线性代数在机器学习领域的应用

题目：线性代数在机器学习领域的应用

小组成员和专业班级：XXX（计XXX）

组长：XX

成员分工和具体贡献：

- **XX**：完成了文献资料的查询和报告的撰写

机器学习是近年来流行的一种技术，其原理就是通过算法让机器学习输入和输出之间的映射关系，并根据输入和输出不断改进，最终在给定的输入下能给出准确率较高的输出，例如近年流行的NovelAI，可以根据用户输入的关键字或图片生成一张插画，而且插画通常非常精细，足以媲美一些画师的作品，这个技术的推广和普及促进了业界插画绘制水平的进步，也对低端画师造成了冲击，一些简单的插画或者插画绘制过程中机械重复的部分可以交给AI完成，能大大加快插画出图的速度，一些低端画师也因为NovelAI的竞争而失去了部分市场，促使他们提高自身的绘制水平，倒逼画师水平的提升，有利于画师创造出更加有创造力的作品。我选择线性代数在机器学习领域的应用，是因为在线性代数开课前我就已经开始尝试学习机器学习，在了解到机器学习领域对理论基础例如线性代数的作用时，就已经领悟到了线性代数的优点。于是在接下来的课程学习中，我一直是以帮助学习机器学习的态度学习线性代数

本文的机器学习使用的框架以Python的PyTorch，Numpy，Pandas等框架为主，主要介绍机器学习算法中涉及到线性代数的部分，还介绍机器学习框架中对线性代数运算的支持。

PyTorch是一个Python的机器学习框架，它封装了DataLoader，DataSet，LossFunction，NeuralNetwork，optimizer等在机器学习实践中常常需要用到的组件。具体展开，DataSet实现了数据集的分块，便于每次取出固定长度的数据用于训练，DataLoder在DataSet的基础上实现了数据打乱，同时它还是一个迭代器，能在训练过程中方便的取出数据用于训练，NeuraNetwork有很多种，常见的例如RNN(Recurrent Neural Network，循环神经网络)，CNN(Convolutional Neural Network，卷积神经网络)，它们是机器学习的核心，通过调节神经网络的参数就能改进神经网络，PyTorch内置了很多常见的神经网络供用户使用，网络一般分为不同的层，每层的功能不同，例如有的层负责处理输入数据，减少特征量，有的则是生成输出的核心部分。LossFunction则是损失函数，它负责计算神经网络的输出和期望输出的偏差，这种偏差往往是很复杂，难以评估的，例如NovelAI的输出，因为插画的水平很难以具体衡量，而LossFunction就负责计算这种损失，损失是用于评价神经网络的重要参数，是机器学习能够不断改进自身以获得更高准确率的先决条件。optimizer就是优化器，它根据LossFunction计算出的损失优化网络，完成后向传播

有了对PyTorch的了解，下面来介绍一种经典的算法——梯度下降，梯度下降使用了线性代数的知识简化表达，并利用线性代数的计算方法优化网络。考虑一个简单的情况，假设训练集只有一个特征，设拟合函数为

$$h_\theta(x) = \theta_0 + \theta_1 x$$

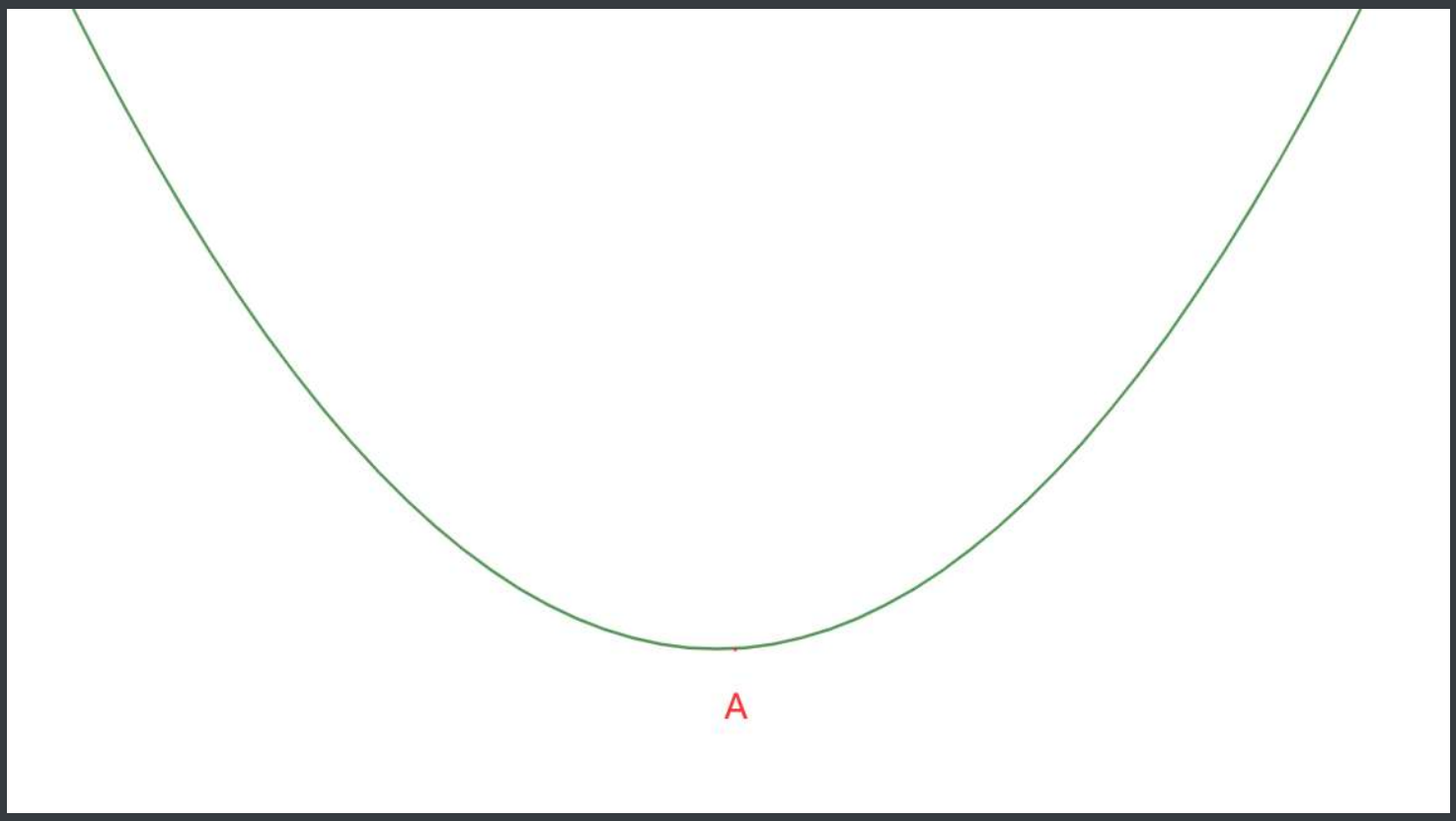
其中，θ_0是bias(偏差)，θ_1是weight（权重），θ_0和θ_1就是这个神经网络（只有一个单元）的权值

设期望值为y，由于我们在不断调节权值使损失函数达到最小值，所以把损失函数表达为θ_0和θ_1的参数，于是问题转化为求θ_0和θ_1取何值时损失函数取得最小值

损失函数为

$$J(\theta_0, \theta_1) = \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

其中i表示第i个样本



如图，横坐标为θ_i纵坐标为损失，如果现在处在A坐标的点，此处的导数小于零，为了取得最小值应该往右移动，如果处在A左边的点，为了取得最小值应该往左移动，于是有

先对$J(\theta_0, \theta_1)$求导

$$\frac{d}{d\theta_i} J(\theta_0, \theta_1)$$

然后选择一个步长参数α

$$\alpha \frac{d}{d\theta_i} J(\theta_0, \theta_1)$$

再把计算出来的步长减到θ_i上（注意，一定要先算出全部右值然后同步赋给θ_i!!!）

$$\theta_i = \theta_i - \alpha \frac{d}{d\theta_i} J(\theta_0, \theta_1)$$

以上操作对每个θ都执行若干次，不断更新权值，网络预测的值就能达到拟合

以上只是特征只有一个的情况下，但是，实践中输入往往有大量的特征，上面这种表示方法计算和表示就有所不便

例如，拟合函数为

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_4 + \theta_5 x_5 \dots$$

于是在这里引入矩阵简化表达和运算

首先定义输入矩阵$X = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 & \dots & x_n \end{bmatrix}$

注意此处的x_0是常数，一般取$x_0 = 1$

然后定义权值矩阵$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \dots \\ \theta_n \end{bmatrix}$

再修改一下h_θ的定义，使其能接收矩阵作为自变量，于是

$$h_\theta(X) = X\theta$$

可见使用矩阵组织数据和表达运算带来了极大的便利。下面计算损失函数

$$J(\theta) = \frac{1}{2}(X\theta - Y)^T(X\theta - Y)$$

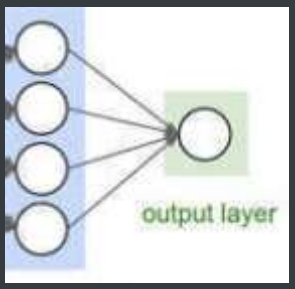
使用矩阵描述梯度下降算法的步骤

1. 随机初始化权值矩阵$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \cdots \\ \theta_n \end{bmatrix}$，要注意的是，不能全部初始化为$0$，否则梯度下降算法无效
2. 获取输入矩阵**X**，计算出损失函数$J(\theta)$
3. 计算步长$\alpha \frac{d}{d\theta} J(\theta)$（此处使用了矩阵的微分），同时更新$\theta = \theta - \alpha \frac{d}{d\theta} J(\theta)$，（由于使用了矩阵表示运算，所以此处的更新也自然是同步更新了，这是矩阵带来的便利）
4. 重复2，3步直至观察到损失函数到达一个较低值而且一段时间内不再下降，说明模型已经拟合

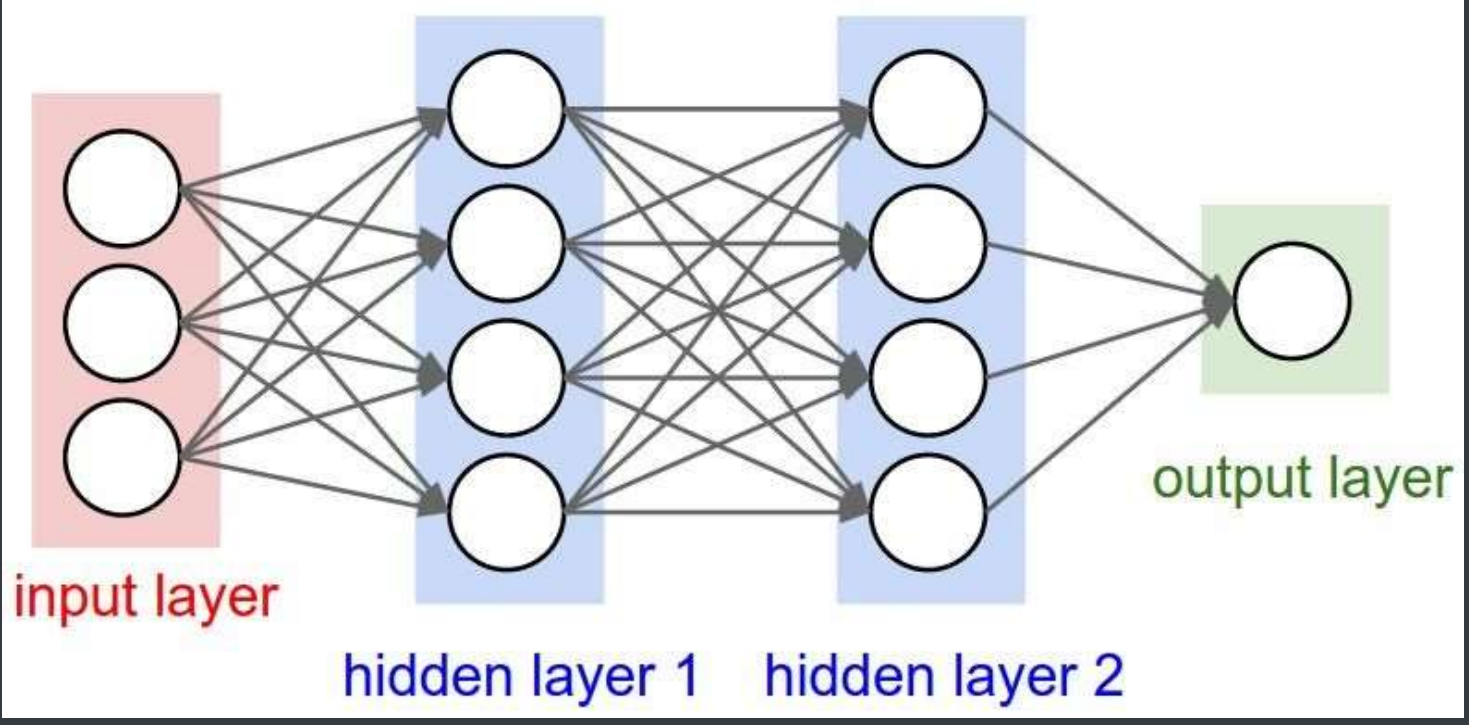
从梯度下降的例子中我们看出，引入了矩阵表述为梯度下降带来了极大的便利，然而还有几点在实践上的便利

1. 矩阵在计算机中表达简单，目前我们之学过二维的矩阵，但是机器学习中往往面临高维矩阵，例如一张彩色图片的输入，有像素的横坐标，纵坐标，像素的**RGB**值，总共三个特征，如果用矩阵表示，需要一个三维的矩阵，但是不管多高维的矩阵，在计算机中都能以约定的方式放入内存，并以约定的方式从内存中读取出来（例如**C**风格的行优先，**Fortran**的列优先），矩阵还有一个好处是，矩阵的存储是连续存储，随机访问速度很快
2. 把一个变量的运算转化成了矩阵的运算。矩阵的运算有许多加快运算速度技巧，另外，在**PyTroch**中内置了专门的**C++**运算引擎处理矩阵的运算，能高效完成运算任务，自动弥补浮点运算误差，加快模型训练速度

以上描述的网络是只有一层的线性回归，可以用下图表示



左边的层有若干个神经元，他们获取输入并计算出激励（例如**sigmoid**激励函数），输出层综合激励并输出（还有**resize**的操作）。但是这样的网络结构太简单了，无法拟合更加复杂的情况，因为它的参数实在太少了，拟合能力有限，实际中往往把多个层复合起来，提高模型复杂度，增强拟合能力



前文所说的激励指前一层运算的结果并不能直接传递给下一层，需要经过一个函数“加工”再输入给下一层，否则改变权值时输出变化太大。常见的激励函数为有sigmoid，ReLU等等，其中sigmoid被认为模拟了真实神经元的行为，在早期神经网络中被大量使用

在Pytorch中，对训练神经网络的各个环节都提供了专用的工具简化开发，例如定义神经网络，有Pytorch提供的各种层（例如ReLU层，Conv（卷积层）），还有应用了特定算法的损失函数（例如利用交叉熵计算损失），还有引应用了特定算法的优化器用于后向传播（例如Adam）

下面来介绍机器学习框架对线性代数运算的支持

- Numpy

Numpy是python科学计算的一个第三方库，其内置了很多数据结构和运算函数，底层用C编写所以运算速度非常快，对矩阵运算的支持也很完备，对MATLAB的支持也很好，复刻了MATLAB很多函数，线性代数中很多运算都能完成

下面我们用 `IPython` 来演示，首先创建两个2×3的 `matrix` 对象

```
>>>import numpy as np
>>>a1 = np.mat([[1, 2, 3], [4, 5, 6]])
matrix([[1, 2, 3],
        [4, 5, 6]])
>>>a1 = np.mat([[1, 2, 3], [4, 5, 6]])
matrix([[2, 3, 4],
        [5, 6, 7]])
```

`matrix` 对象能实现矩阵的运算，例如矩阵加法减法，矩阵数乘，矩阵乘法，还能实现矩阵转置，矩阵求逆，矩阵求迹，矩阵求特征值和特征向量等等

```
a1 = np.mat([[1,1,1],[0,1,0],[0,0,1]])
a1
matrix([[1, 1, 1],
        [0, 1, 0],
        [0, 0, 1]])
np.linalg.pinv(a1)
matrix([[ 1.00000000e+00, -1.00000000e+00, -1.00000000e+00],
        [-3.83438578e-17,  1.00000000e+00, -3.83122027e-17],
        [ 2.95935941e-17, -3.13159935e-17,  1.00000000e+00]])>>>a1+a2
matrix([[ 3,  5,  7],
        [ 9, 11, 13]])
>>>a1-a2
matrix([[-1, -1, -1],
        [-1, -1, -1]])
>>>3*a1
matrix([[ 3,  6,  9],
        [12, 15, 18]])
>>>a1@a2
ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, with gufunc signature (n?,k),(k,m?)->(n?,m?) (size 2 is different from 3)           # matrix重载了@运算符表示矩阵相乘，此处报错是因为a1和a2都是2*3矩阵，不能相乘
>>>a1@a2.reshape(3, 2)
a1@a2.reshape(3,2)
matrix([[28, 34],
        [64, 79]])          # 使用了reshape方法把a2变成3*2的矩阵，于是a1和a2就能相乘了
>>>a1 = np.mat([[1,1,1],[0,1,0],[0,0,1]])
>>>a1
matrix([[1, 1, 1],
        [0, 1, 0],
        [0, 0, 1]])
>>>a1.transpose()           # numpy求矩阵的转置
matrix([[1, 0, 0],
        [1, 1, 0],
        [1, 0, 1]])
>>>np.linalg.pinv(a1)       #numpy求矩阵的逆，对应MATLAB的pinv函数
matrix([[ 1.00000000e+00, -1.00000000e+00, -1.00000000e+00],
        [-3.83438578e-17,  1.00000000e+00, -3.83122027e-17],
        [ 2.95935941e-17, -3.13159935e-17,  1.00000000e+00]])
>>>a1.trace()               # numpy计算矩阵的迹
matrix([[3]])
>>>np.linalg.matrix_rank(a1)    # numpy计算矩阵的秩
```

```
3
>>>np.linalg.det(a1)        #numpy计算矩阵的行列式
1.0
>>>np.linalg.eig(a1)        #numpy计算矩阵的特征值和特征向量
(array([1., 1., 1.]), matrix([[ 1.00000000e+00, -1.00000000e+00, -1.00000000e+00],
        [ 0.00000000e+00,  2.22044605e-16,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  2.22044605e-16]]))
```

可以发现一点，numpy所有的矩阵高级操作，例如求秩，行列式，特征值特征向量的函数都在包 `linalg` 下， `linalg` 就是 `Linear Algebra` 的缩写，既线性代数

- Pytorch

Pytorch是python机器学习的框架，与之相竞争的还有TensorFlow。Pytorch有一个基本的数据结构叫做Tensor（张量），它的效果与Numpy类似，都能表示矩阵，但不同的是它还可以放到GPU上（数据储存在GPU显存，使用GPU超强的并行计算能力加快运算），同样，Pytorch的运算核心部分都是用C编写的，运算速度也很快。

```
>>>a1 = torch.tensor([[1, 2, 3],[2, 3, 4]])
>>>a2 = torch.tensor([[2, 3, 4],[5, 6, 7]])
a1 +a2
tensor([[ 3,  5,  7],
        [ 7,  9, 11]])
>>>a1 -a2
tensor([[-1, -1, -1],
        [-3, -3, -3]])
>>>a1*a2
tensor([[ 2,  6, 12],
        [10, 18, 28]])
>>>a1@a2
RuntimeError: mat1 and mat2 shapes cannot be multiplied (2x3 and 2x3)       # 同样a1和a2不能相乘
>>>a2.resize_(3,2)          #In Place 原地修改矩阵size
tensor([[2, 3],
        [4, 5],
        [6, 7]])
>>>a1@a2
tensor([[28, 34],
        [40, 49]])
```

下面展示如何把声明矩阵放到GPU

```
>>>device = torch.device("cuda" if torch.cuda.is_available() else "cpu")    #检测Nvidia GPU的CUDA驱动是否可用，如果可用就使用CUDA
>>>device
device(type='cpu')      #因为我并没有Pytorch支持的显卡，所以此处只能把Tensor放到CPU上
>>>a1 = a1.to(device)       #把张量a1放到device上（此处为CPU）
```

- **Pandas**

pandas是python的数据分析领域的第三方库，比起Numpy，pandas更重加工数据，Numpy更重运算，pandas的一个常用数据结构就是DataFrame，DataFrame不像矩阵而更像表，它能把不同类型的数据存储到一个DataFrame对象中，但DataFrame也能进行矩阵的运算。

下面演示Pandas的矩阵运算

```
import pandas as pd
>>>d1 = pd.DataFrame([[1,2,3],[2,3,4]])
>>>d1       # 第一行和第一列都是索引Index，其他才是数据
   0  1  2
0  1  2  3
1  2  3  4
>>>d2 = pd.DataFrame([[2,3,4],[5,6,7]])
>>>d2
   0  1  2
0  2  3  4
1  5  6  7
>>>d1+d2
   0  1   2
0  3  5   7
1  7  9  11
>>>d1*d2
    0   1   2
0   2   6  12
1  10  18  28
>>>d1-d2
   0  1  2
0 -1 -1 -1
1 -3 -3 -3
>>2*d1
   0  1  2
0  2  4  6
1  4  6  8
>>>d2.T     #DataFrame转置
```

```
29      0  1
30  0   2  5
31  1   3  6
32  2   4  7
33  >>>d1@d1.T       #三个库都是重载@运算符表示矩阵乘法
34      0   1
35  0   14  20
36  1   20  29
```

综合来看numpy，pytorch，pandas，三个第三方库都有对矩阵的支持，却各有特点，numpy的主要目的就是计算，所以对矩阵的支持最完备，pytorch重神经网络开发，从其能把Tensor放在GPU上就能看出来，pandas重在对数据加工整理清洗，所以功能上偏向了实用，所以pytorch和pandas对矩阵的支持都不如numpy

上文介绍了线性代数在机器学习的应用，还提及了机器学习框架对矩阵的支持，从中可以看出线性代数的工具性很强，能整合处理大量数据，在面对大量数据时能发挥很大作用，随着机器学习的发展，线性代数在计算机领域的应用也将越来越多