人工智能实践：TensorFlow 笔记

第一讲

神经网络计算过程及模型搭建

本讲目标：了解神经网络计算过程，搭建出第一个神经网络模型。

1 人工智能三学派

我们常说的人工智能，就是让机器具备人的思维和意识。人工智能主要有三个学派，即**行为主义、符号主义**和**连接主义**。

行为主义：是基于控制论的，是在构建感知、动作的控制系统。单脚站立是行为主义一个典型例子，通过感知要摔倒的方向，控制两只手的动作，保持身体的平衡。这就构建了一个感知、动作的控制系统，是典型的行为主义。

符号主义：基于算数逻辑表达式。即在求解问题时，先把问题描述为表达式，再求解表达式。例如在求解某个问题时，利用 if case 等条件语句和若干计算公式描述出来，即使用了符号主义的方法，如专家系统。符号主义是能用公式描述的人工智能，它让计算机具备了理性思维。

连接主义：仿造人脑内的神经元连接关系，使人类不仅具备理性思维，还具备无法用公式描述的感性思维，如对某些知识产生记忆。

图 1.1 展示了人脑中的一根神经元，其中紫色部分为树突，其作为神经元的输入。黄色部分为轴突，其作为神经元的输出。人脑就是由 860 亿个这样的神经元首尾相接组成的网络。

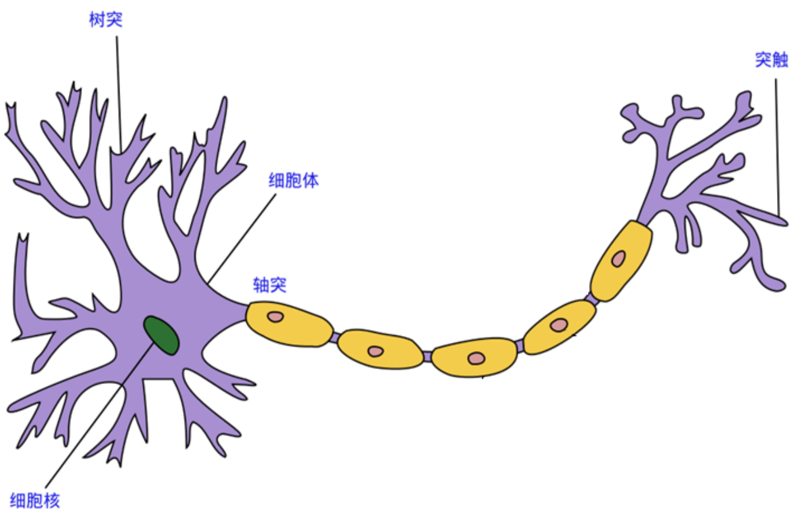

图 1.1 神经元示意图

基于连接主义的神经网络模仿上图的神经元，使计算机具有感性思维。图 1.2 展示了从出生到成年，人脑中神经网络的变化。

图 1.2 人脑神经网络变化示意图

随着我们的成长，大量的数据通过视觉、听觉涌入大脑，使我们的神经网络连接，也就是这些神经元连接线上的权重发生了变化，有些线上的权重增强了，有些线上的权重减弱了。如图 1.3 所示。

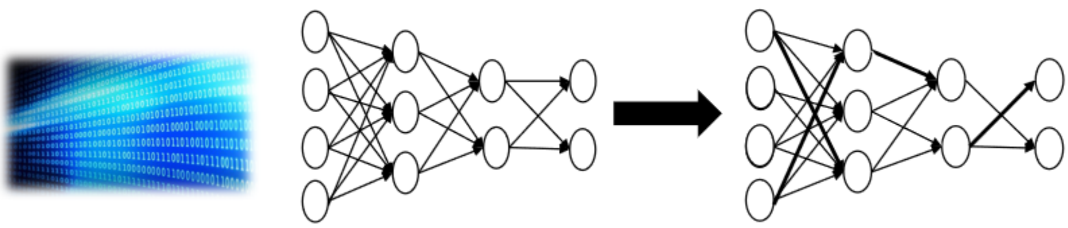

图 1.3 神经网络权重变化示意图

2 神经网络设计过程

我们要用计算机模仿刚刚说到的神经网络连接关系，让计算机具备感性思维。

1、准备数据 首先，需要**准备数据**，数据量越大越好，要构成特征和标签对。如要识别猫，就要有大量猫的图片和这个图片是猫的标签，构成特征标签对。

随后，搭建神经网络的**网络结构**，并通过**反向传播**，优化连线的权重，直到

2、搭建网络 3、优化参数

模型的识别准确率达到要求，得到最优的连线权重，把这个**模型保存**起来。

4、应用网络　最后，用保存的模型，输入从未见过的新数据，它会通过**前向传播**，输出概率值，概率值最大的一个，就是分类或预测的结果。图 2.1 展示了搭建与使用神经网络模型的流程。

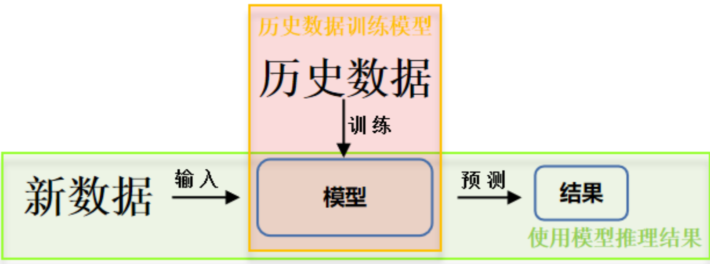

图 2.1 搭建与使用神经网络示意图

2.1 数据集介绍

本讲中采用鸢尾花数据集，此数据集包含鸢尾花花萼长、花萼宽、花瓣长、花瓣宽及对应的类别。其中前 4 个属性作为输入特征，类别作为标签，0 代表狗尾草鸢尾，1 代表杂色鸢尾，2 代表弗吉尼亚鸢尾。人们通过对数据进行分析总结出了规律：通过测量花的花萼长、花萼宽、花瓣长、花瓣宽，可以得出鸢尾花的类别（如：花萼长>花萼宽且花瓣长/花瓣宽>2 ，则杂色鸢尾）。逻辑判别（理性计算）

由上述可知，可通过 if 与 case 语句构成专家系统，进行判别分类。在本讲中，采用搭建神经网络的办法对其进行分类，即将鸢尾花花萼长、花萼宽、花瓣长、花瓣宽四个输入属性喂入搭建好的神经网络，网络优化参数得到模型，输出分类结果。

2.2 网络搭建与训练

本讲中，我们搭建包含输入层与输出层的神经网络模型，通过对输入值乘权值，并于偏置值求和的方式得到输出值，图示如下。

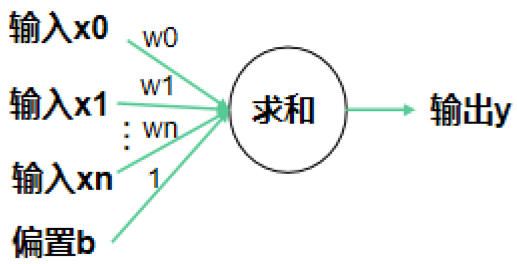

图 2.2 鸢尾花神经网络简要模型

由图 2.2 可知输出 $y = x * w + b$，即所有的输入 x 乘以各自线上的权重 w 求和加上偏置项 b 得到输出 y。由 2.1 部分对数据集的介绍可知，输入特征 x 形状应为(1,4)即 1 行 4 列，输出 y 形状应为(1,3)即 1 行 3 列，w 形状应为(4,3)即 4 行 3 列，b 形状应为(3,)即有 3 个偏置项。

随机初始化为一些随机数

搭建好基本网络后，需要输入特征数据，并对线上权重 w 与偏置 b 进行初始化。搭建的神经网络如图 2.3 所示，w,b 初始化矩阵如图 2.4 所示。在这里，我们输入标签为 0 的狗尾草鸢尾。

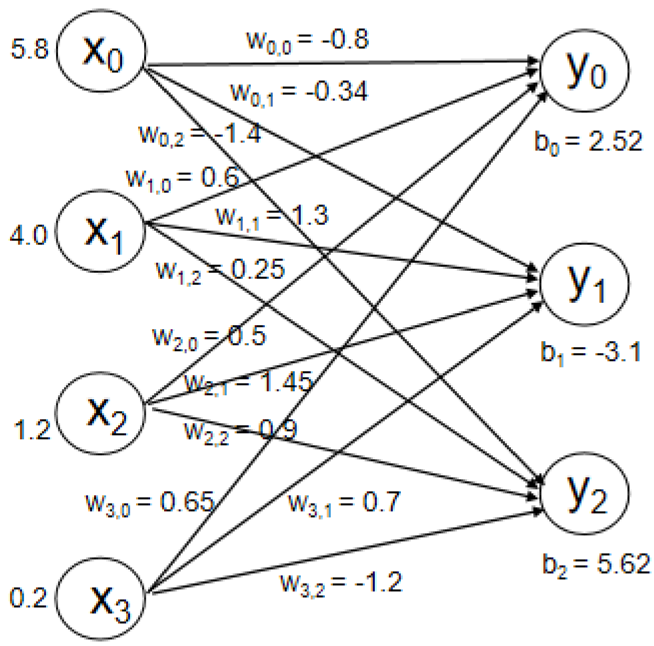

图 2.3 鸢尾花神经网络展开模型

$$\begin{bmatrix} -0.8 & -0.34 & -1.4 \\ 0.6 & 1.3 & 0.25 \\ 0.5 & 1.45 & 0.9 \\ 0.65 & 0.7 & -1.2 \end{bmatrix} \quad \begin{bmatrix} 2.52 & -3.1 & 5.62 \end{bmatrix}$$

图 2.4 权重与偏置初始化矩阵

有了输入数据与线上权重等数据，即可按照 $y = x * w + b$ 方式进行前向传播，计算过程如图 2.5 所示。

$$\underset{\text{输入特征x}}{\begin{bmatrix} 5.8 & 4.0 & 1.2 & 0.2 \end{bmatrix}} * \underset{\text{随机初始化w}}{\begin{bmatrix} -0.8 & -0.34 & -1.4 \\ 0.6 & 1.3 & 0.25 \\ 0.5 & 1.45 & 0.9 \\ 0.65 & 0.7 & -1.2 \end{bmatrix}} + \underset{\text{随机初始化b}}{\begin{bmatrix} 2.52 & -3.1 & 5.62 \end{bmatrix}} = \underset{\text{输出y}}{\begin{bmatrix} 1.01 & 2.01 & -0.66 \end{bmatrix}}$$

图 2.5 前向传播计算过程

图 2.5 中输出 y 中，1.01 代表 0 类鸢尾得分，2.01 代表 1 类鸢尾得分，-0.66 代表 2 类鸢尾得分。通过输出 y 可以看出数值最大(可能性最高)的是 1 类鸢尾，而不是标签 0 类鸢尾。这是由于最初的参数 w 和 b 是随机产生的，现在输出的结果是蒙的。

为了修正这一结果，我们用**损失函数**，定义预测值 y 和标准答案(标签) $y\_$ 的差距，损失函数可以定量的判断当前这组参数 w 和 b 的优劣，当损失函数最小时，即可得到最优 w 的值和 b 的值。

损失函数的定义有多种方法，均方误差就是一种常用的损失函数，它计算每个前向传播输出 y 和标准答案 $y\_$ 的差求平方再求和再除以 n 求平均值，表征了网络前向传播推理结果和标准答案之间的差距。

通过上述对损失函数的介绍，其目的是寻找一组参数 w 和 b 使得损失函数最小。为达成这一目的，我们采用梯度下降的方法。损失函数的梯度表示损失函数

对各参数求偏导后的向量，损失函数梯度下降的方向，就是是损失函数减小的方向。梯度下降法即沿着损失函数梯度下降的方向，寻找损失函数的最小值，从而得到最优的参数。梯度下降法涉及的公式如下

$$w_{t+1} = w_t - lr * \frac{\partial loss}{\partial w_t}$$

$$b_{t+1} = b_t - lr * \frac{\partial loss}{\partial b_t}$$

$$w_{t+1} * x + b_{t+1} \rightarrow y$$

上式中，lr 表示学习率，是一个超参数，表征梯度下降的速度。如学习率设置过小，参数更新会很慢，如果学习率设置过大，参数更新可能会跳过最小值。

上述梯度下降更新的过程为**反向传播**，下面通过例子感受反向传播。利用如下公式对参数 w 进行更新。

$$w_{t+1} = w_t - lr * \frac{\partial loss}{\partial w_t}$$

设损失函数为 $(w+1)^2$，则其对 w 的偏导数为 $2w+2$。设 w 在初始化时被随机初始化为 5，学习率设置为 0.2。则我们可按上述公式对 w 进行更新：

第一次参数为 5，按上式计算即 $5-0.2\times(2\times5+2)$ =2.6。

同理第二次计算得到参数为 1.16，第三次计算得到参数为 0.296……

画出损失函数 $(w+1)^2$ 的图像，可知 $w=-1$ 时损失函数最小，我们反向传播优化参数的目的即为找到这个使损失函数最小的 $w=-1$ 值。

3 TensorFlow2.1 基本概念与常见函数

3.1 基本概念

TensorFlow 中的 Tensor 表示张量，是多维数组、多维列表，用阶表示张量的维数。0 阶张量叫做标量，表示的是一个单独的数，如 123；1 阶张量叫作向量，表示的是一个一维数组如[1,2,3]；2 阶张量叫作矩阵，表示的是一个二维数组，它可以有 i 行 j 列个元素，每个元素用它的行号和列号共同索引到，如在[[1,2,3],[4,5,6],[7,8,9]]中，2 的索引即为第 0 行第 1 列。张量的阶数与方括号的数量相同，0 个方括号即为 0 阶张量，1 个方括号即为 1 阶张量。故张量可以表示

0 阶到 n 阶的数组。也可通过 reshape 的方式得到更高维度数组，举例如下：

```
c = np.arange(24).reshape(2,4,3)
print(c)
```

输出结果：[[[0 1 2] [3 4 5] [6 7 8] [9 10 11]]

[[12 13 14] [15 16 17] [18 19 20] [21 22 23]]]

TensorFlow 中数据类型包括 32 位整型(tf.int32)、32 位浮点(tf.float32)、64 位浮点(tf.float64)、布尔型(tf.bool)、字符串型(tf.string)

创建张量有若干种不同的方法：

(1) 利用 tf.constant(张量内容，dtype=数据类型(可选))，第一个参数表示张量内容，第二个参数表示张量的数据类型。举例如下：

```
import tensorflow as tf a=tf.constant([1,5],dtype=tf.int64)
print(a)
print(a.dtype)
print(a.shape)
```

shape的逗号隔开了几个数字，张量就是几维

输出结果为：<tf.Tensor([1,5], shape=(2 ,) , dtype=int64)>

<dtype: 'int64'>

(2,)

即会输出张量内容、形状与数据类型，shape 中数字为 2，表示一维张量里有 2 个元素。

注：去掉 dtype 项，不同电脑环境不同导致默认值不同，可能导致后续程序 bug

(2) 很多时候数据是由 numpy 格式给出的，此时可以通过如下函数将 numpy 格式化为 Tensor 格式：tf. convert_to_tensor(数据名，dtype=数据类型(可选))。举例如下：

```
import tensorflow as tf
import numpy as np
a = np.arange(0, 5)
b = tf.convert_to_tensor( a,    dtype=tf.int64 )
print(a)
print(b)
```

输出结果为：[0 1 2 3 4]

tf.Tensor([0 1 2 3 4], shape=(5 ,), dtype=int64)

可见，将 numpy 格式的 a 转换成了 Tensor 格式的 b。

(3) 可采用不同函数创建不同值的张量。如用 tf. zeros(维度)创建全为 0 的张量，tf.ones(维度)创建全为 1 的张量，tf. fill(维度，指定值)创建全为指定值的张量。其中维度参数部分，如一维则直接写个数，二维用[行，列]表示，多维用[n,m,j..]表示。举例如下：

```
a = tf.zeros([2, 3])
b = tf.ones(4)
c = tf.fill([2, 2], 9)
print(a)
print(b)
print(c)
```

Tensor默认数据类型是float32

输出结果：tf.Tensor([[0. 0. 0.] [0. 0. 0.]], shape=(2, 3), dtype=float32)

tf.Tensor([1. 1. 1. 1.], shape=(4,), dtype=float32)

tf.Tensor([[9 9] [9 9]], shape=(2, 2), dtype=int32)

可见，tf.zeros([2,3])创建了一个二维张量，第一个维度有两个元素，第二个维度有三个元素，元素的内容全是 0；tf.ones(4)创建了一个一维张量，里边有 4 个元素，内容全是 1；tf.fill([2,2],9)创建了一个两行两列的二维张量，第一个维度有两个元素，第二个维度也有两个元素，内容都是 9。

(4) 可采用不同函数创建符合不同分布的张量。如用 tf. random.normal（维度，mean=均值，stddev=标准差)生成正态分布的随机数，默认均值为 0，标准差为 1；用 tf. random.truncated_normal (维度，mean=均值，stddev=标准差)生成截断式正态分布的随机数，能使生成的这些随机数更集中一些，如果随机生成数据的取值在 $(\mu-2\sigma, u+2\sigma)$ 之外则重新进行生成，保证了生成值在均值附近；利用 tf. random. uniform(维度，minval=最小值，maxval=最大值)，生成指定维度的均匀分布随机数，用 minval 给定随机数的最小值，用 maxval 给定随机数的最大值，最小、最大值是前闭后开区间。举例如下：

数据一定在两倍标准差内，数据更向均值集中

```
d = tf.random.normal ([2, 2], mean=0.5, stddev=1)
```

print(d)

e = tf.random.truncated_normal ([2, 2], mean=0.5, stddev=1)

print(e)

f = tf.random.uniform([2, 2], minval=0, maxval=1)

print(f)

输出结果：tf.Tensor([[0.7925745   0.643315]

[1.4752257 0.2533372]], shape=(2, 2), dtype=float32)

tf.Tensor([[1.3688478   1.0125661]

[0.17475659 -0.02224463]], shape=(2, 2), dtype=float32)

tf.Tensor([[0.28219545 0.15581512]

[0.77972126 0.47817433]], shape=(2, 2), dtype=float32)

3.2 常用函数

(1)利用 tf.cast (张量名，dtype=数据类型)强制将 Tensor 转换为该数据类型；利用 tf.reduce_min (张量名)计算张量维度上元素的最小值；利用 tf.reduce_max (张量名)计算张量维度上元素的最大值。举例如下：

x1 = tf.constant ([1., 2., 3.], dtype=tf.float64)

print(x1)

x2 = tf.cast (x1, tf.int32)

print(x2)

print (tf.reduce_min(x2), tf.reduce_max(x2))

输出结果：tf.Tensor([1. 2. 3.], shape=(3,), dtype=float64)

tf.Tensor([1 2 3], shape=(3,), dtype=int32)

tf.Tensor(1, shape=(), dtype=int32)

tf.Tensor(3, shape=(), dtype=intt32)

(2) 可用 tf.reduce_mean (张量名，axis=操作轴)计算张量沿着指定维度的平均值；可用 f.reduce_sum (张量名，axis=操作轴)计算张量沿着指定维度的和，如不指定 axis，则表示对所有元素进行操作。其中维度可按图 3.1 理解。

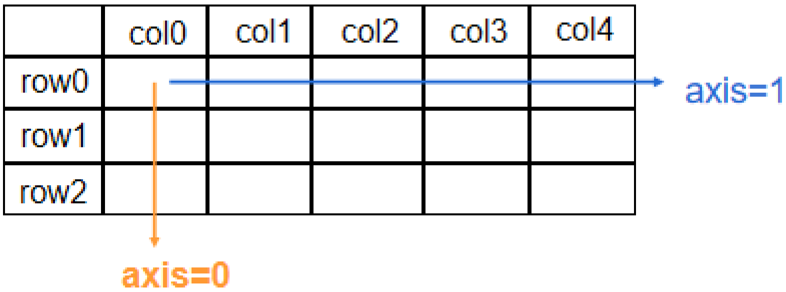

图 3.1 维度定义

由上图可知对于一个二维张量，如果 axis=0 表示纵向操作(沿经度方向) ，axis=1 表示横向操作(沿纬度方向)。举例如下：

x=tf.constant([[1, 2,3],[2, 2, 3]])

print(x)

print(tf.reduce_mean(x))

print(tf.reduce_sum(x, axis=1))

输出结果：tf.Tensor([[1 2 3] [2 2 3]], shape=(2, 3), dtype=int32)

tf.Tensor(2, shape=(), dtype=int32) (对所有元素求均值) 因为数据类型是int，所以小数位被舍弃了，mean后是2

tf.Tensor([6 7], shape=(2,), dtype=int32) (横向求和，两行分别为 6 和 7)

(3) 可利用 tf.Variable(initial_value,trainable,validate_shape,name)函数可以将变量标记为“可训练”的，被它标记了的变量，会在反向传播中记录自己的梯度信息。其中 initial_value 默认为 None，可以搭配 tensorflow 随机生成函数来初始化参数；trainable 默认为 True，表示可以后期被算法优化的，如果不想该变量被优化，即改为 False；validate_shape 默认为 True，形状不接受更改，如果需要更改，validate_shape=False；name 默认为 None，给变量确定名称。举例如下：

w = tf.Variable(tf.random.normal([2, 2], mean=0, stddev=1))，表示首先随机生成正态分布随机数，再给生成的随机数标记为可训练，这样在反向传播中就可以通过梯度下降更新参数 *w* 了。

(4) 利用 TensorFlow 中函数对张量进行四则运算。利用 tf.add (张量 1,张量 2)实现两个张量的对应元素相加；利用 tf.subtract (张量 1,张量 2)实现两个张量的对应元素相减；利用 tf.multiply (张量 1,张量 2)实现两个张量的对应元素相乘；利用 tf.divide (张量 1,张量 2)实现两个张量的对应元素相除。**注：只有维度相同的张量才可以做四则运算**，举例如下：

```
a = tf.ones([1, 3])
b = tf.fill([1, 3], 3.)
print(a)
print(b)
print(tf.add(a,b))
print(tf.subtract(a,b))
print(tf.multiply(a,b))
print(tf.divide(b,a))
```

输出结果：tf.Tensor([[1. 1. 1.]], shape=(1, 3), dtype=float32)

tf.Tensor([[3. 3. 3.]], shape=(1, 3), dtype=float32

tf.Tensor([[4. 4. 4.]], shape=(1, 3), dtype=float32)

tf.Tensor([[-2. -2. -2.]], shape=(1, 3), dtype=float32)

tf.Tensor([[3. 3. 3.]], shape=(1, 3), dtype=float32)

tf.Tensor([[3. 3. 3.]], shape=(1, 3), dtype=float32)

(5) 利用 TensorFlow 中函数对张量进行幂次运算。可用 tf.square（张量名)计算某个张量的平方；利用 tf.pow (张量名，n 次方数)计算某个张量的 n 次方；利用 tf.sqrt (张量名)计算某个张量的开方。举例如下：

```
a = tf.fill([1, 2], 3.)
print(a)
print(tf.pow(a, 3))
print(tf.square(a))
print(tf.sqrt(a))
```

输出结果：tf.Tensor([[3. 3.]], shape=(1, 2), dtype=float32)

tf.Tensor([[27. 27.]], shape=(1, 2), dtype=float32)

tf.Tensor([[9. 9.]], shape=(1, 2), dtype=float32)

tf.Tensor([[1.7320508 1.7320508]], shape=(1, 2), dtype=float32)

(6) 可利用 tf.matmul(矩阵 1，矩阵 2)实现两个矩阵的相乘。举例如下：

```
a = tf.ones([3, 2])
b = tf.fill([2, 3], 3.)
print(tf.matmul(a, b))
```

$$\begin{matrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{matrix} \qquad \begin{matrix} 3 & 3 & 3 \\ 3 & 3 & 3 \end{matrix}$$

输出结果：tf.Tensor([[6. 6. 6.] [6. 6. 6.] [6. 6. 6.]], shape=(3, 3), dtype=float32)，即 a 为一个 3 行 2 列的全 1 矩阵，b 为 2 行 3 列的全 3 矩阵，二者进行矩阵相乘。

(7) 可利用 tf.data.Dataset.from_tensor_slices((输入特征, 标签))切分传入张量的第一维度，生成输入特征/标签对，构建数据集，此函数对 Tensor 格式与 Numpy 格式均适用，其切分的是第一维度，表征数据集中数据的数量，之后切分 batch 等操作都以第一维为基础。举例如下：

```
features = tf.constant([12,23,10,17])
labels = tf.constant([0, 1, 1, 0])
dataset = tf.data.Dataset.from_tensor_slices((features, labels))
print(dataset)
for element in dataset:
    print(element)
```

输出结果：<TensorSliceDataset shapes: ((),()), types: (tf.int32, tf.int32))>

(<tf.Tensor: id=9, shape=(), dtype=int32, numpy=12>, <tf.Tensor: id=10, shape=(), dtype=int32, numpy=0>)

(<tf.Tensor: id=11, shape=(), dtype=int32, numpy=23>, <tf.Tensor: id=12, shape=(), dtype=int32, numpy=1>)

(<tf.Tensor: id=13, shape=(), dtype=int32, numpy=10>, <tf.Tensor: id=14, shape=(), dtype=int32, numpy=1>)

(<tf.Tensor: id=15, shape=(), dtype=int32, numpy=17>, <tf.Tensor: id=16, shape=(), dtype=int32, numpy=0>)

即将输入特征 12 和标签 0 对应，产生配对；将输入特征 23 和标签 1 对应，产生配对……

(8) 可利用 tf.GradientTape( )函数搭配 with 结构计算损失函数在某一张量处的梯度，举例如下：

可以在with结构中，使用GradientTape实现某个函数对指定参数的求导运算

配合Variable函数，可以实现损失函数loss对参数w的求导计算

```
with tf.GradientTape( ) as tape:
    w = tf.Variable(tf.constant(3.0))
    loss = tf.pow(w,2)
grad = tape.gradient(loss,w)
print(grad)
```

输出结果：tf.Tensor(6.0, shape=(), dtype=float32) 在上例中，损失函数为 w^2，w 当前取值为 3，故计算方式为 $\frac{\partial w^2}{\partial w} = 2w = 2 \times 0.3 = 0.6$。

(9) 可利用 enumerate(列表名)函数枚举出每一个元素，并在元素前配上对应的索引号，常在 for 循环中使用。举例如下：

组合为：索引 元素

```
seq = ['one', 'two', 'three']
for i, element in enumerate(seq):
    print(i, element)
```

输出结果：0　one

1　two

2　three

(10) 可用 tf.one_hot(待转换数据，depth=几分类)函数实现用独热码表示标签，在分类问题中很常见。标记类别为为 1 和 0，其中 1 表示是，0 表示非。如在鸢尾花分类任务中，如果标签是 1，表示分类结果是 1 杂色鸢尾，其用把它用独热码表示就是 0,1,0，这样可以表示出每个分类的概率：也就是百分之 0 的可能是 0 狗尾草鸢尾，百分百的可能是 1 杂色鸢尾，百分之 0 的可能是弗吉尼亚鸢尾。举例如下：

```
classes = 3
labels = tf.constant([1,0,2])
output = tf.one_hot( labels, depth=classes )
print(output)
```

输出结果：tf.Tensor([[0. 1. 0.]　[1. 0. 0.]　[0. 0. 1.]], shape=(3, 3), dtype=float32)

索引从 0 开始，待转换数据中各元素值应小于 depth，若带转换元素值大于等于 depth，则该元素输出编码为 [0, 0 … 0, 0]。即 depth 确定列数，待转换元素的个数确定行数。举例如下：

```
classes = 3
labels = tf.constant([1,4,2])          # 输入的元素值 4 超出 depth-1
output = tf.one_hot(labels,depth=classes)
print(output)
```

输出结果：tf.Tensor([[0. 1. 0.] [0. 0. 0.] [0. 0. 1.]], shape=(3, 3), dtype=float32)
即元素 4 对应的输出编码为[0. 0. 0.]。

(11) 可利用 tf.nn.softmax( )函数使前向传播的输出值符合概率分布，进而与独热码形式的标签作比较，其计算公式为 $\frac{e^{y_i}}{\sum_{j=0}^{n} e^{y_i}}$，其中 y_i 是前向传播的输出。在前一部分，我们得到了前向传播的输出值，分别为 1.01、2.01、-0.66，通过上述计算公式，可计算对应的概率值：

softmax函数可以使n分类的n个输出符合概率分布，也就是每个输出值变为0-1之间的概率值，这些概率的和是1

$$\frac{e^{y_0}}{e^{y_0}+e^{y_1}+e^{y_2}}=\frac{2.75}{10.73}=0.256$$
$$\frac{e^{y_1}}{e^{y_0}+e^{y_1}+e^{y_2}}=\frac{7.46}{10.73}=0.695$$
$$\frac{e^{y_2}}{e^{y_0}+e^{y_1}+e^{y_2}}=\frac{0.52}{10.73}=0.048$$

上式中，0.256 表示为 0 类鸢尾的概率是 25.6%，0.695 表示为 1 类鸢尾的概率是 69.5%，0.048 表示为 2 类鸢尾的概率是 4.8%。程序实现如下：

```
y = tf.constant ( [1.01, 2.01, -0.66] )
y_pro = tf.nn.softmax(y)
print("After softmax, y_pro is:", y_pro)
```

输出结果：After softmax, y_pro is:
tf.Tensor([0.25598174 0.69583046 0.0481878], shape=(3,), dtype=float32)与上述计算结果相同。

(12) 可利用 assign_sub 对参数实现自更新。使用此函数前需利用 tf.Variable 定义变量 w 为可训练(可自更新)，举例如下：

```
w = tf.Variable(4)
w.assign_sub(1)  w -= 1，即w=w-1
print(w)
```

输出结果：<tf.Variable 'Variable:0' shape=() dtype=int32, numpy=3> 即实现了参数 w 自减 1。注：直接调用 tf.assign_sub 会报错，要用 w.assign_sub。

(13) 可利用 tf.argmax (张量名,axis=操作轴)返回张量沿指定维度最大值的索引，维度定义与图 3.1 一致。举例如下：

axis=0 纵向
axis=1 横向

```
import numpy as np
```

```
test = np.array([[1, 2, 3], [2, 3, 4], [5, 4, 3], [8, 7, 2]])
print(test)
print( tf.argmax (test, axis=0))   # 返回每一列（经度）最大值的索引
print( tf.argmax (test, axis=1))   # 返回每一行（纬度）最大值的索引
```

输出结果：[[1 2 3] [2 3 4] [5 4 3] [8 7 2]]

tf.Tensor([3 3 1], shape=(3,), dtype=int64)

tf.Tensor([2 2 0 0], shape=(4,), dtype=int64)

4 程序实现鸢尾花数据集分类

4.1 数据集回顾

先回顾鸢尾花数据集，其提供了 150 组鸢尾花数据，每组包括鸢尾花的花萼长、花萼宽、花瓣长、花瓣宽 4 个输入特征，同时还给出了这一组特征对应的鸢尾花类别。类别包括狗尾鸢尾、杂色鸢尾、弗吉尼亚鸢尾三类， 分别用数字 0、1、2 表示。使用此数据集代码如下：

```
from sklearn.datasets import load_iris
x_data = datasets.load_iris().data      # 返回 iris 数据集所有输入特征
y_data = datasets.load_iris().target    # 返回 iris 数据集所有标签
```

即从 sklearn 包中导出数据集，将输入特征赋值给 x_data 变量，将对应标签赋值给 y_data 变量。

4.2 程序实现

我们用神经网络实现鸢尾花分类仅需要三步：

(1)准备数据，包括数据集读入、数据集乱序，把训练集和测试集中的数据配成输入特征和标签对，生成 train 和 test 即永不相见的训练集和测试集；

(2)搭建网络，定义神经网络中的所有可训练参数；

(3)优化这些可训练的参数，利用嵌套循环在 with 结构中求得损失函数 loss 对每个可训练参数的偏导数，更改这些可训练参数，为了查看效果，程序中可以加入每遍历一次数据集显示当前准确率，还可以画出准确率 acc 和损失函数 loss 的变化曲线图。以上部分的完整代码与解析如下：

(1) 数据集读入：

```
from sklearn.datasets import datasets
```

```
x_data = datasets.load_iris().data        # 返回 iris 数据集所有输入特征
y_data = datasets.load_iris().target      # 返回 iris 数据集所有标签
```

(2) 数据集乱序：

```
np.random.seed(116)       # 使用相同的 seed，使输入特征/标签一一对应
np.random.shuffle(x_data)
np.random.seed(116)
np.random.shuffle(y_data)
tf.random.set_seed(116)  # 每次运行这个代码文件时，结果跟上次运行的结果一样
```

(3) 数据集分割成永不相见的训练集和测试集：

```
x_train = x_data[:-30]  把打乱后的数据集前120个数据取出作为训练集
y_train = y_data[:-30]
x_test = x_data[-30:]
y_test = y_data[-30:]
```

(4) 配成[输入特征，标签]对，每次喂入一小撮(batch)：

```
train_db = tf.data.Dataset.from_tensor_slices((x_train, y_train)).batch(32)
test_db = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(32)
```

上述四小部分代码实现了数据集读入、数据集乱序、将数据集分割成永不相见的训练集和测试集、将数据配成[输入特征，标签]对。人类在认识这个世界的时候信息是没有规律的，杂乱无章的涌入大脑的，所以喂入神经网络的数据集也需要被打乱顺序。(2)部分实现了让数据集乱序，因为使用了同样的随机种子，所以打乱顺序后输入特征和标签仍然是一一对应的。(3)部分将打乱后的前 120 个数据取出来作为训练集，后 30 个数据作为测试集，为了公正评判神经网络的效果，训练集和测试集没有交集。(4)部分使用 from_tensor_slices 把训练集的输入特征和标签配对打包，将每 32 组输入特征标签对打包为一个 batch，在喂入神经网络时会以 batch 为单位喂入。

(5) 定义神经网络路中所有可训练参数：

输入特征是4

只用一层网络，输出节点数等于分类数（3分类）

```
w1 = tf.Variable(tf.random.truncated_normal([ 4,   3 ], stddev=0.1, seed=1))
b1 = tf.Variable(tf.random.truncated_normal([ 3 ], stddev=0.1, seed=1))
```

(6) 嵌套循环迭代，with 结构更新参数，显示当前 loss：

1个epoch = batch_size * step

```
for epoch in range(epoch):    #数据集级别迭代
    for step, (x_train, y_train) in enumerate(train_db):    #batch 级别迭代
        with tf.GradientTape() as tape:    # 记录梯度信息
            (前向传播过程计算 y)
            (计算总 loss)
        grads = tape.gradient(loss, [ w1, b1 ])  损失函数loss分别对参数w1和b1计算偏导数
        w1.assign_sub(lr * grads[0])    #参数自更新
        b1.assign_sub(lr * grads[1])
    print("Epoch {}, loss: {}".format(epoch, loss_all/4))
```

上述两部分完成了定义神经网路中所有可训练参数、嵌套循环迭代更新参数。(5)部分定义了神经网络的所有可训练参数。只用了一层网络，因为输入特征是 4 个，输出节点数等于分类数，是 3 分类，故参数 w_1 为 4 行 3 列的张量，b_1 必须与 w_1 的维度一致，所以是 3。(6)部分用两层 for 循环进行更新参数：第一层 for 循环是针对整个数据集进行循环，故用 epoch 表示；第二层 for 循环是针对 batch 的，用 step 表示。在 with 结构中计算前向传播的预测结果 y，计算损失函数 loss 损失函数 loss，分别对参数 w_1 和参数 b_1 计算偏导数，更新参数 w_1 和参数 b_1 的值，打印出这一轮 epoch 后的损失函数值。因为训练集有 120 组数据，batch 是 32，每个 step 只能喂入 32 组数据，需要 batch 级别循环 4 次，所以 loss 除以 4，求得每次 step 迭代的平均 loss。

希望每个epoch循环后可以显示当前模型的效果，即识别准确率

故在epoch循环中又嵌套了一个batch级别的循环

(7) 计算当前参数前向传播后的准确率，显示当前准确率 acc：

```
for x_test, y_test in test_db:
    y = tf.matmul(h, w) + b        # y 为预测结果
    y = tf.nn.softmax(y)           # y 符合概率分布
    pred = tf.argmax(y, axis=1)    # 返回 y 中最大值的索引即预测的分类
    pred = tf.cast(pred, dtype=y_test.dtype)    # 调整数据类型与标签一致
    correct = tf.cast(tf.equal(pred, y_test), dtype=tf.int32)
    correct = tf.reduce_sum (correct)    # 将每个 batch 的 correct 数加起来
    total_correct += int (correct)       # 将所有 batch 中的 correct 数加起来
```

```
        total_number += x_test.shape [0]
    acc = total_correct / total_number  测试集中数据的总数
    print("test_acc:", acc)
```

(8) acc / loss 可视化：

```
    plt.title('Acc Curve')      #  图片标题
    plt.xlabel('Epoch')       # x 轴名称
    plt.ylabel('Acc')         # y 轴名称
    plt.plot(test_acc, label="$Accuracy$")     #  逐点画出 test_acc 值并连线
    plt.legend()
    plt.show()
```

上述两部分完成了对准确率的计算并可视化准确率与 loss。(7)部分前向传播计算出 *y*，使其符合概率分布并找到最大的概率值对应的索引号，调整数据类型与标签一致，如果预测值和标签相等则 correct 变量自加一，准确率即预测对了的数量除以测试集中的数据总数。(9)部分可将计算出的准确率画成曲线图，通过设置图标题、设置 x 轴名称、设置 y 轴名称，标出每个 epoch 时的准确率并画出曲线，可用同样方法画出 loss 曲线。结果图如图 4.1 与 4.2。

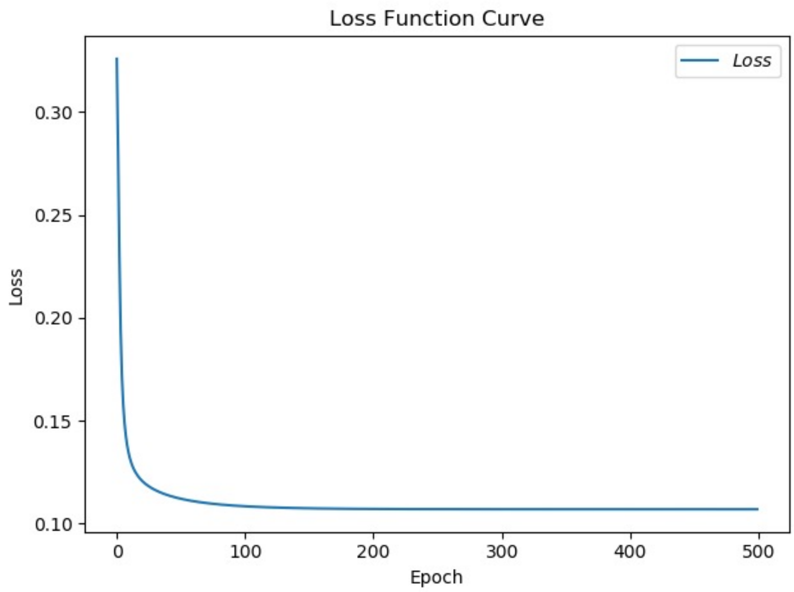

图 4.1 训练过程 loss 曲线

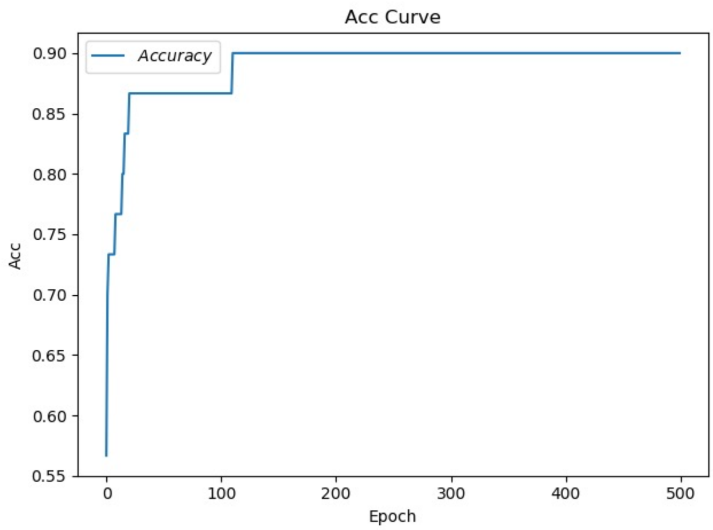

图 4.2 训练过程准确率曲线

5 扩展方法

5.1 本地读取鸢尾花数据集

在这部分我们尝试从本地读取鸢尾花数据集的 txt 文件，并将其输入至神经网络进行训练。鸢尾花数据集的 txt 文件包含内容如图 5.1 所示。

```
"Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
"1" 5.1 3.5 1.4 0.2 "setosa"
"2" 4.9 3 1.4 0.2 "setosa"
"3" 4.7 3.2 1.3 0.2 "setosa"
"4" 4.6 3.1 1.5 0.2 "setosa"
"5" 5 3.6 1.4 0.2 "setosa"
"6" 5.4 3.9 1.7 0.4 "setosa"
"7" 4.6 3.4 1.4 0.3 "setosa"
"8" 5 3.4 1.5 0.2 "setosa"
"9" 4.4 2.9 1.4 0.2 "setosa"
"10" 4.9 3.1 1.5 0.1 "setosa"
"11" 5.4 3.7 1.5 0.2 "setosa"
"12" 4.8 3.4 1.6 0.2 "setosa"
"13" 4.8 3 1.4 0.1 "setosa"
"14" 4.3 3 1.1 0.1 "setosa"
```

图 5.1 鸢尾花数据集 txt 文件内容

读取本地数据集有两种方法：

(1) 利用 pandas 中函数读取，并处理成神经网络需要的数据结构，即利用 pd.read_csv('文件名'，header=第几行作为表头，sep='分割符号')

(2) (2)利用 open 函数打开 txt 文件，并处理成神经网络需要的数据结构，即利用 open('文件名'，'r')。

利用 pandas 中函数读取方法如下：

```
df    = pd.read_csv('iris.txt',header = None,sep=',')   #读取本地文件
data = df.values                                # 去掉索引并取值
x_data = [lines[0:4] for lines in data]         # 取输入特征
x_data = np.array(x_data,float)                 # 转换为 numpy 格式
y_data = [lines[4] for lines in data]           # 取标签
for i in range(len(y_data)):
    if y_data[i] == 'Iris-setosa':
        y_data[i] = 0
    elif y_data[i] == 'Iris-versicolor':
        y_data[i] = 1
    ……
y_data = np.array(y_data)
```

即通过读取本地文件、取特征输入、取标签并将其转换为规定格式，实现本地数据集的读取。

利用 open 函数读取方法如下：

```
f = open('iris.txt','r')            # 取本地文件
contents    = f.readlines()         # 按行读取
i=0
for content in contents:
    temp    = content.split(',')        # 按逗号分隔
    x_data[i] = np.array([temp[0:4]],dtype=float)     # 取输入特征
    if temp[4] ==    'Iris-setosa\n':        # 判断标签并赋值
        y_data[i] = 0
    elif temp[4] == 'Iris-versicolor\n':
        y_data[i] = 1
    ……
    i = i + 1
```

即通过读取本地文件、分割、取输入特征、取标签，实现本地数据集的读取。

5.2 搭建神经网络

数据集较为简单，可利用简单网络结构进行拟合，仅考虑输入层与输出层，构建单层神经网络。参数定义如下：

w1 = tf.Variable(tf.random.truncated_normal[4,3],stddev = 0.1,seed = 1))

b1 = tf.Variable(tf.random.truncated_normal[3],stddev = 0.1,seed = 1))

将学习率设置为 0.5，训练后可发现出现**梯度爆炸**，网络不能有效收敛，训练过程 loss 曲线如图 5.2。

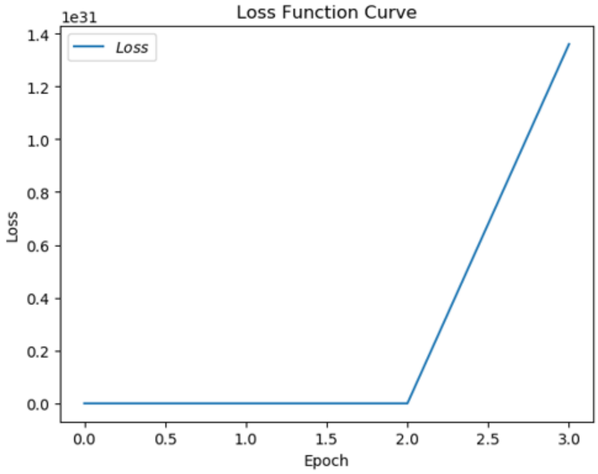

图 5.2 梯度爆炸时 loss 曲线

分析产生梯度爆炸的原因，考虑到使用梯度下降思想时，其计算公式为

$$W_{new} = W_{old} - lr \cdot \frac{\partial L}{\partial W}$$

参数更新量为学习率与损失函数偏导数相乘，二者乘积过大，则会导致梯度爆炸。因此，解决梯度爆炸问题可针对学习率进行调整，也可对数据进行调整。故解决方法可为：(1)逐步减小学习率，0.1、0.01 等；(2)对数据进行**预处理**后再输入神经网络，减小偏差值的大小，抑制梯度爆炸，即数据归一化与标准化，其主要方法有**线性归一化、非线性归一化、Z-Score 标准化**。

线性归一化将数据映射到[0,1]区间中，计算公式如下：

$$x^* = \frac{x - \min\{x\}}{\max\{x\} - \min\{x\}}$$

非线性归一化(log 函数转换)使数据映射到[0,1]区间上，计算公式如下：

$$x^* = \frac{\log_{10} x}{\log_{10} \max\{x\}}$$

Z-Score 标准化使每个特征中的数值平均值变为 0，标准差变为 1，计算公式如下：

$$x^{*}=\frac{x-mean\{x\}}{std\{x\}}$$

以线性归一化为例，其代码实现如下：

```
def normalize(data):
    x_data = data.T    # 每一列为同一属性，转置到每一行
    for i in range(4):
        x_data[i] = (x_data[i] - tf.reduce_min(x_data[i])) /
                    (tf.reduce_max(x_data[i]) - tf.reduce_min(x_data[i]))
     return x_data.T   # 转置回原格式
```

5.3 优化

做完数据标准化，上述网络已经可以跑通，下面对网络进行部分优化，增加指数衰减学习率，指数衰减学习率可在训练初期赋予网络较大学习率，并在训练过程中逐步减小，可有效增加网络收敛速度，其在 tensorflow 中对应函数为 tf.compat.v1.train.exponential_decay(learning_rate_base,global_step,decay_step,decay_rate,staircase =True(False),name)，当 staircase 为 True 时，学习率呈现阶梯状递减。

做完优化后，对网络进行训练。笔者采用 Z-score 标准化后训练 1000 个 eopch，当 staircase =True 时，其 loss、准确率、学习率曲线如图 5.3 所示。

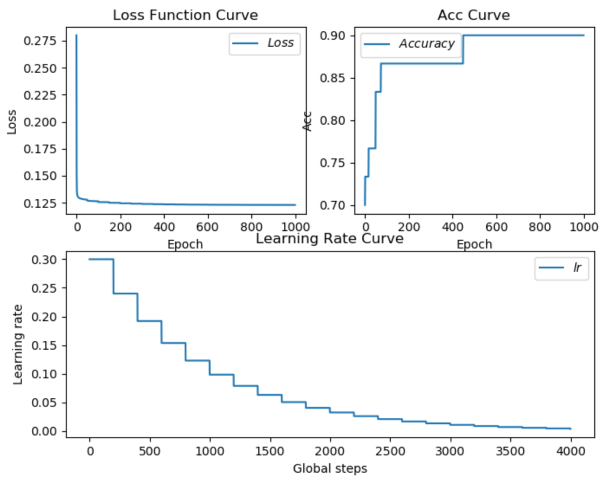

图 5.3 staircase =True 训练过程准确率曲线

当 staircase =False 时，其 loss、准确率、学习率曲线如图 5.4 所示。

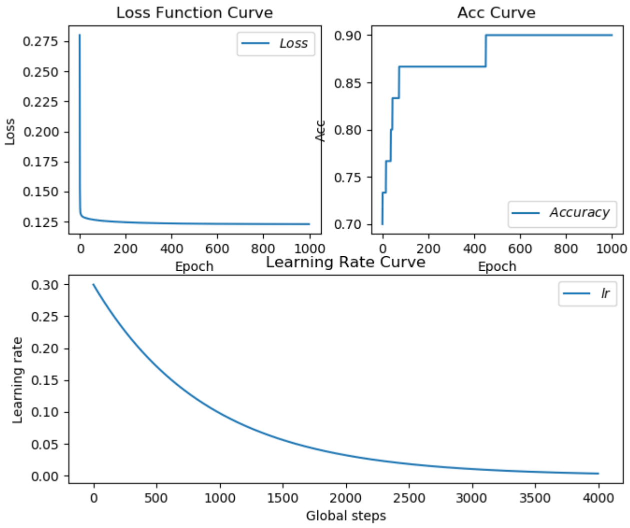

图 5.4 staircase =False 训练过程准确率曲线