

计算智能 课程设计指导书

计算机科学与工程学院

内部使用

湖南科技大学

2021 年 11 月

一、课程设计目的

通过用 Python语言编程实现神经网络、进化计算、模糊逻辑领域一些重要的算法，让学生掌握感知机、多层神经网络结构模型、前向传播计算、BP反向传播算法等基本原理，能够运用多层神经网络解决具体问题；掌握遗传算法的基本原理及其实现，能够运用遗传算法求解优化问题；掌握模糊计算的基本原理，理解模糊集合、模糊关系的运算，掌握运用模糊理论实现模糊聚类的方法。


二、课程设计要求

1. 课程设计报告要求独立完成、不允许相互抄袭。否则不及格。
2. 遵守课程设计在线教学的规章制度，按时签到，不得在课程设计期间做与课程设计无关的事情。对于课程设计过程中出现的问题，及时向指导老师汇报。每天按时填写课程设计日志（格式参考附录 2），并接受指导老师的定期检查。没有日志的学生课程设计不及格。
3. 最终的课程设计报告要求格式规范，语言通顺，写明使用的数据结构，画出算法的程序流程图、关键的源程序代码、仔细记录实验结果，对于遇到的问题或疑惑要一并书写，并作为重点加以思考与分析。不可直接抄袭指导书中的内容，违规者最终成绩直接不及格。
4. 及时提交课程设计报告（请参见附录 1 的课程设计报告格式要求，除封面之外的内容建议使用双面打印）、源程序文件、日志文件。每个实验的报告篇幅建议大约 3-4 页，总课程设计报告页面数大约 25 页左右。

二、课程设计实验环境

WINDOWS+Anaconda3 套件 4.10.3（请优先使用老师提供的 Anaconda3-2021.05-Windows-x86_64.exe进行安装），含Python编译环境及相关的扩展库，包括Python 3.8.8、SciPy 1.6.2、NumPy 1.20.1、Matplotlib 3.3.4（本实验不需要使用TensorFlow等深度学习框架，不需要使用GPU）。

Anaconda3套件4.10.3默认安装了编译开发环境Spyder 4.2.5和jupyter notebook 6.3.0，可选安装jupyterlab 3.0.14，这些都可以作为开发工具，当然也可以自行选择其它开发工具如Pycharm、Eclipse、Sublime Text、VSCODE、IDLE等。本教程默认使用Spyder。



```
命令提示符
Microsoft Windows [版本 10.0.19042.1348]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\user>conda -V
conda 4.10.3

C:\Users\user>python -V
Python 3.8.8

C:\Users\user>python -c "import numpy; print (numpy.__version__)"
1.20.1

C:\Users\user>python -c "import matplotlib; print (matplotlib.__version__)"
3.3.4

C:\Users\user>python -c "import scipy; print (scipy.__version__)"
1.6.2

C:\Users\user>
```

三、课程设计考核

1 本指导书中的五个实验为验证性实验。其中实验一、实验二、实验五各有一道标明【可选】的小题，可选做。另外实验三有一定难度，整体可选做。

2 评分标准，由指导老师根据实际情况把握，建议：完成至少三个【可选】才能评“优”，完成至少两个【可选】可评“良好”。

3 课程设计成绩主要由以下几部分组成：平时成绩、测试成绩、创新成绩设计、设计报告质量(详见附件 3)。学生记录课程设计过程和结果（或用到的数据结构、算法等），形成日志记录文档，计入平时成绩；指导老师进行分段测试，在课程设计中期和末期分别进行测试，形成测试成绩；指导老师采用个人陈述、小组答辩（报告同学须自评小组贡献度）等方式考察学生课设成果的创新性，形成创新成绩。

实验一 基于感知机的鸢尾花分类

1、实验目的

利用感知机算法对鸢尾花种类进行分类, 要求熟悉感知机算法, 掌握利用Python实现机器学习算法的一般流程, 了解scikit-learn机器学习库的使用。

2、背景知识

植物的分类与识别是植物学研究和农林业生产经营中的重要基础工作, 对于区分植物种类、探索植物间的亲缘关系、阐明植物系统的进化规律具有重要意义。传统识别植物的方法主要依靠人工, 需要丰富的专业知识, 工作量大, 效率不高, 而且难以保证分类的客观性和精确性。随着信息技术飞速发展, 将计算机视觉、模式识别、数据库等技术应用于植物种类识别, 使得识别更加简单、准确、易行。相对于植物的其它部分, 其花朵图像更容易获取, 花朵的颜色和形状等都可作为分类依据。本案例在提取花朵形态特征的基础上, 利用感知机算法进行分类与识别。

鸢(读音同“愿”)尾花, 英文为 Iris, 多年生草本植物, 花大而美丽, 叶片青翠碧绿, 观赏价值很高, 是一种重要的观赏用庭园植物。其外形作为独特的徽章, 在欧洲的宗教、皇室建筑上被广泛应用。特别是在法国, 印在法国军旗上的三朵鸢尾花图案标志着法兰西王国的王权。

鸢尾花有三个主要类型(种属): Setosa 山鸢尾、 Versicolour 变色鸢尾 和 Virginica 维吉尼亚鸢尾, 其主要区别是萼片长度、萼片宽度、花瓣长度和花瓣宽度。



Setosa
山鸢尾



Versicolor
变色鸢尾



Virginica
维吉尼亚鸢尾

鸢尾花数据集: scikit-learn 是基于 Python 的机器学习库, 其默认安装包含了几小型的数据集, 并提供了读取这些数据集的接口。其中 `sklearn.datasets.load_iris()` 用于读取鸢尾花数据集, 该数据集有 150 组 3 种类型鸢尾花的 4 种属性: 萼片长度 `sepal length`、萼片宽度 `sepal width`、花瓣长度 `petal length` 和花瓣宽度 `petal width`, 样本编号与类型的关系是: 样本编号 0 至 49 为 Setosa ,50 至 99 为 Versicolour ,100 至 149 为 Virginica。

3、示例代码

1) 加载用到的库

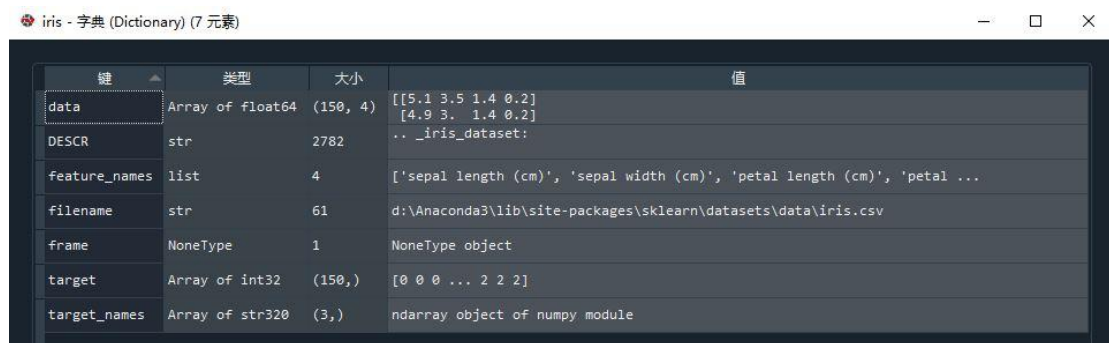
```
1. import numpy as np
2. import matplotlib.pyplot as plt
3. from sklearn.datasets import load_iris #仅用于加载数据集
```

2) 加载鸢尾花数据集

```
1. iris = load_iris()
```

如前所述, scikit-learn 提供了读取鸢尾花数据集的接口 `sklearn.datasets.load_iris()`, 该数据集有 150 组 3 种类型鸢尾花的 4 种属性: 萼片长度 `sepal length`、萼片宽度 `sepal width`、花瓣长度 `petal length` 和花瓣宽度 `petal width`, 样本编号与类型的关系是: 样本编号 0 至 49 为 `Setosa`, 50 至 99 为 `Versicolour`, 100 至 149 为 `Virginica`。

代码 `iris = load_iris()` 中得到的 `iris` 是一个字典, 包含七个 “key-value” 对:



The screenshot shows a Jupyter Notebook cell with the title "iris - 字典 (Dictionary) (7 元素)". It displays the structure of the `iris` dictionary. The table below summarizes the content shown in the screenshot.

键	类型	大小	值
<code>data</code>	Array of float64	(150, 4)	[[5.1 3.5 1.4 0.2] [4.9 3. 1.4 0.2] .. _iris_dataset:]
<code>DESCR</code>	str	2782	.. _iris_dataset:
<code>feature_names</code>	list	4	['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal ...
<code>filename</code>	str	61	d:\Anaconda3\lib\site-packages\sklearn\datasets\data\iris.csv
<code>frame</code>	NoneType	1	NoneType object
<code>target</code>	Array of int32	(150,)	[0 0 0 ... 2 2 2]
<code>target_names</code>	Array of str320	(3,)	ndarray object of numpy module

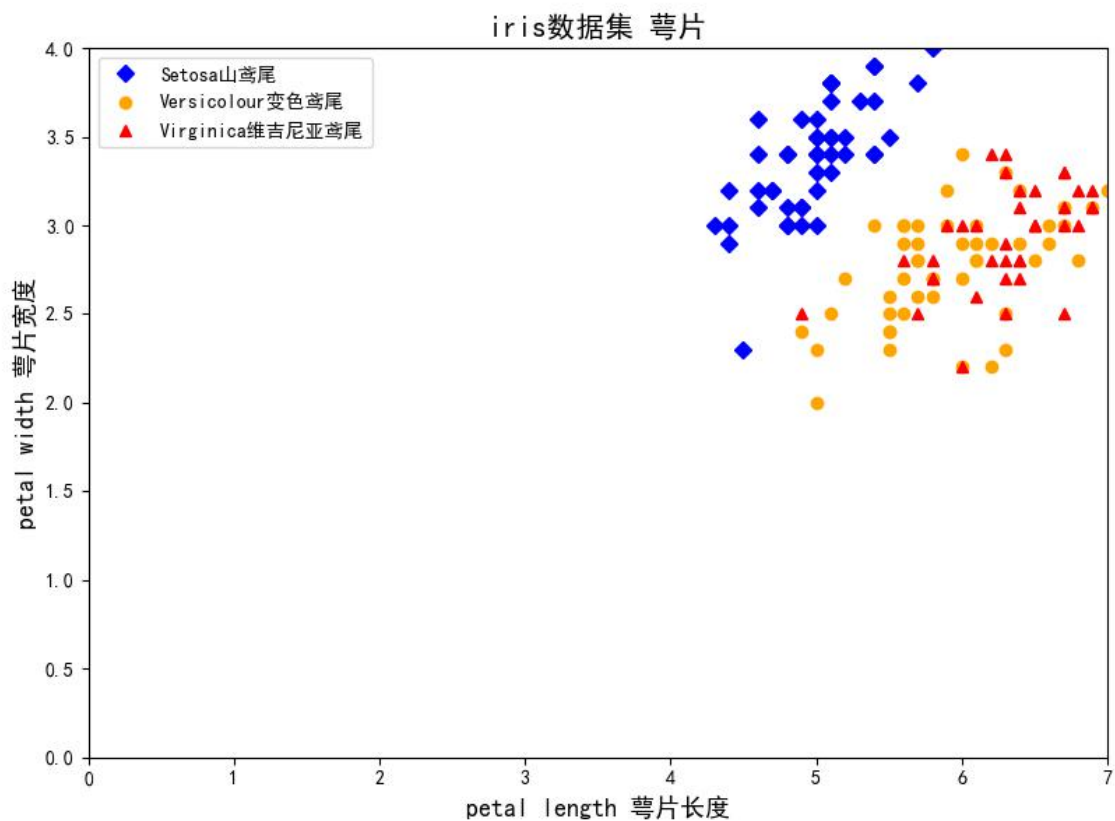
其中每一个 key 的意义和 value 说明如下:

	data	DESCR	feature_names	filename	frame	target	target_names	...
	数据集包含150个数据, 分为3类, 每类50个数据, 每个数据包含4个属性	关于本数据集的文字描述	4个属性的名字和单位	本数据集对应的csv文件	仅当load_iris(as_frame=True)时有效, 此时frame是一个DataFrame	目标值, 又叫真值, 有0、1、2三种	真值所对应的植物, 0对应Setosa, 1对应Versicolour, 2对应Virginica	...
0	5.1 3.5 1.4 0.2	...	sepal length (cm)	0	Setosa	...
1	4.9 3.0 1.4 0.2	...	sepal width (cm)	0	Versicolour	...
2	4.7 3.2 1.3 0.2	...	petal length (cm)	0	Virginica	...
3	4.6 3.1 1.5 0.2	...	petal width (cm)	0
...
49	5.0 3.3 1.4 0.2	0
50	7.0 3.2 4.7 1.4	1
...
99	5.7 2.8 4.1 1.3	1
100	6.3 3.3 6.0 2.5	2
...
148	6.2 3.4 5.4 2.3	2
149	5.9 3.0 5.1 1.8	2

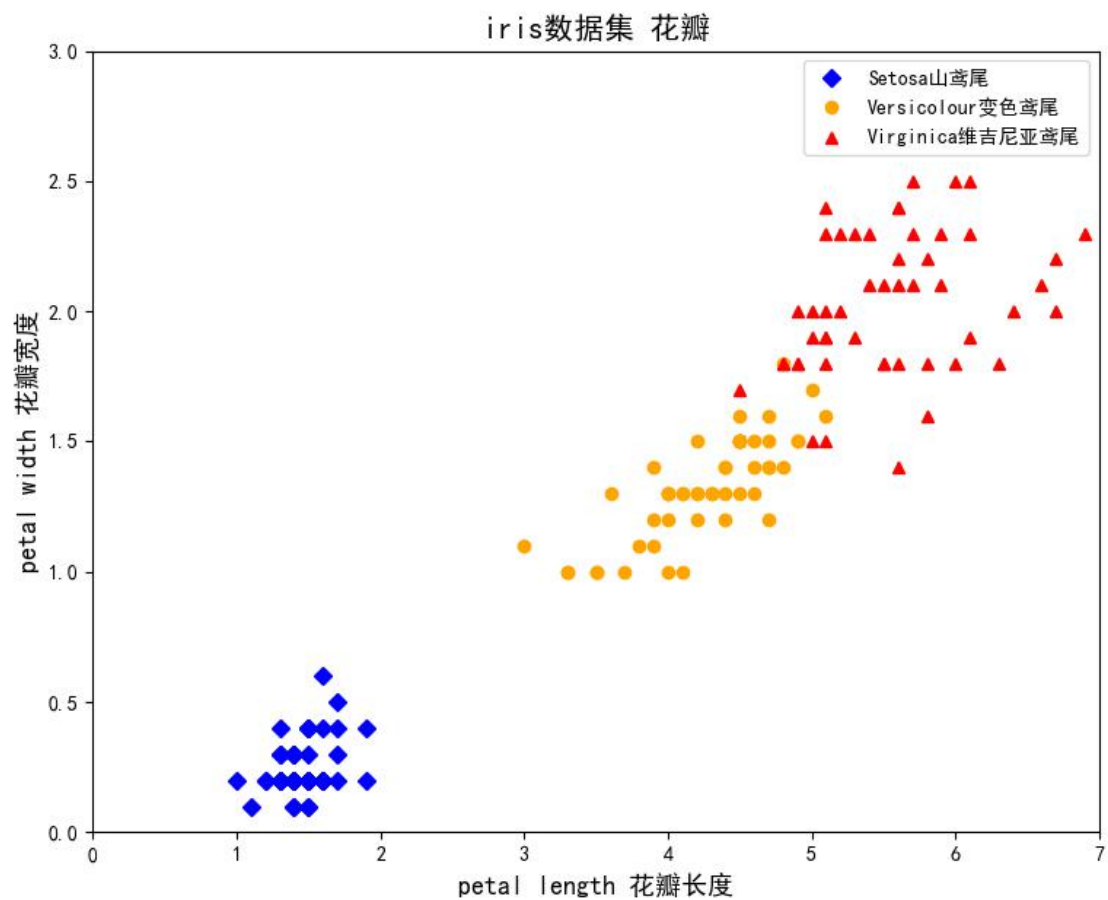
3) 通过画图了解三种鸢尾花的分布

```
1. plt.clf()
2. plt.xlim(0, 7)#x 轴上的最小值和最大值
3. plt.ylim(0, 4)
4. plt.title(u'iris 数据集 萼片', fontsize=15)
5. X=iris.data[:,0:2]
6.
7. plt.xlabel('petal length 萼片长度', fontsize=13)
8. plt.ylabel('petal width 萼片宽度', fontsize=13)
9. plt.plot(X[:50, 0], X[:50, 1], 'bo', color='blue', label='Setosa 山鸢尾')
10. plt.plot(X[50:100, 0], X[50:100, 1], 'bo', color='orange', label='Versicolour 变色鸢尾')
11. plt.plot(X[100:150, 0], X[100:150, 1], 'bo', color='red', label='Virginica 维吉尼亚鸢尾')
12. plt.legend()
13.
14. plt.show()
```

生成图片如下:



从图中大致可以看出，萼片长度和萼片宽度与鸢尾花类型间呈现出非线性关系。



从图中大致可以看出，花瓣长度和花瓣宽度与鸢尾花类型间有较好的线性关系，使用花瓣数据

来划分鸢尾花类型效果更好。

4) 算法初始化

```
1. X=np.c_[np.ones(100),iris.data[:100,2:4]]      #输入 X
2. T=iris.target[:100].reshape(100,1)            #真值 T
3. T[T!=1] = -1                                   #将 0 用-1 表示, 以契合 sign 函数
4. W = np.array([[ -7],                           #权值初始化, 3 行 1 列, 即 w0 w1 w2
5.                [ 1],
6.                [ 1]])
7. lr = 1                                          # 学习率设置
8. Y = 0                                          # 神经网络输出
```

`np.ones(100)` 生成 100 行 1 列的 1, 作为偏置的输入。

`iris.data[:100,2:4]` 得到前 100 行,2 列和 3 列数据, 作为两个特征(花瓣长度、花瓣宽度)的输入。

`np.c_`是按行连接两个矩阵, 就是把两矩阵左右相连形成一个新矩阵, 要求行数相等。

组合两个特征和偏置, 形成最终的输入 `X`。

`X=(x0 x1 x2)`, 即偏置、花瓣长度、花瓣宽度。

真值 `T` 直接从数据集中得到, 然后将 `T` 中所有不等于 1 的元素赋值为-1, 以契合接下来将要使用的 `sign` 函数。

5) 学习算法

```
1. def train():
2.     global W
3.     Y = np.sign(np.dot(X,W))
4.     E = T - Y
5.     delta_W = lr * (X.T.dot(E)) / X.shape[0]
6.     W = W + delta_W
```

请参考代码注释加以理解。

6) 画图函数

```
1. def draw():
2.     plt.clf()
3.     plt.xlim(0, 6)#x 轴上的最小值和最大值
4.     plt.ylim(0, 2)#y 轴上的最小值和最大值
5.     plt.title(u'Perceptron 感知器 epoch:%d\n W0:%f W1:%f W2:%f' %(i+1,W[0],W[1],W[2]), fontsize=15)
6.
7.     plt.xlabel('petal length 花瓣长度', fontsize=13)
8.     plt.ylabel('petal width 花瓣宽度', fontsize=13)
9.
10.    plt.plot(X[:50, 1], X[:50, 2], 'bo', color='red', label='Setosa 山鸢尾')
11.    # 用黄色的点来画出负样本
12.    plt.plot(X[50:100, 1], X[50:100, 2], 'bo', color='blue', label='Versicolour 变色鸢尾')
13.    plt.plot(2.5, 1, 'b+', color='black', label='待预测点')
14.
15.    k = - W[1] / W[2]
16.    d = -W[0] / W[2]
17.    # 设定两个点
18.    xdata = (0,6)
19.    # 通过两个点来确定一条直线, 用红色的线来画出分界线
20.    plt.plot(xdata,xdata * k + d,'black', linewidth=3)
21.    plt.legend()
22.    #####以下绘制决策面两边的颜色, 不要求掌握
23.    # 生成决策面
24.    from matplotlib.colors import ListedColormap #绘制决策面两边的颜色, 不要求掌握
```



```

25.     # 生成 x,y 的数据
26.     n = 256
27.     xx = np.linspace(0, 6, n)
28.     yy = np.linspace(0, 2, n)
29.     # 把 x,y 数据生成 mesh 网格状的数据, 因为等高线的显示是在网格的基础上添加上高度值
30.     XX, YY = np.meshgrid(xx, yy)
31.     # 填充等高线
32.     colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
33.     cmap = ListedColormap(colors[:len(np.unique(np.sign(W[0]+W[1]*XX+W[2]*YY)))]))
34.     plt.contourf(XX, YY, np.sign(W[0]+W[1]*XX+W[2]*YY),8, alpha = 0.5, cmap=cmap)
35.     #####以上绘制决策面两边的颜色, 不要求掌握
36.     plt.pause(0.1)
37.     plt.show()

```

请参考代码注释加以理解。

7) 主函数

```

1. # 训练 1000 次
2. for i in range(1000):
3.     if(i==0):     #特地画出未经训练的初始图像, 以方便理解
4.         draw()
5.         plt.pause(5) #停留两秒, 这是分类直线最初的位置, 取决于 w 的初始值, 是人为决定的超参数
6.         train()     #更新一次权值
7.         draw()     #画出更新一次权值后的图像
8.         Y = np.sign(np.dot(X,W))
9.         # .all()表示 Y 中的所有值跟 T 中所有值都对应相等, 结果才为真
10.        if(Y == T).all():
11.            print('Finished')
12.            # 跳出循环
13.            break

```

请参考代码注释加以理解。

4、实验内容

请回答下列问题：

- 1) 考虑学习率的作用。修改示例代码, 固定初始权值=(1,1,1), 将学习率分别设定为 1、0.5、0.1 (组合 1~3), 程序在 epoch 等于多少时实现分类?
- 2) 考虑初始权值的作用。修改示例代码, 固定学习率=0.1, 将初始权值分别设定为(-1,1,1)、(+1,-1,-1)、(1,-1,+1)、(-1,+1,-1) (组合 4~7), 程序在 epoch 等于多少时实现分类?
- 3) 示例程序使用的是离散感知机还是连续感知机? 如何判断?
- 4) 为什么在学习算法中要除以 X.shape[0]? 示例程序采用的是批量下降还是逐一下降? 是否属于随机下降? 是否属于梯度下降?
- 5) 假设你在自然界找到了一朵鸢尾花, 并测得它的花瓣长度为 2.5cm, 花瓣宽度为 1cm, 它属于哪一类? 在 draw()中已用 plt.plot 画出这个'待预测点'。请观察 1~7 这 7 种组合中, 感知机的判断始终一致么? 这说明它受到什么因素的影响?
- 6) 修改示例代码, 将变色鸢尾的数据替换为维吉尼亚鸢尾, 再进行分类。
即横轴为花瓣长度, 纵轴为花瓣宽度, 数据为 Setosa 山鸢尾+Virginica 维吉尼亚鸢尾。
- 7) 【可选】目前感知机只有两个输入+偏置, 如果有三个输入 (比如增加萼片长度作为输入), 程序应如何修改 (可以不画图)?

实验二 以人事招聘为例的误差反向传播算法

1、实验目的

理解多层神经网络的结构和原理，掌握反向传播算法对神经元的训练过程，了解反向传播公式。通过构建 BP 网络实例，熟悉前馈网络的原理及结构。

2、背景知识

误差反向传播算法即 BP 算法，是一种适合于多层神经网络的学习算法。其建立在梯度下降方法的基础之上，主要由激励传播和权重更新两个环节组成，经过反复迭代更新、修正权值从而输出预期的结果。

BP 算法整体上可以分成正向传播和反向传播，原理如下：

正向传播过程：信息经过输入层到达隐含层，再经过多个隐含层的处理后到达输出层。

反向传播过程：比较输出结果和正确结果，将误差作为一个目标函数进行反向传播：

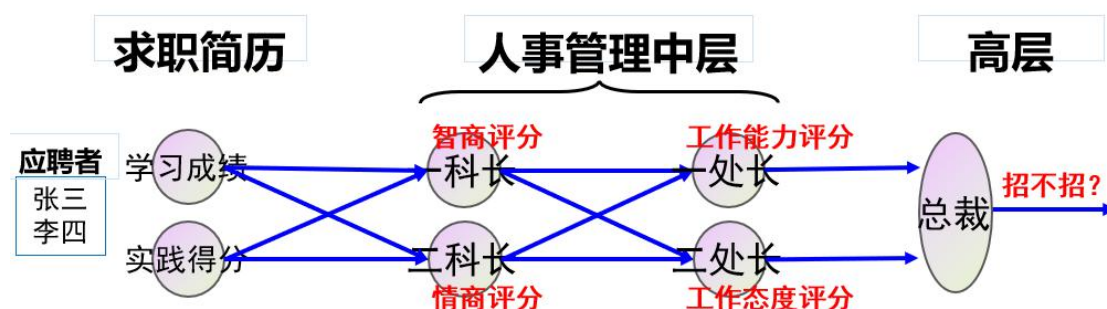
对每一层依次求对权值的偏导数，构成目标函数对权值的梯度，网络权重再依次完成更新调整。依此往复、直到输出达到目标值完成训练。

该算法可以总结为：利用输出误差推算前一层的误差，再用推算误差算出更前一层的误差，直到计算出所有层的误差估计。

1986 年，Hinton 在论文《Learning Representations by Back-propagating Errors》中首次系统地描述了如何利用 BP 算法来训练神经网络。

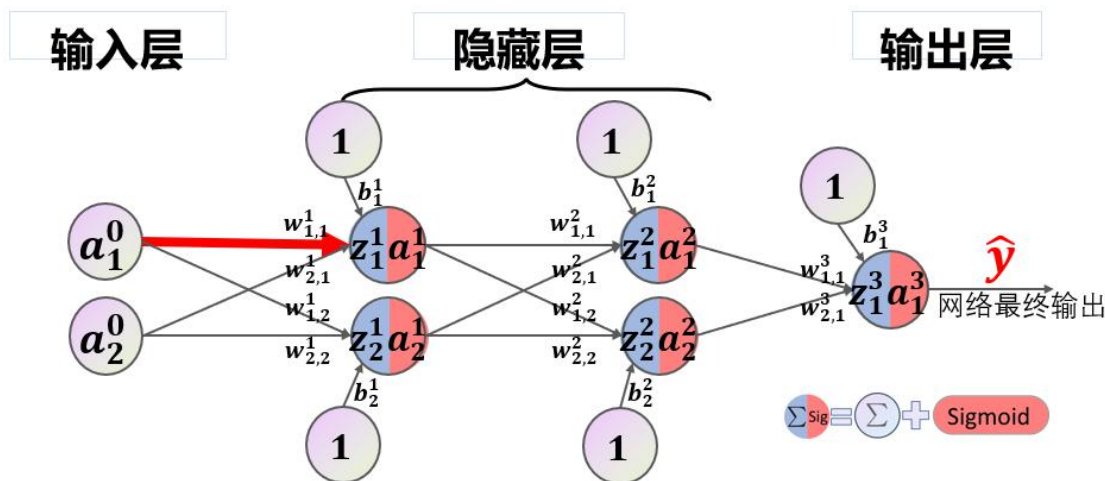
从此，BP 算法开始占据有监督神经网络算法的核心地位。它是迄今最成功的神经网络学习算法之一，现实任务中使用神经网络时，大多是在使用 BP 算法进行训练。

为了说明 BP 算法的过程，本实验使用一个公司招聘的例子：假设有一个公司，其人员招聘由 5 个人组成的人事管理部门负责，如下图所示：



其中张三、李四等人是应聘者，他们向该部门投递简历，简历包括两类数据：学习成绩和社会实践得分，人事部门有三个层级，一科长根据应聘者的学习成绩和实践得分评估其智商，二科长根据同样的资料评估其情商；一处长根据两个科长提供的智商、情商评分，评估应聘者的工作能力，二处长评估工作态度；最后由总裁汇总两位处长的意见，得出最终结论，即是否招收该应聘者。

该模型等价于一个形状为 (2, 2, 2, 1) 的前馈神经网络，输入层、隐藏层 1、隐藏层 2、输出层各自包含 2、2、2、1 个节点，如下图所示。



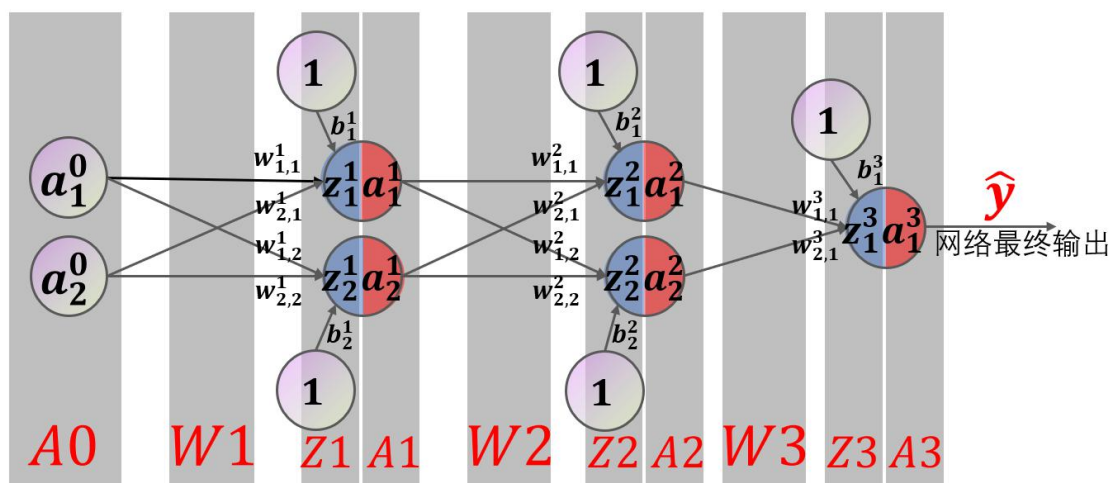
除输入节点外，每个节点都执行**汇总**和**激活**两个操作。汇总得到的数值称为净输入，用字母 z 表示。激活采用 sigmoid 函数，激活后的数据称为该节点的输出，用字母 a 表示。字母的上标代表该节点位于哪一层，下标代表该节点是该层第几个节点。注意输入节点位于 0 层。

节点与节点之间边的权值用字母 w 表示，上标代表该权值属于哪一层，下标有两个，代表其连接的是左侧（上一层）第几个节点到右侧（下一层）第几个节点。**偏置**用字母 b 表示，它可以视为输入恒为 1 的边的权值。

为所有 w 和 b 赋初值，针对一个样本或多个样本从左到右计算出所有 z 和 a 的过程，即正向传播。经正向传播后，神经网络的最终输出，记为 \hat{y} 。在本案例中， $\hat{y} = a_1^3$ 。

通过设计一个由权值和偏置决定的目标函数 $J(W, b)$ ，可以求出目标函数对 \hat{y} 的偏导数 $\frac{\partial J(W, b)}{\partial \hat{y}}$ （目标函数并不一定包含 \hat{y} ，但本案例只讨论这种情况），将该偏导数看成由权值和偏置导致的误差，一层一层将误差反向传导到所有的权值和偏置，就是反向传播过程。

实际编程时进行了向量化（Vectorization），即将对标量的多次循环计算，用对向量、矩阵、张量的一次运算来替代，见下图。应聘者的输入被组织为向量 A_0 ，第一层的 4 个权值被组织为矩阵 W_1 ，隐藏层 1 节点的汇总结果被组织为向量 Z_1 ，对应的输出被组织为 A_1 。其它层类似。最后的 Z_3 和 A_3 在本例中是标量。



3、示例代码

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3.
4. # 输入数据 1 行 2 列
5. X = np.array([[1,0.1]])
6.
7. # 标签，也叫真值，1 行 1 列
8. T = np.array([[1]])
9.
10. # 定义一个 2 隐层的神经网络：2-2-2-1
11. # 输入层 2 个神经元，隐藏 1 层 2 个神经元，隐藏 2 层 2 个神经元，输出层 1 个神经元
12.
13. # 输入层到隐藏层 1 的权值初始化，2 行 2 列
14. W1 = np.array([[0.8,0.2],
15.                [0.2,0.8]])
16. # 隐藏层 1 到隐藏层 2 的权值初始化，2 行 2 列
17. W2 = np.array([[0.5,0.0],
18.                [0.5,1.0]])
19. # 隐藏层 2 到输出层的权值初始化，2 行 1 列
20. W3 = np.array([[0.5],
21.                [0.5]])
22.
23.
24. # 初始化偏置值
25. # 隐藏层 1 的 2 个神经元偏置
26. b1 = np.array([[-1,0.3]])
27. # 隐藏层 2 的 2 个神经元偏置
28. b2 = np.array([[0.1,-0.1]])
29. # 输出层的 1 个神经元偏置
30. b3 = np.array([[-0.6]])
31. # 学习率设置
32. lr = 0.1
33. # 定义训练周期数 10000
34. epochs = 10000
35. # 每训练 1000 次计算一次 loss 值
36. report = 1000
37. # 将所有样本分组，每组大小为
38. batch_size = 1
39.
40. # 定义 sigmoid 函数
41. def sigmoid(x):
42.     return 1/(1+np.exp(-x))
43.
44. # 定义 sigmoid 函数导数
45. def dsigmoid(x):
46.     return x*(1-x)
47.
48. # 更新权值和偏置值
49. def update():
50.     global batch_X, batch_T, W1, W2, W3, lr, b1, b2, b3
51.
52.     # 隐藏层 1 输出
53.     Z1 = np.dot(batch_X, W1) + b1
54.     A1 = sigmoid(Z1)
55.
56.     # 隐藏层 2 输出
57.     Z2 = (np.dot(A1, W2) + b2)
58.     A2 = sigmoid(Z2)
59.
60.     # 输出层输出
61.     Z3 = (np.dot(A2, W3) + b3)
62.     A3 = sigmoid(Z3)
```

```

63.
64.     # 求输出层的误差
65.     delta_A3 = (batch_T - A3)
66.     delta_Z3 = delta_A3 * dsigmoid(A3)
67.
68.     # 利用输出层的误差，求出偏导（即隐藏层 2 到输出层的权值改变）    # 由于一次计算了多个样本，所以
    需要求平均
69.     delta_W3 = A2.T.dot(delta_Z3) / batch_X.shape[0]
70.     delta_B3 = np.sum(delta_Z3, axis=0) / batch_X.shape[0]
71.
72.     # 求隐藏层 2 的误差
73.     delta_A2 = delta_Z3.dot(W3.T)
74.     delta_Z2 = delta_A2 * dsigmoid(A2)
75.
76.     # 利用隐藏层 2 的误差，求出偏导（即隐藏层 1 到隐藏层 2 的权值改变）    # 由于一次计算了多个样本，
    所以需要求平均
77.     delta_W2 = A1.T.dot(delta_Z2) / batch_X.shape[0]
78.     delta_B2 = np.sum(delta_Z2, axis=0) / batch_X.shape[0]
79.
80.     # 求隐藏层 1 的误差
81.     delta_A1 = delta_Z2.dot(W2.T)
82.     delta_Z1 = delta_A1 * dsigmoid(A1)
83.
84.     # 利用隐藏层 1 的误差，求出偏导（即输入层到隐藏层 1 的权值改变）    # 由于一次计算了多个样本，
    所以需要求平均
85.     delta_W1 = batch_X.T.dot(delta_Z1) / batch_X.shape[0]
86.     delta_B1 = np.sum(delta_Z1, axis=0) / batch_X.shape[0]
87.
88.     # 更新权值
89.     W3 = W3 + lr * delta_W3
90.     W2 = W2 + lr * delta_W2
91.     W1 = W1 + lr * delta_W1
92.
93.     # 改变偏置值
94.     b3 = b3 + lr * delta_B3
95.     b2 = b2 + lr * delta_B2
96.     b1 = b1 + lr * delta_B1
97.
98. # 定义空 list 用于保存 loss
99. loss = []
100. batch_X = []
101. batch_T = []
102. max_batch = X.shape[0] // batch_size
103. # 训练模型
104. for idx_epoch in range(epochs):
105.
106.     for idx_batch in range(max_batch):
107.         # 更新权值
108.         batch_X = X[idx_batch*batch_size:(idx_batch+1)*batch_size, :]
109.         batch_T = T[idx_batch*batch_size:(idx_batch+1)*batch_size, :]
110.         update()
111.         # 每训练 5000 次计算一次 loss 值
112.         if idx_epoch % report == 0:
113.             # 隐藏层 1 输出
114.             A1 = sigmoid(np.dot(X,W1) + b1)
115.             # 隐藏层 2 输出
116.             A2 = sigmoid(np.dot(A1,W2) + b2)
117.             # 输出层输出
118.             A3 = sigmoid(np.dot(A2,W3) + b3)
119.             # 计算 loss 值
120.             print('A3:',A3)
121.             print('epochs:',idx_epoch,'loss:',np.mean(np.square(T - A3) / 2))
122.             # 保存 loss 值
123.             loss.append(np.mean(np.square(T - A3) / 2))

```

```

124.
125. # 画图训练周期数与 loss 的关系图
126. plt.plot(range(0,epochs,report),loss)
127. plt.xlabel('epochs')
128. plt.ylabel('loss')
129. plt.show()
130.
131. # 隐藏层 1 输出
132. A1 = sigmoid(np.dot(X,W1) + b1)
133. # 隐藏层 2 输出
134. A2 = sigmoid(np.dot(A1,W2) + b2)
135. # 输出层输出
136. A3 = sigmoid(np.dot(A2,W3) + b3)
137. print('output:')
138. print(A3)
139.
140. # 因为最终的分类只有 0 和 1, 所以我们可以把
141. # 大于等于 0.5 的值归为 1 类, 小于 0.5 的值归为 0 类
142. def predict(x):
143.     if x>=0.5:
144.         return 1
145.     else:
146.         return 0
147.
148. # map 会根据提供的函数对指定序列做映射
149. # 相当于依次把 A2 中的值放到 predict 函数中计算
150. # 然后打印出结果
151. print('predict:')
152. for i in map(predict,A3):
153.     print(i)

```

4、实验内容

请回答下列问题：

1) 如果去掉总裁这一层，相应张三的样本修改为(1.0,0.1,1.0,1.0)，分别对应张三的学习成绩、张三的实践成绩、张三的工作能力真值、张三的工作态度真值，代码应该如何修改？

2) 如果增加一个样本，李四（0.1,1.0,0），分别对应李四的学习成绩，李四的实践成绩，李四被招聘可能性的真值，代码应该如何修改？此时是一个样本计算一次偏导、更新一次权值，还是两个样本一起计算一次偏导、更新一次权值？（提示：注意 `batch_size` 的作用）

3) 样本为张三[1,0.1,1]、李四[0.1,1,0]、王五[0.1,0.1,0]、赵六[1,1,1]，请利用 `batch_size` 实现教材 279 页提到的“批量梯度下降”、“随机梯度下降”和“小批量梯度下降”，请注意“随机梯度下降”和“小批量梯度下降”要体现随机性。

4) 【选做】本例中输入向量、真值都是行向量，请将它们修改为列向量，如 `X = np.array([[1,0.1]])` 改为 `X = np.array([[1],[0.1]])`，请合理修改其它部分以使程序得到与行向量时相同的结果。

实验三 基于神经网络的手写数字识别

1、实验目的

掌握神经网络的设计原理，熟练掌握神经网络的训练和使用方法，能够使用Python语言，针对手写数字分类的训练和使用，实现一个三层全连接神经网络模型。具体包括：

1) 实现三层神经网络模型来进行手写数字分类，建立一个简单而完整的神经网络工程。通过本实验理解神经网络中基本模块的作用和模块间的关系，为后续建立更复杂的神经网络实验奠定基础。

2) 利用Python实现神经网络基本单元的前向传播（正向传播）和反向传播，加深对神经网络中基本单元的理解，包括全连接层、激活函数、损失函数等基本单元。

3) 利用Python实现神经网络的构建和训练，实现神经网络所使用的梯度下降算法，加深对神经网络训练过程的理解。

2、背景知识

2.1 神经网络的组成

一个完整的神经网络通常由多个基本的网络层堆叠而成。本实验中的三层全连接神经网络由三个全连接层构成，在每两个全连接层之间插入 ReLU 激活函数以引入非线性变换, 最后使用 Softmax 层计算交叉熵损失，如图 2.1 所示。因此本实验中使用的基本单元包括全连接层、ReLU 激活函数、Softmax 损失函数。

1. 全连接层

全连接层以一维向量作为输入，输入与权重相乘后再与偏置相加得到输出向量。假设全连接层的输入为 m 维列向量 x ，输出为 n 维列向量 y 。

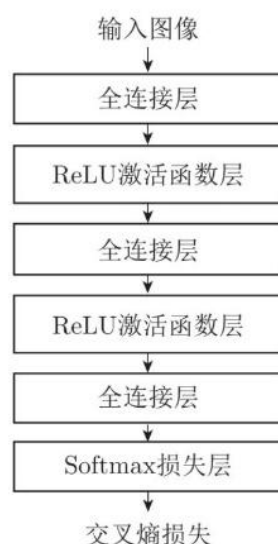


图 2.1 用于手写数字分类的三层全连接神经网络

全连接层的权重 W 是二维矩阵，维度为 $m \times n$, 偏置 b 是 n 维列向量。前向传播时，全连接层的输出的计算公式为(注意偏置可以是向量，计算每一个输出使用不同的值；偏置也可以是一个标量，计算同一层的输出使用同一个值)

$$\mathbf{y} = \mathbf{W}^T \mathbf{x} + \mathbf{b} \quad (2.1)$$

在计算全连接层的反向传播时，给定神经网络损失函数 L 对当前全连接层的输出 \mathbf{y} 的偏导 $\nabla_{\mathbf{y}} L = \frac{\partial L}{\partial \mathbf{y}}$ ，其维度与全连接层的输出 \mathbf{y} 相同，均为 n 。根据链式法则，全连接层的权重和偏置的梯度 $\nabla_{\mathbf{W}} L = \frac{\partial L}{\partial \mathbf{W}}$ 、 $\nabla_{\mathbf{b}} L = \frac{\partial L}{\partial \mathbf{b}}$ 以及损失函数对输入的偏导 $\nabla_{\mathbf{x}} L = \frac{\partial L}{\partial \mathbf{x}}$ 计算公式为：

$$\begin{aligned} \nabla_{\mathbf{W}} L &= \mathbf{x} (\nabla_{\mathbf{y}} L)^T \\ \nabla_{\mathbf{b}} L &= \nabla_{\mathbf{y}} L \\ \nabla_{\mathbf{x}} L &= \mathbf{W} \nabla_{\mathbf{y}} L \end{aligned} \quad (2.2)$$

实际应用中通常使用批量 (batch) 随机梯度下降算法，即选择若干个样本同时计算。假设选择的样本量为 p ，此时输入变为二维矩阵 \mathbf{X} ，维度为 $p \times m$ ，每行代表一个样本，每个样本对应公式 (2.1) 中的 \mathbf{x}^T 。输出也变为二维矩阵 \mathbf{Y} ，维度为 $p \times n$ 。此时全连接层的前向传播计算公式由公式 (2.1) 变为（需要说明的是，批量处理时输入 \mathbf{X} 和输出 \mathbf{Y} 的每一行代表一个样本，分别对应公式 (2.1) 中的 \mathbf{x} 和 \mathbf{y} 的转置，因此公式 (2.3) 和公式 (2.4) 的形式与公式 (2.1) 和 (2.2) 略有不同）

$$\mathbf{Y} = \mathbf{XW} + \mathbf{b}^T \quad (2.3)$$

其中的 $+$ 代表广播运算，表示偏置 \mathbf{b} 中的元素会被加到 \mathbf{XW} 的乘积矩阵的对应的一行元素中。权重和偏置的梯度 $\nabla_{\mathbf{W}} L$ 、 $\nabla_{\mathbf{b}} L$ 以及损失函数对输入的偏导 $\nabla_{\mathbf{x}} L$ 的计算公式由公式 (2.2) 变为

$$\begin{aligned} \nabla_{\mathbf{W}} L &= \mathbf{X}^T \nabla_{\mathbf{Y}} L \\ \nabla_{\mathbf{b}} L &= (\nabla_{\mathbf{Y}} L)^T \mathbf{1} \\ \nabla_{\mathbf{x}} L &= \nabla_{\mathbf{Y}} L \mathbf{W}^T \end{aligned} \quad (2.4)$$

其中计算偏置的梯度 $\nabla_{\mathbf{b}} L$ 时，为确保维度正确，用 $\nabla_{\mathbf{Y}} L$ 与维度为 $p \times 1$ 的全 1 向量 $\mathbf{1}$ 点积（也可以理解为在 $\text{axis}=0$ 轴上求和）。

2. ReLU 激活函数

ReLU 激活函数是按元素运算操作，输出向量 \mathbf{y} 的维度与输入向量 \mathbf{x} 的维度相同。在前向传播中，如果输入 \mathbf{x} 中的元素小于 0，输出为 0，否则输出等于输入。因此 ReLU 的计算公式为：

$$\mathbf{y}(i) = \max(0, \mathbf{x}(i)) \quad (2.5)$$

其中 $\mathbf{x}(i)$ 和 $\mathbf{y}(i)$ 分别代表 \mathbf{x} 和 \mathbf{y} 在位置 i 的值。

由于 ReLU 激活函数不包含参数，在反向传播计算过程中仅需根据损失函数对输出的偏导 $\nabla_{\mathbf{y}} L$ 计算损失函数对输入的偏导 $\nabla_{\mathbf{x}} L$ 。设 i 代表输入 \mathbf{x} 的某个位置，则损失函数对本层的第 i 个输入的偏导 $\nabla_{\mathbf{x}(i)} L$ 的计算公式为

$$\nabla_{\mathbf{x}(i)} L = \begin{cases} \nabla_{\mathbf{y}(i)} L, & \mathbf{x}(i) \geq 0 \\ 0, & \mathbf{x}(i) < 0 \end{cases} \quad (2.6)$$

3. Softmax 损失层

Softmax 损失层是目前多分类问题中最常用的损失函数，假设 Softmax 损失层的输入为向量 \mathbf{x} ，维度为 k 。其中 k 对应分类的类别数，例如对手写数字进行分类时， $k = 10$ 。在前向传播的计算过程中，首先对 \mathbf{x} 计算 e 指数并进行行归一化，从而得到 Softmax 分类概率。假设 \mathbf{x} 对应 i 位置的值为 $\mathbf{x}(i)$ ， $\hat{\mathbf{y}}(i)$ 为 i 位置的 Softmax 分类概率， $i \in [1, k]$ 且为整数，则 $\hat{\mathbf{y}}(i)$ 的计算公式为：

$$\hat{y}(i) = \frac{e^{x(i)}}{\sum_j e^{x(j)}} \quad (2.7)$$

在前向计算时，对 Softmax 分类概率 \hat{y} 取最大概率对应的类别作为预测的分类类别。损失函数层在计算前向传播时还需要根据给定的标记 (label, 也称为真实值或实际值) y 计算总的损失函数值。在分类任务中，标记 y 通常表示为一个维度为 k 的 one-hot 向量，该向量中对应真实类别的分量值为 1, 其他值为 0。Softmax 损失层使用了交叉熵计算损失值，其损失值 L 的计算公式为

$$L = - \sum_i y(i) \ln \hat{y}(i) \quad (2.8)$$

在反向传播的计算过程中，对于 Softmax 损失函数层，损失函数对输入的偏导 $\nabla_x L$ 的计算公式为

$$\nabla_x L = \frac{\partial L}{\partial x} = \hat{y} - y \quad (2.9)$$

由于工程实现中使用批量随机梯度下降算法，假设选择的样本量为 p ，Softmax 损失层的输入变为二维矩阵 X ，维度为 $p \times k$ ， X 的每个行向量代表一个样本。则对每个样本的激活值计算 e 指数并进行行归一化得到

$$\hat{Y}(i, j) = \frac{e^{X(i, j)}}{\sum_j e^{X(i, j)}} \quad (2.10)$$

其中 $X(i, j)$ 代表 X 中对应第 i 个样本在位置 j 的值。当 $X(i, j)$ 数值较大时，求 e 指数可能会出现数值上溢的问题。因此在实际工程实现时，为确保数值稳定性，会在求 e 指数前先进行减最大值处理，此时 $\hat{Y}(i, j)$ 的计算公式变为

$$\hat{Y}(i, j) = \frac{e^{X(i, j) - \max_n X(i, n)}}{\sum_j e^{X(i, j) - \max_n X(i, n)}} \quad (2.11)$$

在前向计算时，对 Softmax 分类概率 $\hat{Y}(i, j)$ 的每个样本 (即每个行向量) 取最大概率对应的类别作为预测的分类类别。此时标记 Y 通常表示为一组 one-hot 向量，维度为 $p \times k$ ，其中每行是一个 one-hot 向量，对应一个样本的标记。则计算损失值的公式(2.8)变为

$$L = - \frac{1}{p} \sum_{i, j} Y(i, j) \ln \hat{Y}(i, j) \quad (2.12)$$

其中损失值是所有样本的平均损失，因此对样本数量 p 取平均。

在反向传播时，当选择的样本量为 p 时，损失函数对输入的偏导 $\nabla_x L$ 的计算公式(2.9)变为

$$\nabla_x L = \frac{1}{p} (\hat{Y} - Y) \quad (2.13)$$

类似地，损失 $\nabla_x L$ 是所有样本的平均损失，因此对样本数量 p 取平均。

2.2 神经网络训练

神经网络训练通过调整网络层的参数来使神经网络计算出来的结果与真实结果 (标记) 尽量接近。神经网络训练通常使用随机梯度下降算法，通过不断的迭代计算每层参数的梯度，利用梯度对每层参数进行更新。具体而言，给定当前迭代的训练样本 (包含输入数据及标记信息)，首先进行神经网络的前向传播处理，输入数据和权重相乘再经过激活函数计算出隐层，隐层与下一层的权重相乘再经过激活函数得到下一个隐层，通过逐层迭代计算出神经网络的输出结果。随后利用输出结果和标记信息计算出损失函数值。然后进行神经网络的反向传播处理，从损失函数开始逆序逐层计算损失函数对权重和偏置的偏导 (即梯度)，最后利用梯度对相应的参数进行更新。更新参数 W 的计算公式为

$$W = W - \eta \nabla_W L \quad (2.14)$$

其中， $\nabla_w L$ 为参数的梯度， η 是学习率。

下面以图 2.2 中的两层神经网络为例，介绍神经网络训练的具体过程。图 2.2 中的网络由两个全连接层及 Softmax 层组成，其中第一个全连接层的权重为 $W^{(1)}$ ，偏置为 $b^{(1)}$ ，第二个全连接层的权重为 $W^{(2)}$ ，偏置为 $b^{(2)}$ 。假设某次迭代的网络输入为 x ，对应的标记为 y ，该神经网络前向传播的逐层计算公式依次是

$$\begin{aligned} h &= W^{(1)T} x + b^{(1)} \\ z &= W^{(2)T} h + b^{(2)} \end{aligned} \quad (2.15)$$

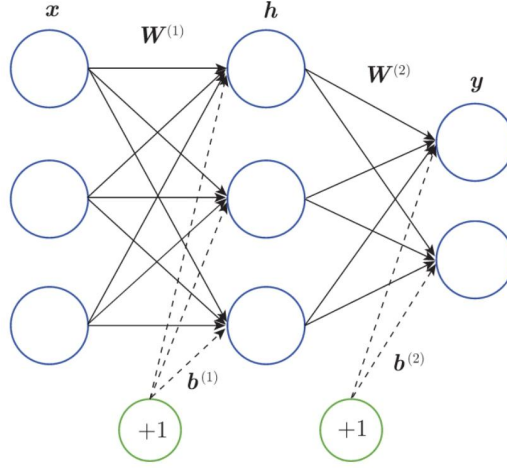


图 2.2 两层神经网络示例

其中 h 、 z 分别是第一、第二层全连接层的输出。Softmax 损失的损失值 L 为：

$$\begin{aligned} \hat{y}(i) &= \frac{e^{z(i)}}{\sum_i e^{z(i)}} \\ L &= -\sum_i y(i) \ln \hat{y}(i) \end{aligned} \quad (2.16)$$

反向传播的逐层计算公式为

$$\begin{aligned} \nabla_z L &= \hat{y} - y \\ \nabla_{W^{(2)}} L &= h(\nabla_z L)^T \\ \nabla_{b^{(2)}} L &= \nabla_z L \\ \nabla_h L &= W^{(2)} \nabla_z L \\ \nabla_{W^{(1)}} L &= x(\nabla_h L)^T \\ \nabla_{b^{(1)}} L &= \nabla_h L \\ \nabla_x L &= W^{(1)} \nabla_h L \end{aligned} \quad (2.17)$$

其中 $\nabla_z L$ 、 $\nabla_h L$ 、 $\nabla_x L$ 分别是损失函数对 Softmax 层、第二层、第一层的偏导， $\nabla_{W^{(2)}} L$ 、 $\nabla_{b^{(2)}} L$ 、 $\nabla_{W^{(1)}} L$ 、 $\nabla_{b^{(1)}} L$ 分别是第二层和第一层的权重和偏置梯度， η 为学习率。更新两个全连接层的权重和偏置的计算为

$$\begin{aligned} W^{(1)} &= W^{(1)} - \eta \nabla_{W^{(1)}} L \\ b^{(1)} &= b^{(1)} - \eta \nabla_{b^{(1)}} L \end{aligned}$$

$$\begin{aligned}W^{(2)} &= W^{(2)} - \eta \nabla_{W^{(2)}} L \\b^{(2)} &= b^{(2)} - \eta \nabla_{b^{(2)}} L\end{aligned}\tag{2.18}$$

3、示例代码

1) 数据集

数据集采用 MNIST 手写数字库 (老师直接提供, 也可在 <http://yann.lecun.com/exdb/mnist/> 自行下载)。该数据集包含一个训练集和一个测试集, 其中训练集有 60000 个样本, 测试集有 10000 个样本。每个样本都由灰度图像 (即单通道图像) 及其标记组成, 图像大小为 28×28 。MNIST 数据集包含 4 个文件, 分别是训练集图像、训练集标记、测试集图像、测试集标记。

2) 总体设计

设计一个三层神经网络实现手写数字图像分类。该网络包含两个隐层和一个输出层, 其中输入神经元个数由输入数据维度决定, 输出层的神经元个数由数据集包含的类别决定, 两个隐层的神经元个数可以作为超参数自行设置。对于手写数字图像的分类问题, 输入数据为手写数字图像, 原始图像一般可表示为二维矩阵 (灰度图像) 或三维矩阵 (彩色图像), 在输入神经网络前会将图像矩阵调整为一维向量作为输入。待分类的类别数一般是提前预设的, 如手写数字包含 0 至 9 共 10 个类别, 则神经网络的输出神经元个数为 10。

为了便于迭代开发, 工程实现时采用模块化的方式来实现整个神经网络的处理, 共划分为5大模块:

1) 数据加载模块: 从文件中读取数据, 并进行预处理, 其中预处理包括归一化、维度变换等处理。如果需要人为对数据进行随机数据扩增, 则数据扩增处理也在数据加载模块中实现。

2) 基本单元模块: 实现神经网络中不同类型的网络层的定义、前向传播、反向传播等功能。

3) 网络结构模块: 利用基本单元模块建一个完整的神经网络。

4) 网络训练 (training) 模块: 用训练集对神经网络进行训练。对建立的神经网络结构, 实现神经网络的前向传播、神经网络的反向传播、对神经网络进行参数更新、保存神经网络参数等基本操作, 以及训练函数主体。

5) 网络推断 (inference) 模块: 使用训练得到的网络模型, 对测试样本进行预测 (也称为测试或推断)。具体操作包括加载训练得到的模型参数、神经网络的前向传播等。

3) 数据加载模块

本实验采用的数据集是 MNIST 手写数字库。该数据集中的图像数据和标记数据采用表 2.1 中的 IDX 文件格式存放。图像的像素值按行优先顺序存放, 取值范围为 $[0, 255]$, 其中 0 表示黑色, 255 表示白色。

表 2.1 MNIST 数据集 IDX 文件格式

图像文件格式			
字节偏移	数据类型	值	描述
0000	int32 (32 位有符号整型)	0x00000803(2051)	magic number (魔数): 表示像素的数据类型以及像素数据的维度信息, MSB (大尾端)
0004	int32	60000 (训练集) 10000 (测试集)	图像数量
0008	int32	28	图像行数, 即图像高度
0012	int32	28	图像列数, 即图像宽度
0016	uint8 (8 位无符号整型)	??	像素值
0017	uint8	??	像素值
.....			
XXXX	uint8	??	像素值

标记文件格式			
字节偏移	数据类型	值	描述
0000	int32	0x00000801(2049)	magic number
0004	int32	60000 (训练集) 10000 (测试集)	标记数量
0008	uint8	??	标记
0009	uint8	??	标记
.....			
XXXX		??	标记

首先编写读取 MNIST 数据集文件并预处理的子函数, 程序示例如图 2.3 所示。然后调用该子函数对 MNIST 数据集中的 4 个文件分别进行读取和预处理, 并将处理过的训练和测试数据存储在 NumPy 矩阵中 (训练模型时可以快速读取该矩阵中的数据), 实现该功能的程序示例如图 2.4 所示。

```

1. def load_mnist(self, file_dir, is_images = 'True'):
2.     bin_file = open(file_dir, 'rb')
3.     bin_data = bin_file.read()
4.     bin_file.close()
5.     # 分析文件头部
6.     if is_images: # 读取图像数据
7.         fmt_header = '>iiii'
8.         magic, num_images, num_rows, num_cols = struct.unpack_from(fmt_header, bin_data, 0)
9.     else: # 读取标记数据
10.        fmt_header = '>ii'
11.        magic, num_images = struct.unpack_from(fmt_header, bin_data, 0)
12.        num_rows, num_cols = 1, 1
13.        data_size = num_images * num_rows * num_cols
14.        mat_data = struct.unpack_from('>' + str(data_size) + 'B', bin_data, struct.calcsize(fmt_header))
15.        mat_data = np.reshape(mat_data, [num_images, num_rows * num_cols])
16.        print('Load images from %s, number: %d, data shape: %s' % (file_dir, num_images, str(mat_data.shape)))
17.        return mat_data

```

图 2.3 MNIST 数据集文件的读取和预处理

```

1. def load_data(self):
2.     # TODO: 调用函数 load_mnist 读取和预处理 MNIST 中训练数据和测试数据的图像和标记

```

```

3.     print('Loading MNIST data from files...')
4.     train_images = self.load_mnist(os.path.join(MNIST_DIR, TRAIN_DATA), True)
5.     train_labels = _____
6.     test_images = _____
7.     test_labels = _____
8.     self.train_data = np.append(train_images, train_labels, axis=1)
9.     self.test_data = np.append(test_images, test_labels, axis=1)

```

图 2.4 MNIST 子数据集的读取和预处理

TODO 提示：代码中已有如下定义，直接按照 train_images 的代码套用即可：

```

TRAIN_DATA = "train-images-idx3-ubyte"
TRAIN_LABEL = "train-labels-idx1-ubyte"
TEST_DATA = "t10k-images-idx3-ubyte"
TEST_LABEL = "t10k-labels-idx1-ubyte"

```

4) 基本单元模块

本实验采用图 2.1 中的三层神经网络，主体是三个全连接层。在前两个全连接层之后使用 ReLU 激活函数层引入非线性变换，本实验采用 ReLU 层作为激活函数层。在神经网络的最后添加 Softmax 层计算交叉熵损失。因此，本实验中需要实现的基本单元模块包括全连接层、ReLU 层和 Softmax 损失层。

在神经网络实现中，通常同类型的层用一个类来定义，多个同类型的层用类的实例来实现，层中的计算用类的成员函数来定义。类的成员函数通常包括层的初始化、参数的初始化、前向传播计算、反向传播计算、参数的更新、参数的加载和保存等。其中层的初始化函数一般会根据实例化层时的输入系数确定该层的超参数，例如该层的输入神经元数量和输出神经元数量等。参数的初始化函数会对该层的参数（如全连接层中的权重和偏置）分配存储空间，并填充初始值。前向传播函数利用前一层的输出作为本层的输入，计算本层的输出结果、反向传播函数根据链式法则逆序逐层计算损失函数对权重和偏置的梯度。参数的更新函数利用反向传播函数计算的梯度对本层的参数进行更新。参数的加载函数从给定的文件中加载参数的值，参数的保存函数将当前层参数的值保持到指定的文件中。有些层（如激活函数层）可能没有参数，就不需要定义参数的初始化、更新、加载和保存函数。有些层（如激活函数层和损失函数层）的输出维度由输入维度决定，不需要人工设定，因此不需要层的初始化函数。

以下便是全连接层、ReLU 层和 Softmax 损失层的具体实现步骤。

1) 全连接层：程序示例如图 2.5 所示，定义了以下成员函数。

- 层的初始化：需要确定该全连接层的输入神经元个数（即输入二维矩阵中每个行向量的维度）和输出神经元个数（即输出二维矩阵中每个行向量的维度）。
- 参数初始化：全连接层的参数包括权重和偏置。根据输入向量的维度 m 和输出向量的维度 n 可以确定权重 W 的维度为 $m \times n$ ，偏置 b 的维度为 n 。在对权重和偏置进行初始化时，通常利用高斯随机数初始化权重的值，而将偏置的所有值初始化为 0。
- 前向传播计算：全连接层的前向传播计算公式为(2.3),可以通过输入矩阵与权重矩阵相乘再与偏置相加实现。
- 反向传播计算：全连接层的反向传播的计算公式为(2.4)。给定损失函数对本层输出的偏导 $\nabla_y L$ ，利用矩阵相乘计算权重和偏置的梯度 $\nabla_w L$ 、 $\nabla_b L$ 以及损失函数对本层输入的偏导 $\nabla_x L$ 。

- 参数更新：给定学习率 η ，可利用反向传播计算得到的权重梯度 $\nabla_w L$ 和偏置梯度 $\nabla_b L$ 对本层的权重 W 和偏置 b 进行更新：

$$W = W - \eta \nabla_w L$$

$$b = b - \eta \nabla_b L$$

- 参数加载：从该函数的输入中读取本层的权重 W 和偏置 b 。
- 参数保存：返回本层当前的权重 W 和偏置 b 。

```

1. class FullyConnectedLayer(object):
2.     def __init__(self, num_input, num_output): # 全连接层初始化
3.         self.num_input = num_input
4.         self.num_output = num_output
5.         print('\tFully connected layer with input %d, output %d.' % (self.num_input, self.
num_output))
6.     def init_param(self, std=0.01): # 参数初始化
7.         self.weight = np.random.normal(loc=0.0, scale=std, size=(self.num_input, self.num_
output))
8.         self.bias = np.zeros([1, self.num_output])
9.     def forward(self, input): # 前向传播计算
10.        start_time = time.time()
11.        self.input = input
12.        # TODO: 全连接层的前向传播, 计算输出结果 提示: 公式 2.1, 矩阵乘法 np.matmul
13.        self.output = _____
14.        return self.output
15.     def backward(self, top_diff): # 反向传播的计算 大水漫灌, top_diff 就是上一层房间的海水
16.        # TODO: 全连接层的反向传播, 计算参数梯度和本层损失
17.        self.d_weight = _____ # 提示: 公式 2.4
18.        self.d_bias = _____
19.        bottom_diff = _____
20.        return bottom_diff # 大水漫灌, bottom_diff 就是传到下一层房间的海水
21.     def update_param(self, lr): # 参数更新
22.        # TODO: 对全连接层参数利用参数进行更新 提示: 公式 2.14
23.        self.weight = _____
24.        self.bias = _____
25.     def load_param(self, weight, bias): # 参数加载
26.        assert self.weight.shape == weight.shape
27.        assert self.bias.shape == bias.shape
28.        self.weight = weight
29.        self.bias = bias
30.     def save_param(self): # 参数保存
31.        return self.weight, self.bias

```

图 2.5 全连接层的实现示例

2) ReLU 层：不包含参数，因此实现中没有参数初始化、参数更新、参数的加载和保存相关的函数。ReLU 层的程序示例如图 2.6 所示，定义了以下成员函数。

- 前向传播计算：根据公式(2.5)可以计算 ReLU 层前向传播的结果。在工程实现中，可以对整个输入矩阵使用 `maximum` 函数，`maximum` 函数会进行广播，计算输入矩阵的每个元素与 0 的最大值。
- 反向传播计算：根据公式(2.6)可以计算损失函数对输入的偏导。在工程实现中，可以获取 $x(i)<0$ 的位置索引，将 y 中对应位置的值为 0。

```

1. class ReLULayer(object):
2.     def __init__(self):
3.         print('\tReLU layer.')

```

```

4.     def forward(self, input): # 前向传播的计算
5.         start_time = time.time()
6.         self.input = input
7.         # TODO: ReLU 层的前向传播, 计算输出结果 提示: 公式 2.5
8.         output = _____
9.         return output
10.    def backward(self, top_diff): # 反向传播的计算
11.        # TODO: ReLU 层的反向传播, 计算本层损失 提示: 公式 2.6
12.        bottom_diff = _____
13.        bottom_diff[self.input<0] = 0
14.        return bottom_diff

```

图 2.6 ReLU 层的实现示例

3) Softmax 损失层: 同样不包含参数, 因此实现中没有参数初始化、更新、加载和保存相关的函数。但该层需要额外计算总的损失函数值, 作为训练时的中间输出结果, 帮助判断模型的训练进程。Softmax 损失层的程序示例如图 2.7 所示, 定义了以下成员函数。

- 前向传播计算: 使用公式(2.11)计算, 该公式为确保数值稳定, 会在求 e 指数前先进行减最大值处理。
- 损失函数计算: 使用公式(2.12)计算, 采用批量随机梯度下降法训练时损失值是 batch 内的所有样本的损失值的均值。需要注意的是, MNIST 手写数字库的标记数据读入的是 0 至 9 的类别编号, 计算损失时需要先将类别编号转换为 one-hot 向量。
- 反向传播计算: 可以使用公式(2.13)计算, 计算时同样需要对样本数量取平均。

```

1. class SoftmaxLossLayer(object):
2.     def __init__(self):
3.         print('\tSoftmax loss layer.')
4.     def forward(self, input): # 前向传播的计算
5.         # TODO: softmax 损失层的前向传播, 计算输出结果 提示: 公式 2.7
6.         input_max = np.max(input, axis=1, keepdims=True)
7.         input_exp = np.exp(input - input_max)
8.         self.prob = _____
9.         return self.prob
10.    def get_loss(self, label): # 计算损失
11.        self.batch_size = self.prob.shape[0]
12.        self.label_onehot = np.zeros_like(self.prob)
13.        self.label_onehot[np.arange(self.batch_size), label] = 1.0
14.        loss = -np.sum(np.log(self.prob) * self.label_onehot) / self.batch_size
15.        return loss
16.    def backward(self): # 反向传播的计算
17.        # TODO: softmax 损失层的反向传播, 计算本层损失 提示: 公式 2.13
18.        bottom_diff = _____
19.        return bottom_diff

```

图 2.7 Softmax 损失层的实现示例

5) 网络结构模块

网络结构模块利用已经实现的神经网络的基本单元来建立一个完整的神经网络。在工程实现中通常用一个类来定义一个神经网络, 用类的成员函数来定义神经网络的初始化、建立神经网络结构、对神经网络进行参数初始化等基本操作。本实验中三层神经网络的网络结构模块的程序示例如图2.8所示, 定义了以下成员函数。

- 神经网络初始化: 确定神经网络相关的超参数, 例如网络中每个隐层的神经元个数。

•建立网络结构：定义整个神经网络的拓扑结构，实例化基本单元模块中定义的层并将这些层进行堆叠。例如本实验使用的三层神经网络包含三个全连接层，并且在前两个全连接层后跟随有ReLU层，神经网络的最后使用了Softmax损失层。

•神经网络参数初始化：对于神经网络中包含参数的层，依次调用这些层的参数初始化函数，从而完成整个神经网络的参数初始化。本实验使用的三层神经网络中，只有三个全连接层包含参数，依次调用其参数初始化函数即可。

```
1. class MNIST_MLP(object):
2.     def __init__(self, batch_size=100, input_size=784, hidden1=32, hidden2=16, out_classes
   =10, lr=0.01, max_epoch=2, print_iter=100):
3.         # 神经网络初始化
4.         self.batch_size = batch_size
5.         self.input_size = input_size
6.         self.hidden1 = hidden1
7.         self.hidden2 = hidden2
8.         self.out_classes = out_classes
9.         self.lr = lr
10.        self.max_epoch = max_epoch
11.        self.print_iter = print_iter
12.    def build_model(self): # 建立网络结构
13.        # TODO: 建立三层神经网络结构
14.        print('Building multi-layer perception model...')
15.        self.fc1 = FullyConnectedLayer(self.input_size, self.hidden1)
16.        self.relu1 = ReLULayer()
17.
18.        self.fc3 = FullyConnectedLayer(self.hidden2, self.out_classes)
19.        self.softmax = SoftmaxLossLayer()
20.        self.update_layer_list = [self.fc1, self.fc2, self.fc3]
21.
22.    def init_model(self):
23.        print('Initializing parameters of each layer in MLP...')
24.        for layer in self.update_layer_list:
25.            layer.init_param()
```

图 2.8 三层神经网络的网络结构模块实现示例

6) 网络训练（training）模块

神经网络训练流程如图2.9所示。在完成数据加载模块和网络结构模块实现之后，需要实现训练模块。本实验中三层神经网络的网络训练模块程序示例如图2.10所示。神经网络的训练模块通常拆解为若干步骤，包括神经网络的前向传播、神经网络的反向传播、神经网络参数更新、神经网络参数保存等基本操作。这些网络训练模块的基本操作以及训练主体用神经网络类的成员函数来定义：

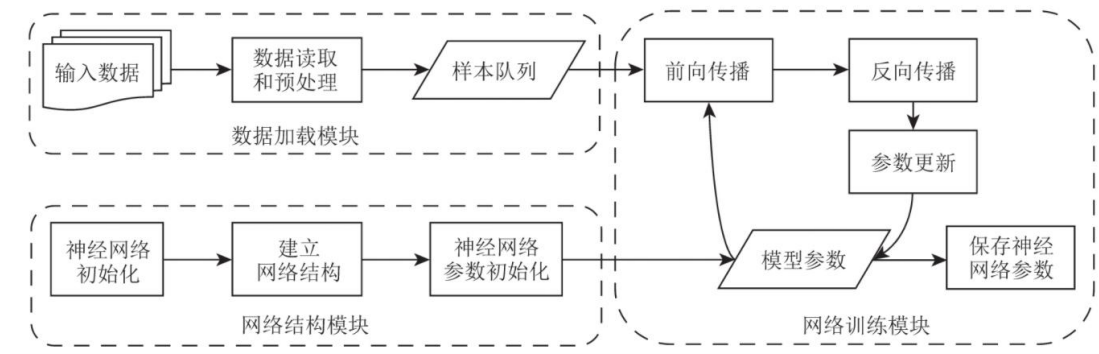


图2.9 神经网络训练流程

神经网络的前向传播：根据神经网络的拓扑顺序，顺序调用每层的前向传播函数。以输入数据作为第一层的输入，之后每层的输出作为其下一层的输入顺序计算每一层的输出，最后得到损失函数层的输出。

- 神经网络的反向传播：根据神经网络的拓扑顺序，逆序调用每层的反向传播函数。采用链式法则逆序逐层计算损失函数对每层参数的偏导，最后得到神经网络所有层的参数梯度。

- 神经网络参数更新：对神经网络中包含参数的层，依次调用各层的参数更新函数，来对整个神经网络的参数进行更新。本实验中的三层神经网络仅其中的三个全连接层包含参数，依次更新三个全连接层的参数即可。

- 神经网络参数保存：对神经网络中包含参数的层，依次收集这些层的参数并存储到文件中。

- 神经网络训练主体：在该函数中，（1）确定训练的一些超参数，如使用批量梯度下降算法时的批量大小、学习率大小、迭代次数（或训练周期次数）、可视化训练过程时每迭代多少次屏幕输出一次当前的损失值等等。（2）开始迭代训练过程。每次迭代训练开始前,可以根据需要对数据进行随机打乱，一般是一个训练周期（即当整个数据集的数据都参与一次训练过程）后对数据集进行随机打乱。每次迭代训练过程中，先选取当前迭代所使用的数据和对应的标记，再进行整个网络的前向传播，随后计算当前迭代的损失值，然后进行整个网络的反向传播来获得整个网络的参数梯度，最后对整个网络的参数进行更新。完成一次迭代后可以根据需要在屏幕上输出当前的损失值,以供实际应用中修改模型作参考。完成神经网络的训练过程后，通常会将训练得到的神经网络模型参数保存到文件中。

```

1. def forward(self, input): # 神经网络的前向传播
2.     # TODO: 神经网络的前向传播
3.     h1 = self.fc1.forward(input)
4.     h1 = self.relu1.forward(h1)
5.
6.     prob = self.softmax.forward(h3)
7.     return prob
8.
9. def backward(self): # 神经网络的反向传播
10.    # TODO: 神经网络的反向传播
11.    dloss = self.softmax.backward()
12.

```

```

13.     dh1 = self.relu1.backward(dh2)
14.     dh1 = self.fc1.backward(dh1)
15.
16. def update(self, lr): # 神经网络的参数更新
17.     for layer in self.update_layer_list:
18.         layer.update_param(lr)
19.
20. def train(self): # 训练函数
21.     max_batch = self.train_data.shape[0] // self.batch_size
22.     print('Start training...')
23.     for idx_epoch in range(self.max_epoch):
24.         self.shuffle_data()
25.         for idx_batch in range(max_batch):
26.             batch_images = self.train_data[idx_batch*self.batch_size:(idx_batch+1)*self.ba
27. tch_size, :-1]
28.             batch_labels = self.train_data[idx_batch*self.batch_size:(idx_batch+1)*self.ba
29. tch_size, -1]
30.             prob = self.forward(batch_images)
31.             loss = self.softmax.get_loss(batch_labels)
32.             self.backward()
33.             self.update(self.lr)
34.             if idx_batch % self.print_iter == 0:
35.                 print('Epoch %d, iter %d, loss: %.6f' % (idx_epoch, idx_batch, loss))

```

图 2.10 三层神经网络的网络训练模块实现示例

7) 网络推断（inference）模块

整个神经网络推断流程如图2.11所示。完成神经网络的训练之后，可以用训练得到的模型对测试数据进行预测，以评估模型的精度。本实验中三层神经网络的网络推断模块程序示例如图2.12所示。工程实现中同样常将一个神经网络的推断模块拆解为若干步骤，包括神经网络模型参数加载、前向传播、精度计算等基本操作。这些网络推断模块的基本操作以及推断主体用神经网络类的成员函数来定义：

- 神经网络的前向传播：网络推断模块中的神经网络前向传播操作与网络训练模块中的前向传播操作完全一致，因此可以直接调用网络训练模块中的神经网络前向传播函数。

- 神经网络参数加载：读取神经网络训练模块保存的模型参数文件，并加载有参数的网络层的参数值。

- 神经网络推断函数主体：在进行神经网络推断前，需要从模型参数文件中加载神经网络的参数。在神经网络推断过程中，循环每次读取一定批量的测试数据，随后进行整个神经网络的前向传播计算得到神经网络的输出结果。得到整个测试数据集的输出结果后，与测试数据集的标记进行对比，利用相关的评价函数计算模型的精度，如手写数字分类问题使用分类平均正确率作为模型的评函数。

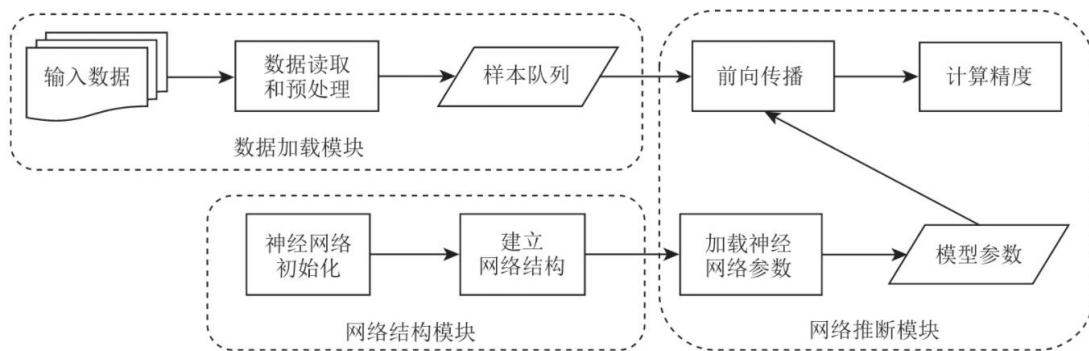


图 2.11 神经网络推断流程

```

1. def load_model(self, param_dir): # 加载神经网络权值
2.     print('Loading parameters from file ' + param_dir)
3.     params = np.load(param_dir, allow_pickle=True).item()
4.     self.fc1.load_param(params['w1'], params['b1'])
5.     self.fc2.load_param(params['w2'], params['b2'])
6.     self.fc3.load_param(params['w3'], params['b3'])
7.
8. def evaluate(self): # 推断函数
9.     pred_results = np.zeros([self.test_data.shape[0]])
10.    start_time = time.time()
11.    for idx in range(self.test_data.shape[0]//self.batch_size):
12.        batch_images = self.test_data[idx*self.batch_size:(idx+1)*self.batch_size, :-1]
13.        prob = self.forward(batch_images)
14.        end = time.time()
15.        pred_labels = np.argmax(prob, axis=1)
16.        pred_results[idx*self.batch_size:(idx+1)*self.batch_size] = pred_labels
17.    print("All evaluate time: %f"%(time.time()-start_time))
18.    accuracy = np.mean(pred_results == self.test_data[:, -1])
19.    print('Accuracy in test set: %f' % accuracy)

```

图 2.12 三层神经网络的网络推断模块实现示例

8) 完整实验流程

完成神经网络的各个模块之后，调用这些模块就可以实现用三层神经网络进行手写数字图像分类的完整流程。本实验中三层神经网络的完整流程的程序示例如图2.13所示。首先实例化三层神经网络对应的类，指定神经网络的超参数，如每层的神经元个数。其次进行数据的加载和预处理。再调用网络结构模块建立神经网络，随后进行网络初始化，在该过程中网络结构模块会自动调用基本单元模块实例化神经网络中的每个层。然后调用网络训练模块训练整个网络，之后将训练得到的模型参数保存到文件中。最后从文件中读取训练得到的模型参数，之后调用网络推断模块测试网络的精度。

```

1. if __name__ == '__main__':
2.     h1, h2, e = 32, 16, 1
3.     mlp = MNIST_MLP(hidden1=h1, hidden2=h2, max_epoch=e)
4.     mlp.load_data()
5.     mlp.build_model()
6.     mlp.init_model()
7.     start_time = time.time()
8.     mlp.train()

```

```

9.     print("All train time: %f"%(time.time()-start_time))
10.    mlp.save_model('mlp-%d-%d-%epoch.npy' % (h1, h2, e))
11.    mlp.load_model('mlp-%d-%d-%epoch.npy' % (h1, h2, e))
12.    mlp.evaluate()

```

图 2.13 三层神经网络的完整流程实现示例

9) 实验评估

在图像分类任务中，通常使用测试集的平均分类正确率判断分类结果的精度。假设共有 N 个图像样本（MNIST手写数据集中共包含10000张测试图像，此时 $N=10000$ ）， $bm p_i$ 为神经网络输出的第 i 张图像的预测结果， p_i 为一个向量，取其中最大分量对应的类别作为预测类别。假设第 i 张图像的标记为 y_i ，即第 i 张图像属于类别 y_i ，则计算平均分类正确率 R 的公式为

$$R = \frac{1}{N} \sum_{i=1}^N 1(\operatorname{argmax}(p_i) = y_i)$$

其中 $1(\operatorname{argmax}(p_i) = y_i)$ 代表当 p_i 中的最大分量对应的类别编号与 y_i 相等时值为1，否则值为0。

4、实验内容

1) 请在代码中有TODO的地方填空，将程序补充完整，在报告中写出相应代码，并给出自己的理解。

2) `mlp.load_data()` 执行到最后时，`train_images`、`train_labels`、`test_images`、`test_labels` 的维度是多少？即多少行多少列，用 (x, y) 来表示。`self.train_data` 和 `self.test_data` 的维度是多少？

3) 本案例中的神经网络一共有几层？每层有多少个神经元？如果要增加或减少层数，应该怎么做（简单描述即可不用编程）？如果要增加或减少某一层的节点，应该怎么做（简单描述）？如果要把 **softmax** 换成 **sigmoid**，应该怎么做（简单描述）？

4) 在 `train()` 函数中，`max_batch = self.train_data.shape[0] // self.batch_size` 这一句的意义是什么？`self.shuffle_data()` 的意义是什么？

5) 最终 `evaluate()` 函数输出的 Accuracy in test set 是多少？请想办法提高该数值。本小题的评估标准设定如下：

- 60 分标准：给定全连接层、ReLU 层、Softmax 损失层的前向传播的输入矩阵、参数值、反向传播的输入，可以得到正确的前向传播的输出矩阵、反向传播的输出和参数梯度。
- 80 分标准：实现正确的三层神经网络，并进行训练和推断，使最后训练得到的模型在 MNIST 测试数据集上的平均分类正确率高于 92%。
- 90 分标准：实现正确的三层神经网络，并进行训练和推断，调整和训练相关的超参数，使最后训练得到的模型在 MNIST 测试数据集上的平均分类正确率高于 95%。
- 100 分标准：在三层神经网络基础上设计自己的神经网络结构，并进行训练和推断，使最后训练得到的模型在 MNIST 测试数据集上的平均分类正确率高于 98%。

实验四 基于传递闭包的模糊聚类

1、实验目的

掌握建立模糊等价矩阵的方法，会求传递闭包矩阵；掌握利用传递闭包进行模糊聚类的一般方法；会使用Python进行模糊矩阵的有关运算。

2、背景知识

“聚类”就是按照一定的要求和规律对事物进行区分和分类的过程，在这一过程中没有任何关于分类的先验知识，仅靠事物间的相似性作为类属划分的准则，属于无监督分类的范畴。传统的聚类分析是基于等价关系的一种硬划分，它把每个待辨识的对象根据等价关系严格地划分到某个类中，具有非此即彼的性质，因此这种分类类别界限是分明的。但是，现实的分类往往伴随着模糊性，即考虑的不是有无关系，而是关系的深浅程度，它们在形态和类属方面存在着中介性，适合进行软划分。人们用模糊的方法来处理聚类问题，并称之为模糊聚类分析。常用的模糊聚类方法有(1)基于模糊等价矩阵的聚类方法(传递闭包法、Bool 矩阵法);(2)基于模糊相似矩阵的直接聚类法(直接聚类法、最大树法、编网法);(3)基于目标函数的模糊聚类分析法。三种方法各有优缺点:直接聚类法不用计算模糊等价矩阵,计算量较小,可以比较直接地进行聚类。基于模糊等价矩阵的聚类方法较之编网法理论上更成熟,当矩阵阶数较高时,手工计算量较大,但在计算机上还是容易实现的。第三种聚类提出“聚类中心”的概念,这种方法可以得出某一对象在多大程度上属于某一类,但一般需要知道聚类数。

本实验采用基于传递闭包的模糊聚类方法，传递闭包法聚类首先需要通过标定的模糊相似矩阵 R ，然后求出包含矩阵 R 的最小模糊传递矩阵,即 R 的传递闭包 TR ,最后依据 TR 进行聚类。具体包括以下几个步骤:

1、得到特征指标矩阵

设论域 $U = \{x_1, x_2, \dots, x_n\}$ 为被分类对象，每个对象又由 m 个特征指标表示其性状：

$$x_i = \{x_{i1}, x_{i2}, \dots, x_{im}\}, i = 1, 2, \dots, n$$

则得到原始数据矩阵为 $X = (x_{ij})_{n \times m}$

2、采用最大值规格化法将数据规格化

在实际问题中，不同的数据一般有不同的量纲，为了使有不同量纲的数据可以相互比较，需要将原始数据 x_{ij} 压缩至 $[0,1]$ 区间，这一过程称为数据规格化。通常的方法有标准差规格化、极差规格化、最大值规格化等。

本案例采用最大值规格化，就是将 x_{ij} 换成 x'_{ij} ，即

$$x'_{ij} = \frac{x_{ij}}{M_j}, \text{ 其中 } M_j = \max(x_{1j}, x_{2j}, \dots, x_{nj})$$

由上式可知，对原始数据进行正规化处理以后，变量最大值为1，最小值为0，即新数据在区间 $[0,1]$ 内。

3、采用最大最小法构造得到模糊相似矩阵

为了建立模糊相似矩阵,引入相似系数 r_{ij} 表示两个样本 x_i 与 x_j 之间的相似程度,组成模糊相似矩阵 $R = (r_{ij})_{n \times n}$ 。建立方法有最大最小法、算术平均最小法、几何平均最小法等。最大最小法的具体公式

为:

$$r_{ij} = \frac{\sum_{k=1}^m x_{ik} \wedge x_{jk}}{\sum_{k=1}^m x_{ik} \vee x_{jk}}$$

算术平均最小法的具体公式为:

$$r_{ij} = \frac{\sum_{k=1}^m x_{ik} \wedge x_{jk}}{\frac{1}{2} \sum_{k=1}^m x_{ik} + x_{jk}}$$

4、采用平方法合成传递闭包

通过上述方法得到的模糊相似矩阵 R 一般只具有自反性和对称性, 不满足传递性。要对 R 进行变换, 使得 R 成为一个模糊等价关系, 即满足传递性。由于 R 的传递性不易验证, 因此, 进行聚类分析时, 需用 R 的传递闭包 $t(R)$ 代替 R 以此作为 R 的近似, 这种方法称为 Washall 算法, 又称传递闭包法。

模糊等价矩阵的定义: 设 $R = (r_{ij})_{n \times n}$ 是 n 阶模糊方阵, I 是 n 阶单位方阵, 若 R 满足

(1)自反性: $I \leq R \Leftrightarrow r_{ii} = 1$

(2)对称性: $R^T = R \Leftrightarrow r_{ij} = r_{ji}$

(3)传递性: $R^2 \leq R \Leftrightarrow \max\{(r_{ik} \wedge r_{kj}) | 1 \leq k \leq n\} \leq r_{ij}$

则称 R 为模糊等价矩阵。由模糊等价矩阵的传递性可知 $R^2 \leq R$,

而通过自反性可以推出 $R \leq R^2$, 从而得到: 对于模糊等价矩阵, 实际上有 $R^2 = R$ 。

关系的闭包的定义:

关系的闭包运算是关系上的一元运算, 它把给出的关系 R 扩充成一个新关系 R' , 使 R' 具有一定的性质, 且所进行的扩充又是最“节约”的。比如自反闭包, 相当于把关系 R 对角线上的元素全改成 1, 其他元素不变, 这样得到的 R' 是自反的, 且是改动次数最少的, 即是最“节约”的。

一个关系 R 的闭包, 是指加上最小数目的有序偶而形成的具有自反性, 对称性或传递性的新的有序偶集, 此集就是关系 R 的闭包。

设 R 是集合 A 上的二元关系, R 的自反(对称、传递)闭包是满足以下条件的关系 R' :

(i) R' 是自反的(对称的、传递的);

(ii) $R' \supseteq R$;

(iii) 对于 A 上的任何自反(对称、传递)关系 R'' , 若 $R'' \supseteq R$, 则有 $R'' \supseteq R'$ 。

R 的自反、对称、传递闭包分别记为 $r(R)$ 、 $s(R)$ 和 $t(R)$ 。

从模糊相似矩阵得到传递闭包, 采用的方法是平方法, 具体方法是: 把一个相似矩阵 R 进行自乘操作, 得到 R^2 并且检验 R^2 是否满足传递性, 如果 R^2 满足传递性则停止, 否则计算 R^4 并且检验 R^4 是否满足传递性, 如果 R^4 满足传递性则停止, 否则计算 $R^8 \dots$, 依此类推。这里的自乘操作是模糊集合下的算子, 即用模糊集合操作的“交”和“并”取代了原来矩阵操作中的“乘”和“加”操作。理论上已经证明, 该相乘操作只需要进行最多 $\log_2 n + 1$ 次, 就可以得到 $t(R)$ 。

5、计算截集, 得到模糊聚类结果

由于模糊相似矩阵 R 本身已经是自反的、对称的, 其传递闭包 $t(R)$ 又是传递的, 则 $t(R)$ 一定是模糊等价矩阵, 可以决定一个划分。

将 $t(R)$ 中的元素从大到小排列，逐一作为 α 求得对应的 α 水平截集。

α 水平截集是指隶属度大于等于 α 的元素组成的集合，可表示为 $R_\alpha = \{x, y | \mu_R(x, y) \geq \alpha\}$ ，其对应的特征函数为 $\mu_{R_\alpha}(x, y) = \begin{cases} 1 & \text{if } (x, y) \in R_\alpha \\ 0 & \text{if } (x, y) \notin R_\alpha \end{cases}$

注意：模糊集合的水平截集不再是模糊集合，而是一个确定集合。

按 α 由大到小进行聚类，得到模糊聚类结果。

数据源：本案例所用数据来自教材《计算智能导论》，原文如下：

在对教师的课堂教学质量进行评价时，考虑以下五项指标：师德师风、教学过程、教学方法、教学内容以及基本功。每项满分 20，总分 100。现在有 $\{x_1, x_2, x_3, x_4\}$ 四位老师进行参评，参评成绩如下矩阵，试对老师的成绩进行分类

$$X = \begin{bmatrix} 17 & 15 & 14 & 15 & 16 \\ 18 & 16 & 13 & 14 & 12 \\ 18 & 18 & 19 & 17 & 18 \\ 16 & 18 & 16 & 15 & 18 \end{bmatrix}$$

3、示例代码

```
1. import numpy as np
2.
3. np.set_printoptions(precision=2) # 设置矩阵输出精度,保留两位小数
4.
5. def MaxNormalization(a):
6.     """
7.     采用最大值规格化法将数据规格化为
8.     """
9.     c = np.zeros_like(a, dtype=float)
10.    for j in range(c.shape[1]): # 遍历 c 的列
11.        for i in range(c.shape[0]): # 遍历 c 的行
12.            c[i, j] = a[i, j]/np.max(a[:, j])
13.    return c
14.
15. def FuzzySimilarMatrix(a):
16.     """
17.     用最大最小法构造得到模糊相似矩阵
18.     """
19.     a = MaxNormalization(a) # 用标准化后的数据
20.     c = np.zeros((a.shape[0], a.shape[0]), dtype=float)
21.     mmax = []
22.     mmin = []
23.     for i in range(c.shape[0]): # 遍历 c 的行
24.         for j in range(c.shape[0]): # 遍历 c 的列
25.             mmax.extend([np.fmax(a[i, :], a[j, :])]) # 取 i 和 j 行的最大值,即求 i 行和 j 行的
并
26.             mmin.extend([np.fmin(a[i, :], a[j, :])]) # 取 i 和 j 行的最大值,即求 i 行和 j 行的
交
27.     for i in range(len(mmax)):
28.         mmax[i] = np.sum(mmax[i]) # 求并的和
29.         mmin[i] = np.sum(mmin[i]) # 求交的和
30.     mmax = np.array(mmax).reshape(c.shape[0], c.shape[1]) # 变换为与 c 同型的矩阵
31.     mmin = np.array(mmin).reshape(c.shape[0], c.shape[1]) # 变换为与 c 同型的矩阵
32.     for i in range(c.shape[0]): # 遍历 c 的行
33.         for j in range(c.shape[1]): # 遍历 c 的列
34.             c[i, j] = mmin[i, j]/mmax[i, j] # 赋值相似度
35.    return c
```

```

36.
37. def MatrixComposition(a, b):
38.     """
39.     合成模糊矩阵 a 和模糊矩阵 b
40.     """
41.     a, b = np.array(a), np.array(b)
42.     c = np.zeros_like(a.dot(b))
43.     for i in range(a.shape[0]): # 遍历 a 的行元素
44.         for j in range(b.shape[1]): # 遍历 b 的列元素
45.             empty = []
46.             for k in range(a.shape[1]):
47.                 empty.append(min(a[i, k], b[k, j])) # 行列元素比小
48.             c[i, j] = max(empty) # 比小结果取大
49.     return c
50.
51. def TransitiveClosure(a):
52.     """
53.     平方法合成传递闭包
54.     """
55.     a = FuzzySimilarMatrix(a) # 用最大最小法构造得到模糊相似矩阵
56.     c = a
57.     while True:
58.         m = c
59.         c = MatrixComposition(MatrixComposition(a, c), MatrixComposition(a, c))
60.         if (c == m).all(): # 闭包条件
61.             return np.around(c, decimals=2) # 返回传递闭包,四舍五入,保留两位小数
62.         break
63.     else:
64.         continue
65.
66. def CutSet(a):
67.     """
68.     水平截集
69.     """
70.     a = TransitiveClosure(a) # 用传递闭包
71.
72.     return np.sort(np.unique(a).reshape(-1))[:-1]
73.
74. def get_classes(temp_pairs):
75.     lists = []
76.
77.     for item1 in temp_pairs:
78.         temp_list = []
79.         for item2 in temp_pairs:
80.             if item1[0] == item2[1]:
81.                 temp_list.append(item2[0])
82.             lists.append(list(set(temp_list)))
83.
84.     return(list(np.unique(lists)))
85.
86. def Result(a):
87.     """
88.     模糊聚类结果
89.     """
90.     lambdas = CutSet(a)
91.     a = TransitiveClosure(a)
92.
93.     classes = []
94.
95.     for lam in lambdas:
96.         if lam == lambdas[0]:
97.             classes.append([[a] for a in range(len(a))])
98.         else:
99.             pairs = np.argwhere(a >= lam)
100.            classes.append(get_classes(pairs))

```

```

101.         return classes
102.
103.     def main():
104.         """
105.         特性指标矩阵
106.         """
107.         input = np.array([[17, 15, 14, 15, 16],
108.                             [18, 16, 13, 14, 12],
109.                             [18, 18, 19, 17, 18],
110.                             [16, 18, 16, 15, 18]])
111.
112.         print("特性指标矩阵\n", input)
113.         print("\n 采用最大值规格化法将数据规格化为\n", MaxNormalization(input))
114.         print("\n 用最大最小法构造得到模糊相似矩阵\n", FuzzySimilarMatrix(input))
115.         print("\n 平方法合成传递闭包\n", TransitiveClosure(input))
116.         print("\n 水平截集为\n", CutSet(input))
117.         print("\n 模糊聚类结果\n", Result(input))
118.
119.
120.     if __name__ == "__main__":
121.         main()

```

4、实验内容

运行和理解示例代码，回答下列问题：

- 1) 为什么按最大最小法得到的一定是一个方阵？且一定是自反方阵？且一定是对称方阵？
- 2) 为什么可以根据水平截集对数据进行分类？（提示：一个等价关系唯一确定一个划分）
- 3) 请解释代码 72 行中两个-1 的含义：

```
return np.sort(np.unique(a).reshape(-1))[:, -1]
```

- 4) 在平方法的代码实现中，如何判断平方后的矩阵是否满足传递性？为什么可以这么判断？
- 5) 请修改代码，将最大最小法替换为算术平均最小法。这会改变最终的聚类结果么？

实验五 遗传算法求解无约束单目标优化问题

1、实验目的

理解遗传算法原理，掌握遗传算法的基本求解步骤，包括选择、交叉、变异等，学会运用遗传算法求解无约束单目标优化问题。

2、背景知识

遗传算法(Genetic Algorithm)是借鉴生物界自然选择、适者生存遗传机制的一种随机搜索方法。遗传算法模拟了进化生物学中的遗传、突变、自然选择以及杂交等现象，是进化算法的一种。对于一个最优化问题，一定数量的候选解(每个候选解称为一个个体)的抽象表示(也称为染色体)的种群向更好的方向解进化，通过一代一代不断繁衍，使种群收敛于最适应的环境，从而求得问题的最优解。进化从完全随机选择的个体种群开始，一代一代繁殖、进化。在每一代中，整个种群的每个个体的适应度被评价,从当前种群中随机地选择多个个体(基于它们的适应度)，通过自然选择、优胜劣汰和突变产生新的种群，该种群在算法的下一次迭代中成为当前种群。传统上，解一般用二进制表示(0 和 1 组成的串)。

遗传算法的主要特点是直接对结构对象进行操作，不存在函数求导、连续、单峰的限定；具有内在的隐闭并行性和更好的全局寻优能力；采用概率化的寻优方法，能自动获取和指导优化搜索，自适应调整搜索方向，不需要确定的规则。遗传算法已被人们广泛地应用于组合优化、机器学习、信号处理、自适应控制和人工智能等领域中的问题求解，已成为现代智能计算中的一项关键技术。

关键术语：

(1)个体(individuals):遗传算法中所处理的对象称为个体。个体通常可以含解的编码表示形式、适应度值等构成成分，因而可看成是一个结构整体。其中，主要成分是编码。

(2)种群(population):由个体构成的集合称为种群。

(3)位串(bit string):解的编码表示形式称为位串。解的编码表示可以是 0、1 二值串、0~9 十进制数字串或其他形式的串，可称为字符串或简称为串。位串和染色体(chromosome)相对应。在遗传算法的描述中，经常不加区分地使用位串和染色体这两个概念。位串/染色体与个体的关系:位串/染色体一般是个体的成分，个体还可含有适应度值等成分。个体、染色体、位串或字符串有时在遗传算法中可不加区分地使用。

(4)种群规模(population scale):又称种群大小,指种群中所含个体的数目。

(5)基因(gene):位串中的每个位或元素统称为基因。基因反映个体的特征。同一位上的基因不同，个体的特征可能也不相同。基因对应于遗传学中的遗传物质单位。在 DNA 序列表示中，遗传物质单位也是用特定编码表示的。遗传算法中扩展了编码的概念，对于可行解，可用 0、1 二值、0~9 十个数字,以及其他形式的编码表示。例如，在 0、1 二值编码下，有一个串 $S=1011$ ，则其中的 1, 0, 1, 1 这 4 个元素分别称为基因。基因和位在遗传算法中也可不加区分地使用。

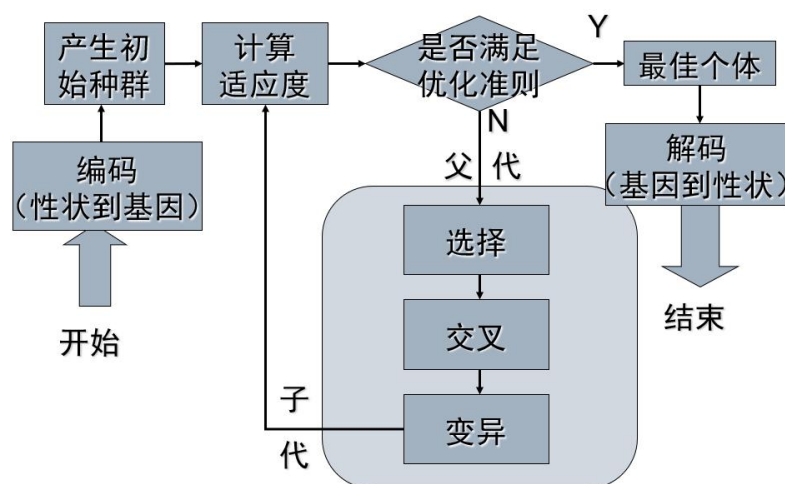
(6)适应度(fitness):个体对环境的适应程度称为适应度(fitness)。为了体现染色体的适应能力，通常引入一个对每个染色体都能进行度量的函数，称为适应度函数。

(7)选择(selection):在整个种群或种群的一部分中选择某个个体的操作。

(8)交叉(crossover):两个个体对应的一个或多个基因段的交换操作。

(9)变异(mutation):个体位串上某个基因的取值发生变化。如在 0、1 串表示下, 某位的值从 0 变为 1, 或由 1 变为 0。

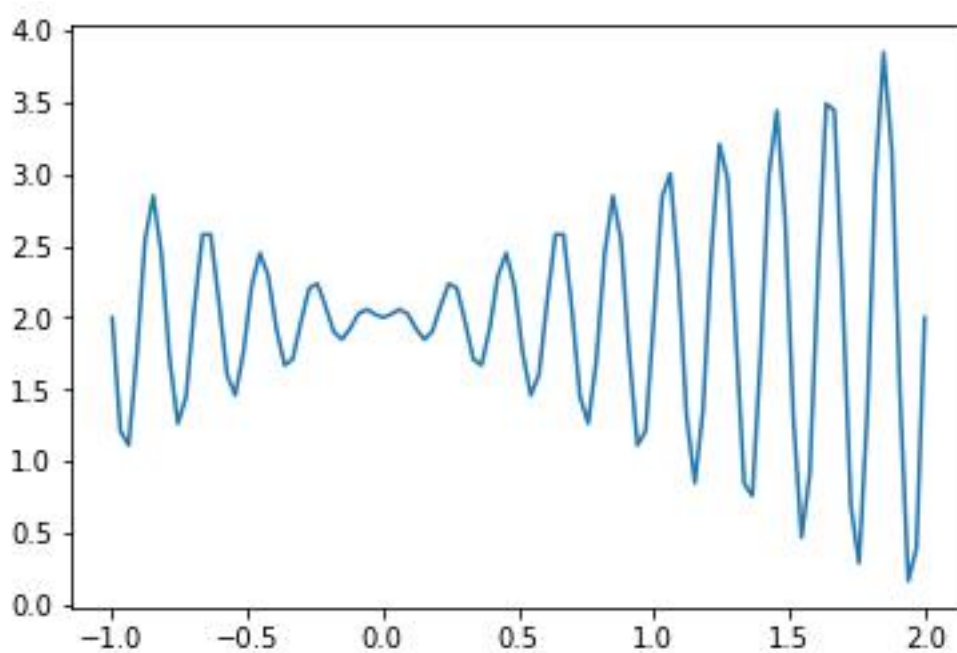
遗传算法的基本流程如下:



本案例意在说明如何使用遗传算法求解无约束单目标优化问题, 即求一元函数

$$f(x) = x \cdot \sin(10\pi \cdot x) + 2$$

在区间 $[-1, 2]$ 上的最大值。该函数图像如下:



由图像可知该函数在在区间 $[-1, 2]$ 上有很多极大值和极小值, 对于求其最大值或最小值的问题, 很多单点优化的方法 (梯度下降等) 就不适合, 这种情况下可以考虑使用遗传算法。。

3、示例代码

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3.
4.
5. def fun(x):
6.     return x * np.sin(10*np.pi*x) + 2
7.
8.
9. Xs = np.linspace(-1, 2, 100)
10.
11. np.random.seed(0) # 令随机数种子=0, 确保每次取得相同的随机数
12.
13. # 初始化原始种群
14. population = np.random.uniform(-1, 2, 10) # 在[-1,2)上以均匀分布生成 10 个浮点数, 做为初始种群
15.
16. for pop, fit in zip(population, fun(population)):
17.     print("x=%5.2f, fit=%5.2f" % (pop, fit))
18.
19. plt.plot(Xs, fun(Xs))
20. plt.plot(population, fun(population), '*')
21. plt.show()
22.
23.
24. def encode(population, _min=-1, _max=2, scale=2**18, binary_len=18): # population 必须为 float 类型, 否则精度不能保证
25.     # 标准化, 使所有数据位于 0 和 1 之间, 乘以 scale 使得数据间距拉大以便使用二进制表示
26.     normalized_data = (population-_min) / (_max-_min) * scale
27.     # 转成二进制编码
28.     binary_data = np.array([np.binary_repr(x, width=binary_len)
29.                             for x in normalized_data.astype(int)])
30.     return binary_data
31.
32.
33. choms = encode(population) # 染色体英文(chromosome)
34.
35.
36. for pop, chrom, fit in zip(population, choms, fun(population)):
37.     print("x=%5.2f, chrom=%s, fit=%5.2f" % (pop, chrom, fit))
38.
39.
40. def decode(popular_gene, _min=-1, _max=2, scale=2**18): # 先把 x 从 2 进制转换为 10 进制, 表示这是第几份
41.     # 乘以每份长度(长度/份数), 加上起点, 最终将一个 2 进制数, 转换为 x 轴坐标
42.     return np.array([(int(x, base=2)/scale*3)+_min for x in popular_gene])
43.
44.
45. tmp = decode(choms)
46. fitness = fun(decode(choms))
47.
48. for pop, chrom, dechrom, fit in zip(population, choms, decode(choms), fitness):
49.     print("x=%5.2f, chrom=%s, dechrom=%5.2f, fit=%5.2f" %
50.           (pop, chrom, dechrom, fit))
51.
52. fitness = fitness - fitness.min() + 0.000001 # 保证所有的都为正
53. print(fitness)
54.
55.
56. def Select_Crossover(choms, fitness, prob=0.6): # 选择和交叉
57.     probs = fitness/np.sum(fitness) # 各个个体被选择的概率
58.     probs_cum = np.cumsum(probs) # 概率累加分布
59.
60.     each_rand = np.random.uniform(size=len(fitness)) # 得到 10 个随机数 0 到 1 之间
```

```

61.
62.     # 轮盘赌, 根据随机概率选择出新的基因编码
63.     # 对于 each_rand 中的每个随机数, 找到被轮盘赌中的那个染色体
64.     newX = np.array([chroms[np.where(probs_cum > rand)[0][0]]
65.                       for rand in each_rand])
66.
67.     # 繁殖, 随机配对 (概率为 0.6)
68.
69.     # 这个数字小于 0.6 就交叉, 则平均下来应有三组进行交叉, 即 6 个染色体要进行交叉
70.     pairs = np.random.permutation(
71.         int(len(newX)*prob//2*2)).reshape(-1, 2) # 产生 6 个随机数乱排一下分成二列
72.     center = len(newX[0])//2 # 交叉方法采用最简单的, 中心交叉法
73.     for i, j in pairs:
74.         # 在中间位置交叉
75.         x, y = newX[i], newX[j]
76.         newX[i] = x[:center] + y[center:] # newX 的元素都是字符串, 直接用+号拼接
77.         newX[j] = y[:center] + x[center:]
78.     return newX
79.
80.
81. chroms = Select_Crossover(chroms, fitness)
82.
83. dechroms = decode(chroms)
84. fitness = fun(dechroms)
85.
86. for gene, dec, fit in zip(chroms, dechroms, fitness):
87.     print("chrom=%s, dec=%5.2f, fit=%.2f" % (gene, dec, fit))
88.
89. # 对比一下选择和交叉之后的结果
90. fig, (axs1, axs2) = plt.subplots(1, 2, figsize=(14, 5))
91. axs1.plot(Xs, fun(Xs))
92. axs1.plot(population, fun(population), 'o')
93. axs2.plot(Xs, fun(Xs))
94. axs2.plot(dechroms, fitness, '*')
95. plt.show()
96.
97. # 输入一个原始种群 1, 输出一个变异种群 2 函数参数中的冒号是参数的类型建议符。
98.
99.
100. def Mutate(chroms: np.array):
101.     prob = 0.1 # 变异的概率
102.     clen = len(chroms[0]) # chroms[0]="111101101 000010110" 字符串的长度=18
103.     m = {'0': '1', '1': '0'} # m 是一个字典, 包含两对: 第一对 0 是 key 而 1 是 value; 第二对 1
        是 key 而 0 是 value
104.     newchroms = [] # 存放变异后的新种群
105.     each_prob = np.random.uniform(size=len(chroms)) # 随机 10 个数
106.
107.     for i, chrom in enumerate(chroms): # enumerate 的作用是整一个 i 出来
108.         if each_prob[i] < prob: # 如果要进行变异(i 的用处在这里)
109.             pos = np.random.randint(clen) # 从 18 个位置随机找一个位置, 假设是 7
110.             # 0~6 保持不变, 8~17 保持不变, 仅将 7 号翻转, 即 0 改为 1, 1 改为 0。注意 chrom 中字符
                不是 1 就是 0
111.             chrom = chrom[:pos] + m[chrom[pos]] + chrom[pos+1:]
112.             newchroms.append(chrom) # 无论 if 是否成立, 都在 newchroms 中增加 chroms 的这个元
                素
113.     return np.array(newchroms) # 返回变异后的种群
114.
115.
116. newchroms = Mutate(chroms)
117.
118.
119. def DrawTwoChroms(chroms1, chroms2, fitfun): # 画 2 幅图, 左边是旧种群, 右边是新种群
120.     Xs = np.linspace(-1, 2, 100)
121.     fig, (axs1, axs2) = plt.subplots(1, 2, figsize=(14, 5))

```



```

122.     dechroms = decode(chroms1)
123.     fitness = fitfun(dechroms)
124.     axs1.plot(Xs, fitfun(Xs))
125.     axs1.plot(dechroms, fitness, 'o')
126.
127.     dechroms = decode(chroms2)
128.     fitness = fitfun(dechroms)
129.     axs2.plot(Xs, fitfun(Xs))
130.     axs2.plot(dechroms, fitness, '*')
131.     plt.show()
132.
133.
134.     # 对比一下变异前后的结果
135.     DrawTwoChroms(chroms, newchroms, fun)
136.
137.     # 上述代码只是执行了一轮，这里反复迭代
138.     np.random.seed(0)
139.     population = np.random.uniform(-1, 2, 100) # 这次多找一些点
140.     chroms = encode(population)
141.
142.     for i in range(1000):
143.         fitness = fun(decode(chroms))
144.         fitness = fitness - fitness.min() + 0.000001 # 保证所有的都为正
145.         newchroms = Mutate(Select_Crossover(chroms, fitness))
146.         if i % 300 == 1:
147.             DrawTwoChroms(chroms, newchroms, fun)
148.             chroms = newchroms
149.
150.     DrawTwoChroms(chroms, newchroms, fun)

```

4、实验内容

运行和理解示例代码，回答下列问题：

1) 代码第 64 行的语义是什么？两个[0]各自代表什么？最后 newX 有几个元素？

```

1.     newX = np.array([chroms[np.where(probs_cum > rand)[0][0]]
2.                     for rand in each_rand])

```

2) 代码第 70 行的语义是什么？为什么要除以 2 再乘以 2？reshape 中的-1 表示什么？

```

1.     pairs = np.random.permutation(
2.         int(len(newX)*prob//2*2)).reshape(-1, 2) # 产生 6 个随机数乱排一下分成二列

```

3) 请结合 Mutate 函数的内容，详述变异是如何实现的。

4) 将代码第 145 行修改为 newchroms = Select_Crossover(chroms, fitness)，即不再执行变异，执行结果有什么不同，为什么会出现这种变化？

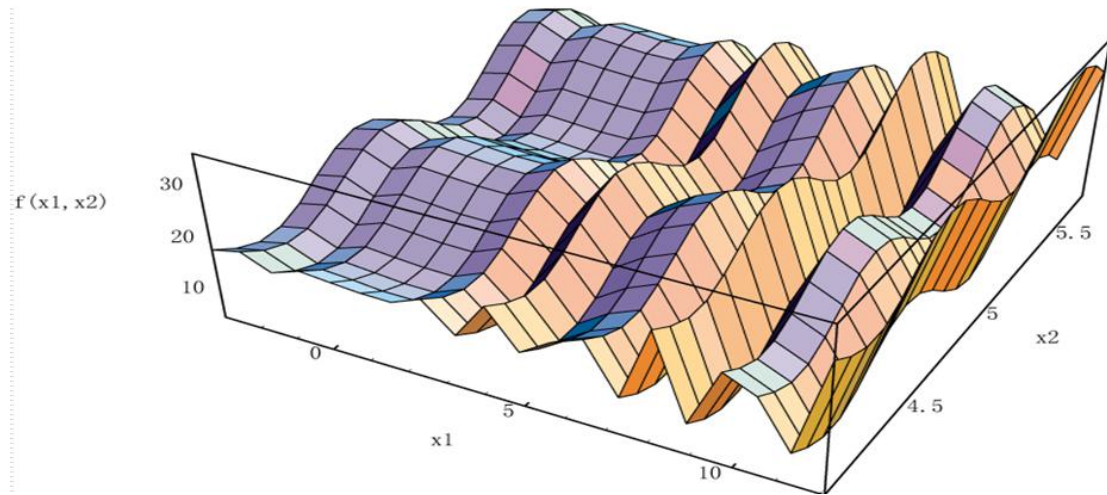
5) 轮盘让个体按概率被选择，对于适应度最高的个体而言，虽然被选择的概率高，但仍有可能被淘汰，从而在进化过程中失去当前最优秀的个体。一种改进方案是，让适应度最高的那个个体不参与选择，而是直接进入下一轮（直接晋级），这种方案被称为精英选择(elitist selection)。请修改 Select 部分的代码，实现这一思路。

6) 【选做】请借鉴示例代码，实现教材 P57 的例 2.6.1，即用遗传算法求解下列二元函数的最大值。

$$\max f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi \cdot x_1) + x_2 \cdot \sin(20\pi \cdot x_2)$$

$$\text{s. t. } -2.9 \leq x_1 \leq 12.0$$

$$4.2 \leq x_2 \leq 5.7$$



湖南科技大学计算机科学与工程学院

课程设计报告

专业班级：_____

姓 名：_____

学 号：_____

指导教师：_____

时 间：_____

地 点：_____

指导教师评语：

成绩：

等级：

签名：_____

年 月 日

正文格式如下：

一、实验题目（四号宋体，加粗，以下标题的字体格式与此相同）

二、实验目的

例如：理解多层神经网络的结构和原理。（正文小四号宋体）

三、实验内容（实验原理/运用的理论知识、算法/程序流程图、步骤和方法、关键源代码）

（正文小四号宋体，关键源代码五号字体）

四、实验结果与分析

五、小结与心得体会

附件 2：课程设计报告日志参考格式

时间		设计内容简要记录
第 周	星期一 ()	
	星期二 ()	
	星期三 ()	
	星期四 ()	
	星期五 ()	

周 末	() ()	
第 周	星期一 ()	
	星期二 ()	
	星期三 ()	
	星期四 ()	
	星期五 ()	

注：请同学们及时填写，以便指导老师及时了解你的课程设计进展等情况。表格空间不够可自行加页。

计算机科学与工程学院课程设计成绩单

上课班级:

考核方式: 考查

[illegible]

2.平时成绩（0—20分）、测试成绩（0-40分）、创新成绩（0-10分）以及报告成绩（30分）
 这四项求和为总成绩（0—100分）。

在课程设计报告封面上附上百分制成绩和等级制成绩。