失函数

```python
def squared_loss(y_hat, y):  #@save
    """均方损失"""
    return (y_hat - y.reshape(y_hat.shape)) ** 2 / 2
```

定义优化算法

在每一步中，使用从数据集中随机抽取的一个小批量，然后根据参数计算损失的梯度。 接下来，朝着减少损失的方向更新我们的参数。 下面的函数实现小批量随机梯度下降更新。 该函数接受模型参数集合、学习速率和批量大小作为输入。每 一步更新的大小由学习速率`lr`决定。 因为我们计算的损失是一个批量样本的总和，所以我们用批量大小（`batch_size`） 来规范化步长，这样步长大小就不会取决于我们对批量大小的选择。

```python
def sgd(params, lr, batch_size):  #@save
    """小批量随机梯度下降"""
    with torch.no_grad():
        for param in params:
            param -= lr * param.grad / batch_size
            param.grad.zero_()
```

训练

现在我们已经准备好了模型训练所有需要的要素，可以实现主要的训练过程部分了。 理解这段代码至关重要，因为从事深度学习后， 相同的训练过程几乎一遍又一遍地出现。 在每次迭代中，我们读取一小批量训练样本，并通过我们的模型来获得一组预测。 计算完损失后，我们开始反向传播，存储每个参数的梯度。 最后，我们调用优化算法`sgd`来更新模型参数。

在每个*迭代周期*（epoch）中，我们使用`data_iter`函数遍历整个数据集， 并将训练数据集中所有样本都使用一次（假设样本数能够被批量大小整除）。 这里的迭代周期个数`num_epochs`和学习率`lr`都是超参数，分别设为3和0.03。

```python
lr = 0.03
num_epochs = 3
net = linreg
loss = squared_loss

for epoch in range(num_epochs):
    for X, y in data_iter(batch_size, features, labels):
        l = loss(net(X, w, b), y)  # X和y的小批量损失
        # 因为l形状是(batch_size,1)，而不是一个标量。l中的所有元素被加到一起，
        # 并以此计算关于[w,b]的梯度
        l.sum().backward()
        sgd([w, b], lr, batch_size)  # 使用参数的梯度更新参数
    with torch.no_grad():
        train_l = loss(net(features, w, b), labels)
        print(f'epoch {epoch + 1}, loss {float(train_l.mean()):f}')
```

最后的结果为

```
epoch 1, loss 0.051143
epoch 2, loss 0.000229
epoch 3, loss 0.000049
```

最后我们来看一下训练的结果

因为我们使用的是自己合成的数据集，所以我们知道真正的参数是什么。 因此，我们可以通过[**比较真实参数和通过训练学到的参数来评估训练的成功程度**]。 事实上，真实参数和通过训练学到的参数确实非常接近。

```
[14]: print(f'w的估计误差: {true_w - w.reshape(true_w.shape)}')
      print(f'b的估计误差: {true_b - b}')

      w的估计误差: tensor([ 0.0009, -0.0004], grad_fn=<SubBackward0>)
      b的估计误差: tensor([0.0013], grad_fn=<RsubBackward1>)

[16]: print(f'得到的w的值为：{true_w}')
      print(f'得到的b的值为：{true_b}')

      得到的w的值为：tensor([ 2.0000, -3.4000])
      得到的b的值为：4.2
```

注意，我们不应该想当然地认为我们能够完美地求解参数。 在机器学习中，我们通常不太关心恢复真正的参数，而更关心如何高度准确预测参数。 幸运的是，即使是在复杂的优化问题上，随机梯度下降通常也能找到非常好的解。 其中一个原因是，在深度网络中存在许多参数组合能够实现高度精确的预测。

实验报告——主观贝叶斯方法

在主观Bayes方法中，知识和证据均具有不确定性。主观Bayes方法推理的任务就是根据证据E的概率P(E)及LS、LN的值，把H的先验概率P(H)更新为后验概率P(H|E)或者P(H|~E)。但由于证据的不确定性，要采取不同的更新方法。

一、实验目的

在证据不确定的情况下，以充分性量度LS、必要性量度LN、E的先验性概率P(E)和H的先验概率P(H)作为前提条件，分析P(H|S)和P(E|S)的关系。

二、实验原理

在现实中，证据肯定存在和肯定不存在的极端情况是不多的，更多的是介于二者之间的不确定情况，即0<P(E|S)<1。此时，我们用杜达等人在1976年证明了的公式来计算H的后验概率：

$$P(H|S) = P(H|E) \times P(E|S) + P(H|\neg E) \times P(\neg E|S)$$

分四种情况讨论这个公式：

1.P(E|S)=1（证据肯定存在）

当P(E|S)=1时，P(~E|S)=0。此时公式变成：

$$P(H|S) = P(H|E)$$

2.P(E|S)=0（证据肯定不存在）

当P(E|S)=0时,P(~E|S)=1。此时公式变成：

$$P(H|S) = P(H|\neg E)$$

3.P(E|S)=P(E)（观察与证据、结论无关）

当P(E|S)=P(E)时，表示E与S无关。利用全概率公式得

$$P(H|S) = P(H|E) \times P(E) + P(H|\neg E) \times P(\neg E) = P(H)$$

4.当P(E|S)为其它值时，通过分段线性插值就可以计算P(H|S)公式为：

$$P(H|S) = \begin{cases} P(H|\neg E) + \dfrac{P(H) - P(H|\neg E)}{P(E)} \times P(E|S) & 0 \le P(E|S) < P(E) \\ P(H) + \dfrac{P(H|E) - P(H)}{1 - P(E)} \times [P(E|S) - P(E)]) & P(E) \le P(E|S) \le 1 \end{cases}$$

该公式称为EH公式或UED公式

三、实验内容

编写程序，以LS、LN、P(E)、P(H)为已知输入，利用公式作出P(H|S)随P(E|S)变化关系图。

四、实验程序

```python
import numpy as np
from matplotlib import pyplot as plt

pH = float(input('pH= '))
pE = float(input('pE= '))
LN = float(input('LN= '))
LS = float(input('LS= '))

y2 = LS * pH / ((LS - 1) * pH + 1)
y1 = pH
y0 = LN * pH / ((LN - 1) * pH + 1)


def func(x):
    y = 0
    if x == 1:
        y = LS * pH / ((LS - 1) * pH + 1)
    elif x == 0:
        y = LN * pH / ((LN - 1) * pH + 1)
    elif x == pE:
        y = pH
    elif 0 < x < pE:
        y = y0 + x * (y1 - y0) / pE
    else:
        y = y1 + (x - pE) * (y2 - y1) / (1 - pE)
    return y


vfunc = np.vectorize(func)

x = np.linspace(0, 1, 200)
res = vfunc(x)

plt.plot(x, res, linestyle='-', color='g', marker='')
```

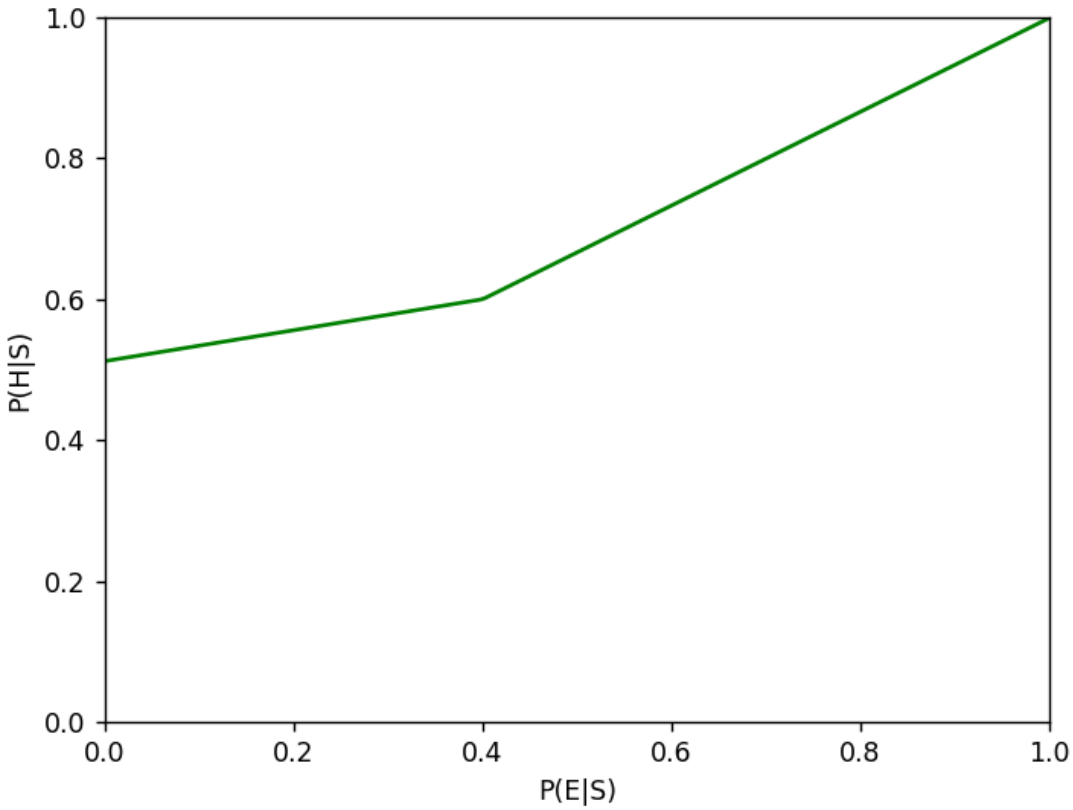
```
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel('P(E|S)')
plt.ylabel('P(H|S)')
plt.savefig("test.png")
plt.show()

```

五、实验结果

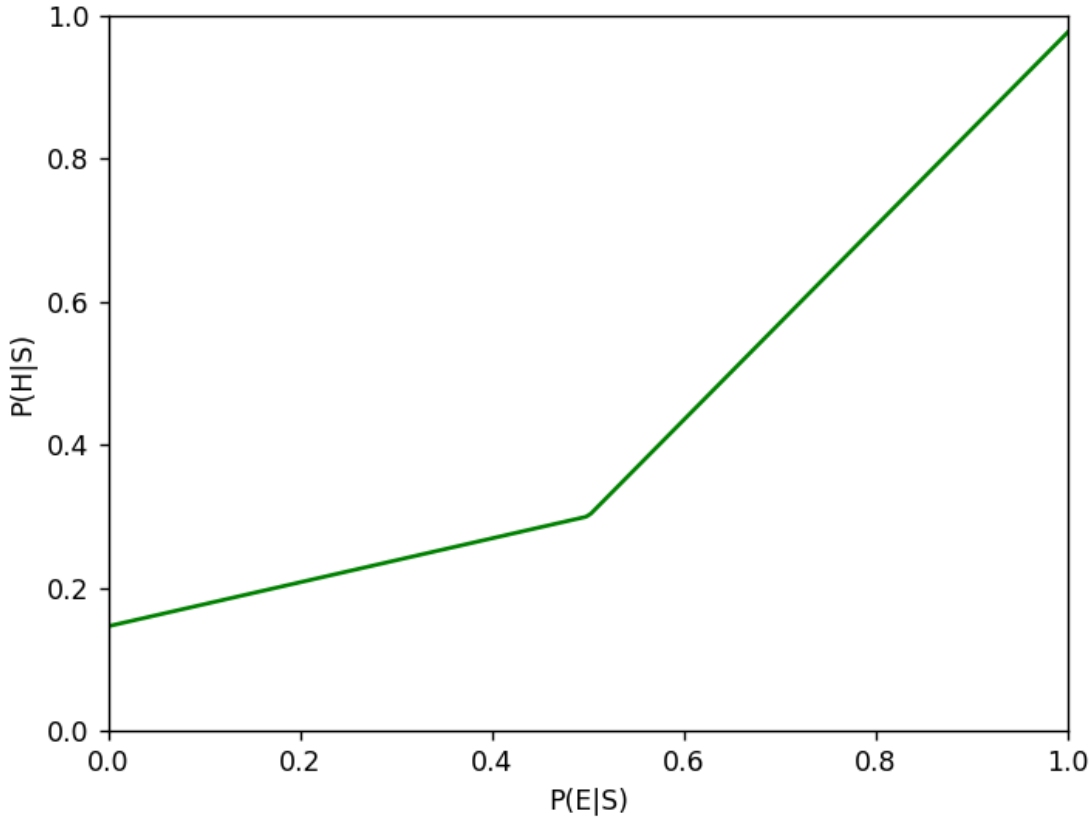
输入：LS=800，LN=0.7，PH=0.6，PE=0.4

结果显示：



输入：LS=100，LN=0.4，PH=0.3，PE=0.5

结果显示：



结果分析：

1. 当P(E|S)=0时，P(H|S)=P(H|~E)；
2. 当P(E|S)=P(E)时，P(H|S)=P(H)；
3. 当P(E|S)=1时，P(H|S)=P(H|E)；
4. 当P(E|S)为其它值时，P(H|S)即为分段线性插值计算结果

以上计算结果与实验原理所述完全一致

六、实验总结

通过实验，进一步掌握了主观Bayes方法中不确定性的传递算法及其公式，同时，通过多次输入参数并作图，经观察更深刻的理解了LS、LN的意义和性质，得知了主观Bayes方法的实质。

实验报告——重排九宫问题

2.1实验内容

在一个3*3的方格棋盘上放置8个标有1、2、3、4、5、6、7、8数字的棋牌，留下一个空格（用0表示）。规定：与空格相邻的棋牌可以移入空格。要求：利用非启发式算法和A*算法，寻找一条从初始状态到目标状态的棋牌移动路线。

2.2实验原理

2.2.1搜索

搜索的一般描述如算法1所述

> **算法 1** 搜索算法
>
> **输入：** 初始节点S0，目标节点条件

> **输出：** 搜索结果
>
> 1. 把初始节点S0放入OPEN表，并建立目前只包含S0的图，记为 G;
> 2. 检查 OPEN 表是否为空，若为空则问题无解，退出;
> 3. 把 OPEN 表的第一个节点取出放入 CLOSE 表，并计该节点为 n;
> 4. 考察节点 n 是否为目标节点。若是，则求得了问题的解，退出;
> 5. 扩展节点 n, 生成一组子节点。把其中不是节点 n 先辈的那些子节点记做集合 M，并把这些子节点作为节点 n 的子节点加入 G 中;
> 6. 针对 M 中子节点的不同情况，分别进行如下处理:
> 7. 对于那些未曾在 G 中出现过的 M 成员设置一个指向父节点（即节点 n的指针，并把它们放入 OPEN 表;(不在 OPEN 表)
> 8. 对于那些先前已经在 G 中出现过的 M 成员，确定是否需要修改它指向父节点的指针;(在 OPEN 表中)
> 9. 对于那些先前已在 G 中出现并且已经扩展了的 M 成员，确定是否需要修改其后继节点指向父节点的指针;(在 CLOSE 表中)
> 10. 按某种搜索策略对 OPEN 表中的节点进行排序;
> 11. 转第 2 步。

2.2.2广度搜索

广度优先搜索的基本思想: 从初始节点S0开始，逐层地对节点进行扩展，并考察它是否为目标节点。在第n层的节点没有全部扩展并考察之前，不对第n + 1层的节点进行扩展。 OPEN表中节点总是按进入的先后顺序排列，先进入的节点排在前面，后进入的排在后面。

算法如算法 2 所述。

> **算法 2** 广度搜索
>
> **输入：** 初始节点S0，目标节点条件
>
> **输出：** 搜索结果
>
> 1. 把初始节点 S0 放入 OPEN 表。
> 2. 检查 OPEN 表是否为空，若为空则问题无解，退出;
> 3. 把 OPEN 表的第一个节点取出放入 CLOSE 表，并计该节点为 n;
> 4. 把 OPEN 表的第一个节点取出放入 CLOSE 表，并计该节点为 n;
> 5. 若节点 n 不可扩展，则转第 2 步。
> 6. 扩展节点 n，将其子节点放入 OPEN 表的尾部，并为每一个子节点都配置指向父节点的指针，然后转第 2 步。

2.2.3 A*算法

如果一般搜索过程满足如下限制，则它就称为 A* 算法 ：

(1) 把 OPEN 表中的节点按估价函数:f(x) = g(x) + h(x) 的值从小至大进行排序（一般搜索过程的第 7 步)。

(2) g(x) 是从初始节点 SO 到节点 x 的路径的代价， g(x) 是对 g*(x) 的估计， g(x)>0。 (3) h(x) 是 h*(x) 的下界，即对所有的 x 均有: h(x)h*(x)。其中， g*(x) 是从初始节点 S0 到节点 x 的最小代价;h*(x) 是从节点 x 到目标节点的最小代价。

算法如算法 3 所述。

> **算法 3** A*算法
>
> **输入：** 初始节点 S0，目标节点条件，估值函数 h(x)

> **输出：** 搜索结果
>
> 1. 把初始节点 S0 放入 OPEN 表，并建立目前只包含 S0 的图，记为 G;
> 2. 检查 OPEN 表是否为空，若为空则问题无解，退出;
> 3. 把 OPEN 表的第一个节点取出放入 CLOSE 表，并计该节点为 n;
> 4. 考察节点 n 是否为目标节点。若是，则求得了问题的解，退出;
> 5. 扩展节点 n, 生成一组子节点。把其中不是节点 n 先辈的那些子节点记做集合 M，并把这些子节点作为节点 n 的子节点加入 G 中;
> 6. 针对 M 中子节点的不同情况，分别进行如下处理:
> 7. 对于那些未曾在 G 中出现过的 M 成员设置一个指向父节点（即节点 n的指针，并把它们放入 OPEN 表;(不在 OPEN 表)
> 8. 对于那些先前已经在 G 中出现过的 M 成员，确定是否需要修改它指向父节点的指针;(在 OPEN 表中, 对 g(x) 进行更新)
> 9. 对于那些先前已在 G 中出现并且已经扩展了的 M 成员，确定是否需要修改其后继节点指向父节点的指针;(在 CLOSE 表中, 对节点 n 子节点的子节点更新 g(x)
> 10. 对 OPEN 表中的节点按估价函数进行排序;
> 11. 转第 2 步。

2.3实验结果

输入：

初始状态：[0,1,2,7,8,3,6,5,4]

结束状态：[1,2,3,8,0,4,7,6,5]

输出：

```
BFS: ['R', 'R', 'D', 'D', 'L', 'L', 'U', 'R']
price: 429
time: 0.0020015239715576l7

A*: ['R', 'R', 'D', 'D', 'L', 'L', 'U', 'R']
price: 21
time: 0.0030026435852050l8
```

2.4实验结论

广度优先搜索算法比较可靠，只要问题有解，这种算法总可以得到解，而且得到的是最优解。但同时也有很大的缺点，例如它的盲目性较大，当目标节点距初始节点较远时将会产生许多无用节点，搜索效率低。相比之下，使用到代价函数的 A* 搜索算法的效率较高。

2.5实验程序

```
main.py
from time import time
from BFS_search import breadth_first_search
from Astar_search import Astar_search
```

```
from puzzle import Puzzle


state=[0,1,2,7,8,3,6,5,4]


Puzzle.num_of_instances=0
t0=time()
bfs=breadth_first_search(state)
t1=time()-t0
print('BFS:', bfs)
print('price:', Puzzle.num_of_instances)
print('time:', t1)
print()

Puzzle.num_of_instances = 0
t2 = time()
astar = Astar_search(state)
t3 = time() - t0
print('A*:', astar)
print('price:', Puzzle.num_of_instances)
print('time:', t3)
print()

puzzle.py
class Puzzle:
    goal_state=[1,2,3,8,0,4,7,6,5]
    heuristic=None
    evaluation_function=None
    needs_hueristic=False
    num_of_instances=0
    def __init__(self,state,parent,action,path_cost,needs_hueristic=False):
        self.parent=parent
        self.state=state
        self.action=action
        if parent:
            self.path_cost = parent.path_cost + path_cost
        else:
            self.path_cost = path_cost
        if needs_hueristic:
            self.needs_hueristic=True
            self.generate_heuristic()
            self.evaluation_function=self.heuristic+self.path_cost
        Puzzle.num_of_instances+=1

    def __str__(self):
        return
str(self.state[0:3])+'\n'+str(self.state[3:6])+'\n'+str(self.state[6:9])


    def generate_heuristic(self):
        self.heuristic=0
        for num in range(1,9):
            distance=abs(self.state.index(num) -
self.goal_state.index(num))
            i=int(distance/3)
```

```
            j=int(distance%3)
            self.heuristic=self.heuristic+i+j

    def goal_test(self):
        if self.state == self.goal_state:
            return True
        return False

    @staticmethod
    def find_legal_actions(i,j):
        legal_action = ['U', 'D', 'L', 'R']
        if i == 0:  # up is disable
            legal_action.remove('U')
        elif i == 2:  # down is disable
            legal_action.remove('D')
        if j == 0:
            legal_action.remove('L')
        elif j == 2:
            legal_action.remove('R')
        return legal_action

    def generate_child(self):
        children=[]
        x = self.state.index(0)
        i = int(x / 3)
        j = int(x % 3)
        legal_actions=self.find_legal_actions(i,j)

        for action in legal_actions:
            new_state = self.state.copy()
            if action == 'U':
                new_state[x], new_state[x-3] = new_state[x-3], new_state[x]
            elif action == 'D':
                new_state[x], new_state[x+3] = new_state[x+3], new_state[x]
            elif action == 'L':
                new_state[x], new_state[x-1] = new_state[x-1], new_state[x]
            elif action == 'R':
                new_state[x], new_state[x+1] = new_state[x+1], new_state[x]

 children.append(Puzzle(new_state,self,action,1,self.needs_hueristic))
        return children

    def find_solution(self):
        solution = []
        solution.append(self.action)
        path = self
        while path.parent != None:
            path = path.parent
            solution.append(path.action)
        solution = solution[:-1]
        solution.reverse()
        return solution

  BFS_search.py
from queue import Queue
```

```
from puzzle import Puzzle


def breadth_first_search(initial_state):
    start_node = Puzzle(initial_state, None, None, 0)
    if start_node.goal_test():
        return start_node.find_solution()
    q = Queue()
    q.put(start_node)
    explored=[]
    while not(q.empty()):
        node=q.get()
        explored.append(node.state)
        children=node.generate_child()
        for child in children:
            if child.state not in explored:
                if child.goal_test():
                    return child.find_solution()
                q.put(child)
    return

Astar_search.py
from queue import PriorityQueue
from puzzle import Puzzle


def Astar_search(initial_state):
    count=0
    explored=[]
    start_node=Puzzle(initial_state,None,None,0,True)
    q = PriorityQueue()
    q.put((start_node.evaluation_function,count,start_node))

    while not q.empty():
        node=q.get()
        node=node[2]
        explored.append(node.state)
        if node.goal_test():
            return node.find_solution()

        children=node.generate_child()
        for child in children:
            if child.state not in explored:
                count += 1
                q.put((child.evaluation_function,count,child))
    return
```