第 7 章 深度学习-TensorFlow

经过前面几章的学习，我们已经学会了使用 Python 的基本函数（包括 Numpy 的函数）建立神经网络模型，使用正向传播算法、反向传播算法以及梯度下降法训练神经网络，并且知道如何评估神经网络模型的模型误差以及控制过拟合。但是，当处理复杂问题时，建立的神经网络模型需要更加复杂的结构和多样化的控制过拟合的方法。例如，大量的隐藏层、L_2惩罚、Dropout 方法等。这时，每次建立神经网络模型都从最基础的 Python 语句开始会非常困难，容易出错，而且运行效率低。因此，大型科技公司或者高校研究团队开发了深度学习框架，旨在提高深度学习的研究和应用效率。目前较为流行的深度学习框架有 TensorFlow、PyTorch、Caffe、Theano、MXNet 等。

TensorFlow 是一个开源深度学习框架，由 Google 推出并且维护，拥有最大的用户群体和社区。TensorFlow 的优点包括易用、灵活、高效，并且具有良好的支持。

- 易用性。TensorFlow 提供大量容易理解且可读性强的函数，使得它的工作流程相对容易，兼容性好。例如，TensorFlow 可以很好地与 Numpy 结合，让数据科学家更容易理解和使用。
- 灵活性。TensorFlow 能够在各种类型的设备上运行，从手机到超级计算机。TensorFlow 可以很好地支持分布式计算，可以同时在多个 CPU 或者 GPU 上运行。
- 高效性。TensorFlow 在计算资源要求高的部分采用 C++编写，且开发团队花费了大量时间和努力来改进 TensorFlow 的大部分代码。随着越来越多开发人员的努力，TensorFlow 的运行效率不断提高。
- 良好的支持。TensorFlow 有 Google 团队支持，在声望上有 Google 背书。Google 在自己的日常工作中也使用 TensorFlow，并且持续对其提供支持，在 TensorFlow 周围形成了 一个强大社区。

在本章中，我们将学习使用 TensorFlow 2.0 建立深度学习模型。

7.1 安装 TensorFlow

无论是 Windows、macOS 还是 Linux，都可以使用同样的方式安装 TensorFlow。

- 在 Anaconda 中，进入想要安装 TensorFlow 的环境（例如，进入 py37 环境，`conda activate py37`）。
- 输入 `conda install tensorflow` 即可安装 TensorFlow，如图 7-1 所示。

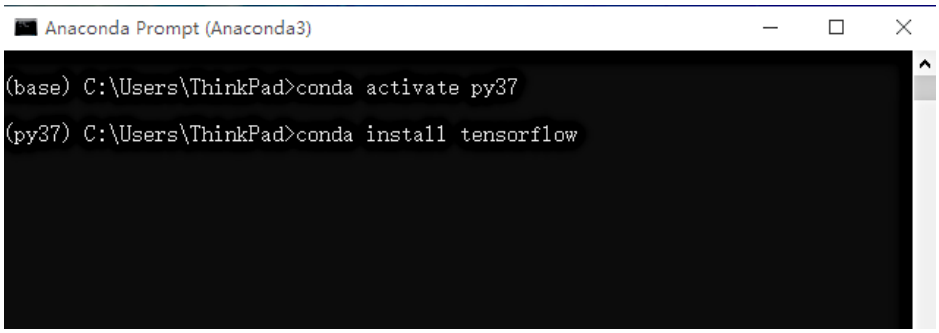

图 7-1 安装 TensorFlow

在一个已安装 TensorFlow 的环境中，打开 Jupyter Notebook，在代码框中输入下面代码，可以输出 TensorFlow 版本号，则说明已经成功安装 TensorFlow。

```
import tensorflow as tf
print(tf.__version__)
2.0.0
```

7.2 TensorFlow 基本用法

7.2.1 Tensor

TensorFlow 的 Tensor 是一个多维数组，类似 Numpy 的数组。但是，Tensor 包含更多信息，包括数据、数据维度和数据类型。例如，下面代码中 `x` 为一个 Tensor。

```
x = tf.add([[1.0, 2.0]], [[3.0, 4.0]])

print(x)
print("x's shape: {}".format(x.shape))
print("x's data type: {}".format(x.dtype))

tf.Tensor([[4. 6.]], shape=(1, 2), dtype=float32)
x's shape: (1, 2)
x's data type: <dtype: 'float32'>
```

TenorFlow 提供了数值运算函数，包括加、减、乘（逐点相乘，矩阵乘法）、乘方等。

- `tf.add(x, y)`，加法，矩阵对应元素相加
- `tf.subtract(x, y)`，减法，矩阵对应元素相减
- `tf.multiply(x, y)`，乘法，矩阵对应元素相乘
- `tf.matmul(x, y)`，矩阵乘法

- `tf.square(x)`，乘方，即x^2
- `tf.reduce_sum(x)`，求x的所有元素和

在下面代码中，分别定义$\boldsymbol{X}$，$\boldsymbol{Y}$，$\boldsymbol{Z}$为

$$\boldsymbol{X}=\begin{pmatrix}1 & 2 & 1\\ 2 & 1 & 1\end{pmatrix},\quad \boldsymbol{Y}=\begin{pmatrix}0\\ 1\\ 1\end{pmatrix},\quad \boldsymbol{Z}=\begin{pmatrix}1 & 1 & 0\\ 1 & 0 & 1\end{pmatrix}$$

然后使用 TensorFlow 的函数计算$\boldsymbol{X}+\boldsymbol{Z}$，$\boldsymbol{X}-\boldsymbol{Z}$，$\boldsymbol{X}\circ\boldsymbol{Z}$，$\boldsymbol{XY}$，$\boldsymbol{X}^2$，$\sum_{i=1}^{3}\boldsymbol{Y}_i$。

```
import numpy as np

x = np.array([[1, 2, 1],[2, 1, 1]])
y = np.array([[0], [1], [1]])
z = np.array([[1, 1, 0], [1, 0, 1]])
w1 = tf.add(x, z)
w2 = tf.subtract(x, z)
w3 = tf.multiply(x, z)
w4 = tf.matmul(x, y)
w5 = tf.square(x)
w6 = tf.reduce_sum(y)

print("w1: \n", w1)
print("w2: \n", w2)
print("w3: \n", w3)
print("w4: \n", w4)
print("w5: \n", w5)
print("w6: \n", w6)

w1:
tf.Tensor(
[[2 3 1]
 [3 1 2]], shape=(2, 3), dtype=int32)
w2:
tf.Tensor(
[[0 1 1]
 [1 1 0]], shape=(2, 3), dtype=int32)
w3:
tf.Tensor(
[[1 2 0]
 [2 0 1]], shape=(2, 3), dtype=int32)
w4:
tf.Tensor(
[[3]
 [2]], shape=(2, 1), dtype=int32)
w5:
tf.Tensor(
[[1 4 1]
 [4 1 1]], shape=(2, 3), dtype=int32)
w6:
tf.Tensor(2, shape=(), dtype=int32)
```

7.2.2 TensorFlow 和 Numpy 的兼容性

TensorFlow 和 Numpy 有很好的兼容性，其主要特性可以总结如下。

- TensorFlow 的运算可以自动把 Numpy 数组转换成 TensorFlow 的 Tensor
- Numpy 的运算可以自动把 TensorFlow 张量转换成 Numpy 数组
- TensorFlow 的 Tensor 可以使用方法.numpy()转换成 Numpy 数组

```
np_array = np.ones([3, 3])

print("TensorFlow 的运算可以自动把 Numpy 数组转换成 TensorFlow 的
Tensor")
tensor = tf.multiply(np_array, 42)
print(tensor)

print("Numpy 的运算可以自动把 TensorFlow 的 Tensor 转换成 Numpy 数组")
print(np.add(tensor, 1))

print("TensorFlow 的 Tensor 可以使用方法.numpy()转换成 Numpy 数组")
print(tensor.numpy())

TensorFlow 的运算可以自动把 Numpy 数组转换成 TensorFlow 的 Tensor
tf.Tensor(
[[42. 42. 42.]
 [42. 42. 42.]
 [42. 42. 42.]], shape=(3, 3), dtype=float64)

Numpy 的运算可以自动把 TensorFlow 的 Tensor 转换成 Numpy 数组
[[43. 43. 43.]
 [43. 43. 43.]
 [43. 43. 43.]]

TensorFlow 的 Tensor 可以使用方法.numpy()转换成 Numpy 数组
[[42. 42. 42.]
 [42. 42. 42.]
 [42. 42. 42.]]
```

7.3 深度神经网络建模基本步骤

在 TensorFlow 2.0 中，建立深度神经网络模型分为 3 个步骤：

- 创建模型结构
- 训练模型
- 模型评估和预测

7.3.1 创建模型结构

在 TensorFlow2.0 中，使用函数 tf.keras.models.Sequential()把隐藏层、输出层等深度学习模型的层结合在一起。函数 tf.keras.models.Sequential()

建立深度神经网络可以有多种方式。这里将以图 7-2 所示的神经网络模型为例介绍两种常用方式。该神经网络模型包含 4 个层，输入层、隐藏层 1、隐藏层 2、输出层，节点数分别为 5、3、3、2，隐藏层激活函数为 ReLU 函数，输出层激活函数为 softmax 函数。图 7-2 中，两层之间的方框表示权重矩阵和截距项，并列出了权重矩阵和截距项的维度。

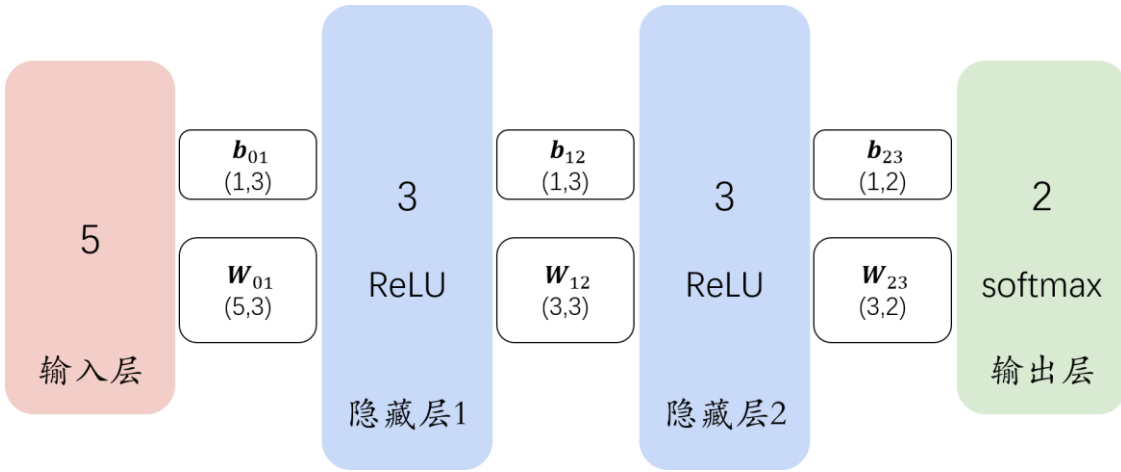

图 7-2 具有两个隐藏层的神经网络

- 第一种使用函数 `tf.keras.models.Sequential()` 建立深度神经模型的方式。

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(3, input_shape=(5,), \
                                activation='relu'))
model.add(tf.keras.layers.Dense(3, activation='relu'))
model.add(tf.keras.layers.Dense(2, activation='softmax'))
```

1. 函数 `tf.keras.models.Sequential()` 建立了一个模型对象 `model`。
2. 在 TensorFlow2.0 中，`tf.keras.layers()` 提供了多种层的类型。其中，函数 `tf.keras.layers.Dense()` 可以创建最常见的神经网络的全连接层（dense layer）。全连接层指该层的所有节点与上一层的所有节点都有连接，即该层的每一个节点都是上一层所有节点的加权和再通过一个激活函数的结果。函数 `tf.keras.layers.Dense()` 的第一个参数表示该层的节点数；参数 `input_shape` 表示输入数据的维度；参数 `activation` 表示激活函数。
3. 函数 `model.add()` 可以把 `tf.keras.layers.Dense()` 创建的层加入到 `model` 中。

上面代码首先创建了两个节点数都为 3，激活函数都是 relu 的隐藏层，然后创建了节点数为 2，激活函数为 sotfmax 的输出层。

- 第二种建立深度神经网络的方式把函数 `tf.keras.layers.Dense()` 创建的层放在一个列表中；然后使用函数

tf.keras.models.Sequential()一次性建立模型 model。两种建立神经网络模型的方式是等价的。

```
model = tf.keras.models.Sequential([ \
tf.keras.layers.Dense(3, input_shape=(5,), activation='relu'),\
tf.keras.layers.Dense(3, activation='relu'),    \
tf.keras.layers.Dense(2, activation='softmax')])
```

这时可以使用函数 model.summary()得到神经网络模型的基本信息。

```
model.summary()

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 3)                 18
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 12
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 8
=================================================================
Total params: 38
Trainable params: 38
Non-trainable params: 0
_________________________________________________________________
```

函数 model.summary()显示了模型的两个隐藏层，以及输出层的信息（函数 model.summary()不显示模型输入层信息）。函数 model.summary()的结果还显示了每一层的节点数，以及得到该层所需要的参数数量。例如，

- 第一个隐藏层的节点数为 3，从输入层到第一个隐藏层的参数个数为$5 \times 3 + 3 = 18$（输入层的节点数为 5），5×3表示 5 个输入层节点，连接 3 个该层节点所需的权重的元素个数；在$5 \times 3 + 3$中，加号后面的3表示截距项个数，该层有 3 个节点，所以截距项个数为 3。
- 第二个隐藏层的节点数也为 3，从第一个隐藏层到第二个隐藏层的参数个数为$3 \times 3 + 3 = 12$。
- 输出层的节点数为2，从第二个隐藏层到输出层的参数个数为$3 \times 2 + 2 = 8$。

最后，模型 model 总的参数个数为 $18 + 12 + 8 = 38$。通过 model.trainable_variables 可以看到参数的具体数值。例如，

```
model.trainable_variables

[<tf.Variable 'dense_3/kernel:0' shape=(5, 3) dtype=float32,
numpy=
 array([[ 0.58498806, -0.83305913, -0.8193064 ],
        [ 0.8531212 , -0.3258524 ,  0.13195276],
```

```
        [ 0.51721174, -0.7353755 , -0.3585236 ],
        [-0.6890211 , -0.72342956, -0.32241583],
        [-0.28896993,  0.76220506,  0.6864143 ]],
dtype=float32)>,
 <tf.Variable 'dense_3/bias:0' shape=(3,) dtype=float32,
numpy=array([0., 0., 0.], dtype=float32)>,
 <tf.Variable 'dense_4/kernel:0' shape=(3, 3) dtype=float32,
numpy=
 array([[ 0.42070603,  0.52094555,  0.22916198],
        [-0.13093519, -0.77951264,  0.13630629],
        [-0.54199314, -0.00618839, -0.7810683 ]],
dtype=float32)>,
 <tf.Variable 'dense_4/bias:0' shape=(3,) dtype=float32,
numpy=array([0., 0., 0.], dtype=float32)>,
 <tf.Variable 'dense_5/kernel:0' shape=(3, 2) dtype=float32,
numpy=
 array([[ 0.23387682,  0.19419336],
        [ 0.7517618 ,  1.0273001 ],
        [-0.75748616, -0.7398752 ]], dtype=float32)>,
 <tf.Variable 'dense_5/bias:0' shape=(2,) dtype=float32,
numpy=array([0., 0.], dtype=float32)>]

len(model.trainable_variables)
6
```

`model.trainable_variables` 返回了长度为 6 的列表，分别为隐藏层 1 的权重矩阵（维度为5×3）和截距项向量（维度为1×3）；隐藏层 2 的权重矩阵（维度为3×3）和截距项向量（维度为1×3）；输出层的权重矩阵（维度为3×2）和截距项向量（维度为1×2）。现在，神经网络模型还没有开始训练，`model.trainable_variables` 返回了 TensorFlow 自动给予这些参数的初始值。

- 默认的权重矩阵初始值为方法 glorot_uniform 得到的随机数。glorot_uniform 方法从$[-\text{limit}, \text{limit}]$的均匀分布中产生随机数，其中

$$\text{limit} = \sqrt{\frac{6}{\text{row\_num} + \text{col\_num}}}$$

- row_num 和 col_num 分别是权重矩阵的行和列的数量。可以看出，当权重矩阵维度比较大时，默认权重初始值将会在更小的范围随机产生。根据经验，这样产生的权重初始值可以表现得更好。
- 默认截距项初始值都为 0。

如果需要，可以通过函数 `tf.keras.layers.Dense()` 的参数 `kernel_initializer` 设置产生权重初始值的方法，通过参数 `bias_initializer` 设置产生截距项初始值的方法。

7.3.2 训练模型

现在开始设置模型训练的最优化方法、损失函数和模型评价指标。函数 model.compile()可以实现这些设置。例如，

```
model.compile(optimizer=tf.keras.optimizers.SGD(),
              loss='mse',
              metrics=['mse'])
```

上面代码使用函数 tf.keras.optimizers.SGD()实现最优化，SGD 表示随机梯度下降法（Stochastic Gradient Descent，SGD）；使用 mse 作为损失函数；同时使用 mse 评价模型。

至此，model 已经具备建立神经网络模型的全部要素：

- 模型结构
- 损失函数
- 最优化方法

接着可以使用函数 model.fit()训练模型。下面代码从均匀分布中随机产生 train_x，train_y，test_x，test_y，然后把 train_x、train_y 代入到函数 model.fit()中训练模型。训练模型时，设置迭代次数为 epochs=3，每个批量的数据为 batch_size=100。可以看到，函数 model.fit()计算过程中会动态显示每次迭代中每个观测点训练时间，每次迭代后损失函数和评价指标的值。

```
train_x = np.random.random((1000, 5))
train_y = np.random.random((1000, 2))
test_x = np.random.random((200, 5))
test_y = np.random.random((200, 2))

history = model.fit(train_x, train_y, epochs=3, batch_size=100)

Train on 1000 samples
Epoch 1/3
1000/1000 [====] - 0s 55us/sample - loss: 0.0843 - mse: 0.0843
Epoch 2/3
1000/1000 [====] - 0s 11us/sample - loss: 0.0843 - mse: 0.0843
Epoch 3/3
1000/1000 [====] - 0s 11us/sample - loss: 0.0843 - mse: 0.0843
```

7.3.3 模型评估和预测

最后可以通过函数 model.evaluate()得到预测误差。从下面的运算结果可以看到，该神经网络模型的损失函数的值和测试误差都约为0.08（在该模型中，损失函数和模型评价指标都是均方误差，因此两者相等）。

```
model.evaluate(test_x, test_y)
```

```
200/200 [====] - 0s 160us/sample - loss: 0.0827 - mse: 0.0827

[0.08271521747112275, 0.08271521]
```

此外可以使用函数 model.predict()得到观测点的预测值。例如，下面代码随机产生一个观测点，然后使用函数 model.predict()计算该观测点的预测值。

```
one_obs = np.random.random((1, 5))
one_obs

array([[0.62068234, 0.31588733, 0.47846199, 0.58297638, \
        0.37947331]])
model.predict(one_obs)

array([[0.48976234, 0.5102377 ]], dtype=float32)
```

7.4 基于 TensorFlow 建立线性回归模型

本节将以广告数据为例，通过建立线性回归模型一步一步学习 TensorFlow 的一般建模过程。在第 3 章中，我们曾使用过该数据。数据包含了某个商品在 200 个市场的广告投入和销量。广告投入分为 3 种不同形式，分别是 TV、Radio、Newspaper（广告投入的单位是千美元（thousands of dollars）。数据的因变量为销量（商品销量的单位是 thousands of units）。在这个例子中，我们关心各种广告投入对商品销量的影响。线性回归模型为

$$y = b + w_1 x_1 + w_2 x_2 + w_3 x_3 + \epsilon$$

➢ 预处理数据

下面的代码定义了函数 loadDataSet()，该函数可以读入 data 文件夹的数据 Advertising.csv 。 for 循环对数据进行逐行处理。line.strip().split(',')可以去掉换行符“\n”，同时根据逗号“，”把数据分隔开。接着将数据转换成浮点型，每一行的前 3 个数字放入列表 x 中，最后一个数字放入列表 y 中。最后把列表 x,y 转换成 Numpy 数组，并把 y 的维度变成 200×1。

```
def loadDataSet():
    x = []; y = []

    # 打开 data 文件中的文件 Advertising.csv
    f = open("./data/Advertising.csv")

    # 函数`readlines`读入文件 f 的所有行
    for line in f.readlines()[1:]:
        lineArr = line.strip().split(',')
        x.append([float(lineArr[0]), float(lineArr[1]),
```

```
float(lineArr[2])])
        y.append(float(lineArr[3]))
    return np.array(x), np.array(y).reshape(-1, 1)
x, y = loadDataSet()

print("shape of x:")
print(x.shape)
print("shape of y:")
print(y.shape)

shape of x:
(200, 3)
shape of y:
(200, 1)
```

我们把数据分成两个部分，训练数据和测试数据。训练数据包含 100 个观测点，测试数据包含 100 个观测点。

```
np.random.seed(1)
train_x, train_y = x[:100], y[:100]
test_x, test_y = x[-100:], y[-100:]
```

接着把因变量 y 中心化，centered_y $= y - \overline{y}$，把自变量x标准化scaled_x $= \frac{x-\overline{x}}{\mathrm{sd}(x)}$，使得centered_y的均值为 0，scaled_x的均值为 0，方差为 1。

```
mean_x = np.mean(train_x, axis = 0, keepdims=True)
sd_x = np.std(train_x, axis=0, keepdims=True)
mean_y = np.mean(train_y)

train_scaled_x = (train_x - mean_x)/sd_x
train_centered_y = train_y - mean_y

test_scaled_x = (test_x - mean_x)/sd_x
test_centered_y = test_y - mean_y
```

通过上面的步骤，数据已经完成预处理。接下来将正式使用 TensorFlow 建立线性回归模型。线性回归模型相当于图 7-3 所示的简单神经网络模型，该模型没有隐藏层，输出层只有一个节点且激活函数为线性函数。

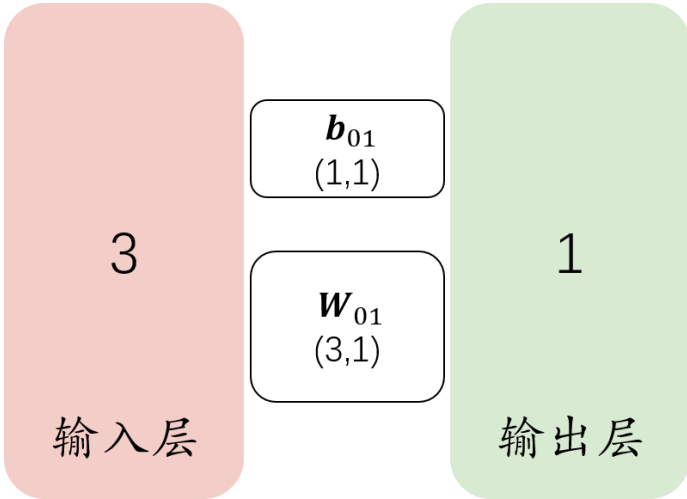

图 7-3 线性回归模型

- 创建模型结构

线性回归模型没有隐藏层，输出层的节点数为 1，可以通过如下代码使用函数 tf.keras.models.Sequential()构建模型。

```
model = tf.keras.models.Sequential([ \
                    tf.keras.layers.Dense(1,  input_shape=(3,))])
```

- 训练模型

接着使用函数 model.compile()设置优化方法、损失函数和评价指标；然后使用函数 model.fit()代入训练数据训练模型。在函数 model.fit()中，测试数据 test_scaled_x，test_centered_y 通过参数 validation_data 代入函数中；这样，每一次迭代都可以得到一个测试误差。函数 model.fit()的运行过程都记录在 lm_history 中。

```
model.compile(optimizer=tf.keras.optimizers.SGD(0.1), \
                    loss='mse', metrics=['mse'] )
lm_history = model.fit(train_scaled_x, train_centered_y, \
     epochs=50,  verbose=0, batch_size=100,      \
     validation_data=(test_scaled_x, test_centered_y))
```

lm_history 保存了诸多信息，包括 lm_history.epoch 和 lm_history.history。lm_history.history 包含了每一次循环的模型损失函数值（损失函数的值即为训练误差）和评价指标的值（评价指标的值即为测试误差）。图 7-4 画出了训练误差和测试误差随着每一次循环的变化情况。

```
%config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False   #用来正常显示负号
```

```
plt.plot(lm_history.epoch, lm_history.history['mse'], \
                                                label="训练误差")
plt.plot(lm_history.epoch, lm_history.history['val_mse'], \
                                                label="测试误差")

plt.xlabel("迭代次数", fontsize=16)
plt.ylabel("模型误差", fontsize=16)
plt.legend(fontsize=16)
plt.show()
```

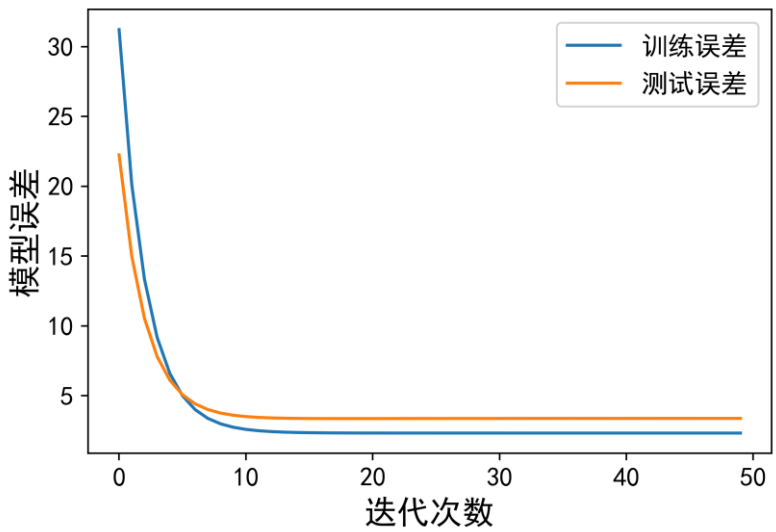

图 7-4 线性回归模型训练误差和测试误差变化曲线

通过 model.trainable_variables 可以查看线性模型的权重和截距项。

```
print("Weights:\n{}".format(model.trainable_variables[0].numpy(
)))
print("Bias:\n{}".format(model.trainable_variables[1].numpy()))

Weights:
[[ 3.7997427 ]
 [ 2.769601  ]
 [-0.23214102]]
Bias:
[-5.6775757e-08]
```

7.5 基于 TensorFlow 建立神经网络分类模型

我们以 mnist 手写数据为例，使用 TensorFlow 建立具有两个隐藏层的神经网络模型。通过这个例子，我们学习使用 TensorFlow 建立分类模型的方法，并且进一步学习 TensorFlow 的一些常用函数。

7.5.1 神经网络分类模型

我们将建立如图 7-5 所示的神经网络模型。该模型包含两个隐藏层，节点数分别为 1024 和 512。

- 从输入层到隐藏层 1 的权重矩阵为784×1024（输入层的节点数乘以隐藏层 1 的节点数），截距项为1×1024（截距项为一个行向量，元素个数为隐藏层 1 节点数）；
- 从隐藏层 1 到隐藏层 2 的权重矩阵为1024×512（隐藏层 1 的节点数乘以隐藏层 2 的节点数），截距项为1×512（截距项为一个行向量，元素个数为隐藏层 2 节点数）。
- 从隐藏层 2 到输出层的权重矩阵为512×10（隐藏层 2 的节点数乘以输出层的节点数），截距项为1×10（截距项为一个行向量，元素个数为输出层节点数）。

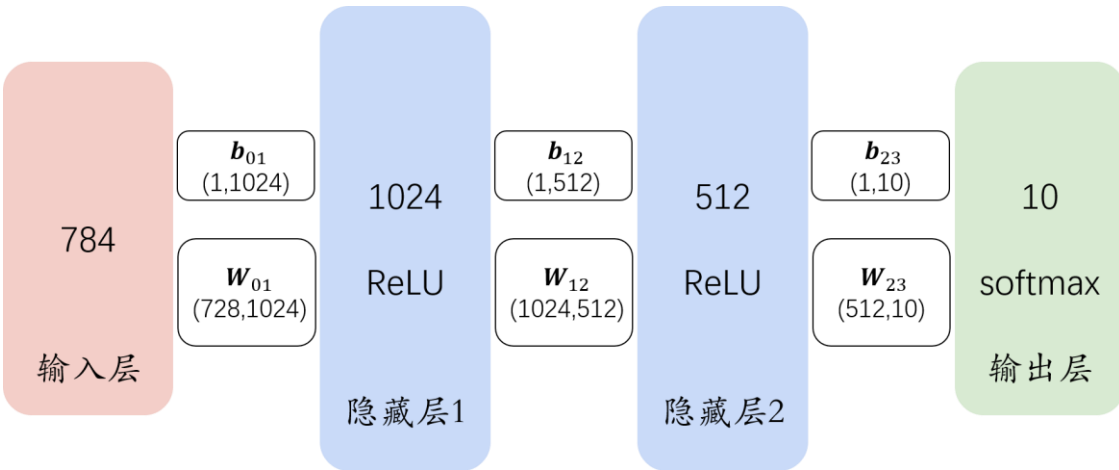

图 7-5 具有两个隐藏层的神经网络，隐藏层 1 的节点数为 1024，隐藏层 2 的节点数为 512

下面的代码载入了建模所需要的 Python 包，并且载入 mnist 数据。其中，训练数据 `x_train` 和 `y_train` 包含了 60 000 幅图片的信息，测试数据 `x_test` 和 `y_test` 包含了 10 000 幅图片的信息。

```
"""
载入 mnist 数据
"""
import idx2numpy

train_images = idx2numpy.convert_from_file(  \
                        './data/mnist/train-images.idx3-ubyte')
train_labels  = idx2numpy.convert_from_file( \
                        './data/mnist/train-labels.idx1-ubyte')
test_images = idx2numpy.convert_from_file(   \
                        './data/mnist/t10k-images.idx3-ubyte')
```

```
test_labels = idx2numpy.convert_from_file(    \
                        './data/mnist/t10k-labels.idx1-ubyte')
```

接着把所有像素点的值除以 255，以使得自变量矩阵的元素的值都处在[0,1]之间。为了方便选择超参数，训练数据进一步随机分成两部分：训练数据 train_images 和 train_labels（训练数据包含 50 000 幅图片的信息），验证数据 valid_images 和 valid_labels（验证数据包含 10 000 幅图片的信息）。

```
"""
输入数据除以 255
"""
np.random.seed(1)
train_images, test_images = train_images/255, test_images/255

"""
把训练数据(60000*28*28)分成训练数据(50000*28*28)和验证数据
(10000*28*28)
"""
index = np.arange(len(train_images))
np.random.shuffle(index)

valid_images, valid_labels = train_images[index[-10000:]], \
                        train_labels[index[-10000:]]   # 验证数据
train_images, train_labels = train_images[index[:50000]], \
                        train_labels[index[:50000]]    # 训练数据
```

然后使用 tf.keras.models.Sequential()构建模型。因为 mnist 手写数据每一幅图片用 28×28 的灰度值表示，我们需要使用函数 tf.keras.layers.Flatten()把每一幅图片的维度变为784。

```
mnist_model = tf.keras.models.Sequential([     \
          tf.keras.layers.Flatten(input_shape=(28, 28)), \
          tf.keras.layers.Dense(1024, activation='relu'), \
          tf.keras.layers.Dense(512, activation='relu'),  \
          tf.keras.layers.Dense(10, activation='softmax')])
```

下面通过 mnist_model.summary()查看每一层的节点数和参数个数。可以看到，Flatten 的维度为(None,784)，没有参数，在这里，None 表示训练数据每一批数据的观测点个数，不需要指定。该模型总的参数个数超过百万，具体为1333770。

```
mnist_model.summary()
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_3 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_16 (Dense)             (None, 1024)              803840
_________________________________________________________________
dense_17 (Dense)             (None, 512)               524800
```

```
_________________________________________________________________
dense_18 (Dense)             (None, 10)                5130
=================================================================
Total params: 1,333,770
Trainable params: 1,333,770
Non-trainable params: 0
_________________________________________________________________
```

现在使用函数 model.compile()设置 mnist_model 的优化方法、损失函数和评价指标。

- 我们使用 Adam 优化方法 tf.keras.optimizers.Adam()，Adam 方法是 2014 年提出的一个改进版的梯度下降法。
- 损失函数选用 sparse_categorical_crossentropy，因为这是一个多分类问题，且因变量没有 one-hot 编码，保持整数形式。
- 模型评价方法采用预测准确率 accuracy。对于分类问题，这是一个很直观的指标。

```
mnist_model.compile(optimizer=tf.keras.optimizers.Adam(), \
                    loss='sparse_categorical_crossentropy', \
                    metrics=['accuracy'])
```

最后使用函数 mnist_model.fit()训练模型，并用函数 mnist_model.evaluate()计算测试准确率。

```
mnist_model_history = mnist_model.fit(train_images,
    train_labels, epochs=20, verbose=0,  \
    validation_data=(valid_images,valid_labels), batch_size=128)
mnist_model.evaluate(test_images, test_labels)

10000/10000 [====] - 1s 71us/sample - loss: 0.1123 - accuracy:
0.9799

[0.11226291974880187, 0.9799]
```

从上面结果可以看出，测试准确率大约为98%。图 7-6 画出预测准确率随着每一次迭代的变化情况。从图 7-6 可以看出，训练准确率一直在增长，到最后几乎等于100%；验证准确率在 2，3 次循环之后就不再增长，稳定在略低于98%的水平。

```
plt.plot(mnist_model_history.epoch,  \
          mnist_model_history.history['accuracy'],  \
          label="训练准确率")
plt.plot(mnist_model_history.epoch, \
         mnist_model_history.history['val_accuracy'], \
         label="验证准确率")

plt.xlabel("迭代次数", fontsize=16)
plt.ylabel("预测准确率", fontsize=16)
_ = plt.legend(fontsize=16)
```

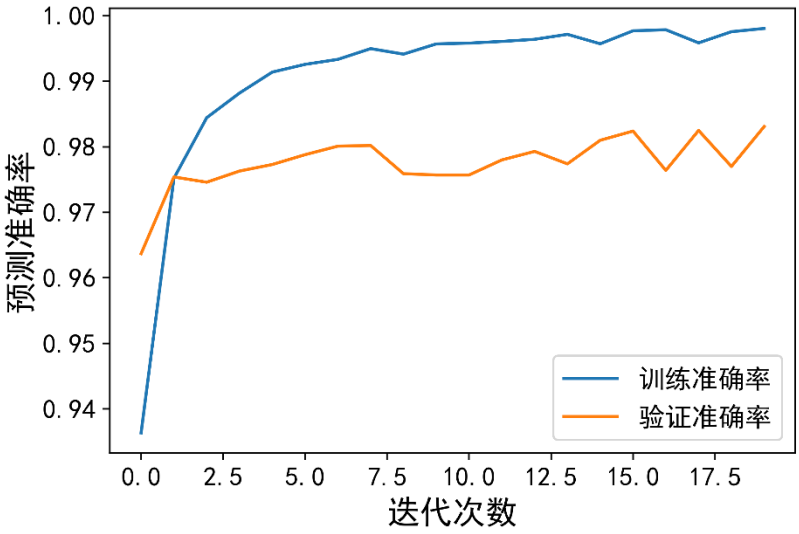

图 7-6 训练准确率和测试准确率随着训练过程的变化曲线

7.5.2 神经网络模型的正则化

从图 7-6 可以看出，训练准确率在增大，验证准确率不变，训练准确率和验证准确率的差值增大。可以怀疑 7.5.1 节的模型过拟合。在本节中，我们尝试两种正则化方法控制过拟合：L_2正则化和 Dropout。

- L_2惩罚法。使用 `tf.keras.models.Sequential()`建立模型时，只需要设置函数 `tf.keras.layers.Dense()`的参数 `kernel_regularizer` 为函数 `tf.keras.regularizers.l2()`便可以实现对该层的L_2正则化。函数 `tf.keras.regularizers.l2(0.001)`表示L_2正则化的参数$\lambda = 0.001$。其余建模过程和没有正则化的情况完全一样。

```
mnist_l2_model = tf.keras.models.Sequential([\
       tf.keras.layers.Flatten(input_shape=(28, 28)), \
       tf.keras.layers.Dense(1024, activation='relu', \
          kernel_regularizer=tf.keras.regularizers.l2(1e-4)), \
       tf.keras.layers.Dense(512, activation='relu',
          kernel_regularizer=tf.keras.regularizers.l2(1e-4)), \
       tf.keras.layers.Dense(10, activation='softmax')])

mnist_l2_model.compile(optimizer=tf.keras.optimizers.Adam(), \
                       loss='sparse_categorical_crossentropy',\
                       metrics=['accuracy'])
mnist_l2_model_history = mnist_l2_model.fit(train_images, \
                     train_labels, epochs=20, verbose=0,
                     validation_data=(valid_images,valid_labels),
                     batch_size=128)

mnist_l2_model.evaluate(test_images, test_labels)
10000/10000 [====] - 1s 108us/sample - loss: 0.1457 - accuracy:
```

```
0.9750

[0.14569806067943572, 0.975]
```

- Dropout。使用 `tf.keras.models.Sequential()` 建立模型时，只需要在函数 `tf.keras.layers.Dense()` 的后面使用函数 `tf.keras.layers.Dropout()`，便可以实现对该层的 dropout。函数 `tf.keras.layers.Dropout(0.5)` 表示失活节点的比例为50%。其余建模过程和没有正则化的情况完全一致。

```
mnist_dropout_model = tf.keras.models.Sequential([
              tf.keras.layers.Flatten(input_shape=(28, 28)),  \
              tf.keras.layers.Dense(1024, activation='relu'),  \
              tf.keras.layers.Dropout(0.5),   \
              tf.keras.layers.Dense(512, activation='relu'),  \
              tf.keras.layers.Dropout(0.5),   \
              tf.keras.layers.Dense(10, activation='softmax')])
mnist_dropout_model.compile(optimizer= \
              tf.keras.optimizers.Adam(),\
              loss='sparse_categorical_crossentropy',\
              metrics=['accuracy'])
mnist_dropout_model_history = mnist_dropout_model.fit( \
            train_images, train_labels, epochs=20, verbose=0,  \
            validation_data=(valid_images,valid_labels),  \
            batch_size=128)
mnist_dropout_model.evaluate(test_images, test_labels)
10000/10000 [====] - 1s 71us/sample - loss: 0.0676 - accuracy:
0.9826


[0.06755764851501081, 0.9826]
```

图 7-7 画出了 3 个模型训练准确率，验证准确率随着每一次迭代的变化情况。实线为验证准确率，虚线为训练准确率。可以看到，L_2正则化和 Dropout 都使得过拟合推迟了一些。总的来说，Dropout 的效果更好一些，使用 Dropout 的模型的测试准确率达到了98.3%。

```
def plot_history(histories, key='accuracy'):
    plt.figure(figsize=(10,7))

    for name, history in histories:
        val = plt.plot(history.epoch, \
                       history.history['val_'+key], \
                       '--', label= '验证准确率'+name.title())
        plt.plot(history.epoch, history.history[key], \
                 color=val[0].get_color(),   \
                 label= '训练准确率'+name.title())

    plt.xlabel('迭代次数', fontsize=16)
    plt.ylabel('预测准确率', fontsize=16)
    plt.legend(fontsize=16)

    plt.xlim([0,max(history.epoch)])
```

```
plot_history([('（无正则化）', mnist_model_history),\
              ('（L2 惩罚）', mnist_l2_model_history), \
              ('（Dropout）', mnist_dropout_model_history)])
```

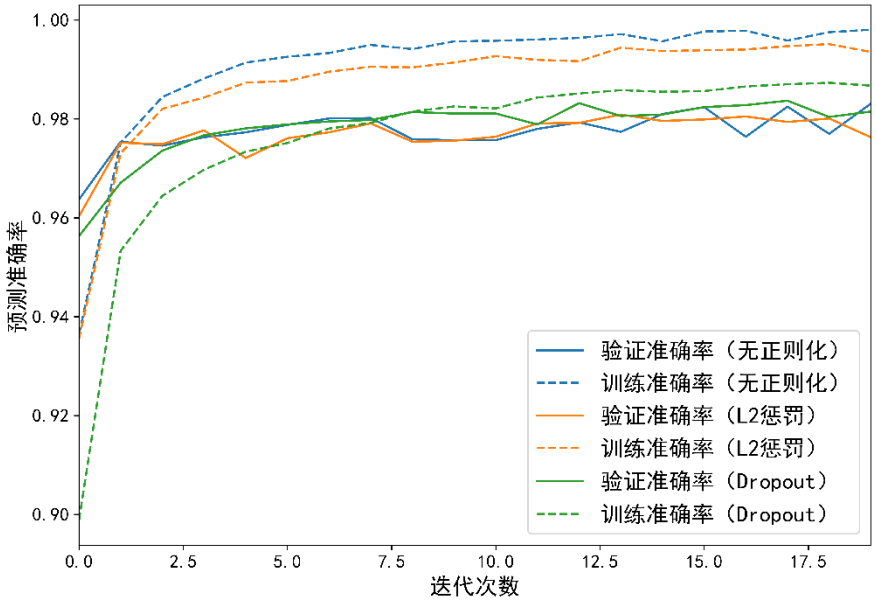

图 7-6 训练准确率和验证准确率的随着训练过程的变化曲线，L_2惩罚和 Dropout 控制过拟合。

7.6 本章小结

TensorFlow 是现在最流行，最强大的深度学习框架。在本章中，我们学习了用 TensorFlow 建立深度学习模型的方法。TensorFlow2.0 建立深度学习模型通常包含 3 个部分。

- 创建模型结构，主要使用的函数有 `model=tf.keras.models.Sequential()`，`tf.keras.layers.Dense()`。
- 训练模型，主要使用的函数有 `model.compile(optimizer, loss, metrics)`，`model.fit()`。
- 模型评估和预测，主要使用的函数有 `model.evaluate()`，`model.predict()`。

习题

1. 分析 Default 数据，使用 TensorFlow 建立 logistic 模型。
2. 分析 mnist 数据，使用 TensorFlow 建立具有 3 个隐藏层的神经网络模型，尝试用不同的正则化方法控制过拟合。

3. 分析 mnist 数据，使用 TensorFlow 建立神经网络模型，尝试不同的隐藏层个数、隐藏层节点数和正则化方法。在建模过程中，把数据分成训练数据、验证数据和测试数据，使用训练数据建立神经网络模型，并且使用验证数据计算验证误差，选择合适的隐藏层个数、隐藏层的节点数以及正则化方法。你的神经网络模型测试正确率最高可以达到多少？
4. 分析 Fashion-mnist 数据，使用 TensorFlow 建立神经网络模型，尝试不同的隐藏层个数、隐藏层节点数和正则化方法。在建模过程中，把数据分成训练数据、验证数据和测试数据，使用训练数据建立神经网络模型，并且使用验证数据计算验证误差，选择合适的隐藏层个数、隐藏层的节点数以及正则化方法。你的神经网络模型测试正确率是多少？
5. 分析 mnist 数据，使用 TensorFlow 建立没有隐藏层的神经网络模型，得到权重矩阵，把权重矩阵的每一列都变成28×28的矩阵，然后画出图形，观察图形特征，思考权重在神经网络中的意义。