人工智能基础

人工智能是研究使用计算机来模拟人的某些思维过程和智能行为（如学习、推理、思考、规划等）的学科，主要包括计算机实现智能的原理、制造类似于人脑智能的计算机，使计算机能实现更高层次的应用。

简单来说，人工智能其实就是一个机器，它可以接收输入并给出输出。

中文房间

我们都知道，外国人通常认为中文是一门很难学习的语言。1980年，美国学者约翰·塞尔提出了中文房间，这有助于我们理解人工智能。

假设有一个房间，里面有一个不会说中文的外国人。房间与外界封闭，只有一个小口用于接收或发送消息。房间里提供了一个中文对照手册。另一个人在房间外向房间内发送中文，而房间里的人需要根据对照手册，向房间外的人回应。

假设它的对照手册是这样的：

输入	输出
你好	你好
你叫什么	我叫中文房间
……	……

那么，如果房间外的人向房间内发送消息“你好”，房间内的人就会向房间外的人回应“你好”，如图所示：

但是，房间外的人看房间，却什么也看不到，因为房间几乎是完全封闭的。

当然，现在的人工智能通常不采用对照手册输出数据，而是通过数学方法计算得到。

训练

人工智能的训练过程就是计算方法的生成过程。训练过程中，需要给出输入和对应的期待输出，机器通过数学方法调整参数，尽可能降低实际输出与期待输出的损失，达到训练的目的。

测试

为了防止模型在训练时过于关注训练数据的特征，而无法应用在训练数据以外的数据上（通常称这种情况为过拟合），在模型训练完毕后，通常需要用不同于训练数据的数据进行测试。

paddle框架的安装

在PyCharm的下面点击Terminal，进入终端界面，依次输入以下命令：

```
cd .\venv\Scripts\
./pip install paddlepaddle-gpu
```

普通数据处理

本节我们将以波士顿房价预测为例，展示如何使用Paddle框架进行普通数据处理。

数据集

该数据集的下载地址为archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data，打开后右击另存为，下载即可。

这个数据集共有506条数据，每个数据有13个输入和1个输出（房价），13个输入如下表所示：

字段名	类型	含义
CRIM	float	该镇的人均犯罪率
ZN	float	占地面积超过25,000平方呎的住宅用地比例
INDUS	float	非零售商业用地比例
CHAS	int	是否邻近 Charles River 1=邻近；0=不邻近
NOX	float	一氧化氮浓度
RM	float	每栋房屋的平均客房数
AGE	float	1940年之前建成的自用单位比例
DIS	float	到波士顿5个就业中心的加权距离
RAD	int	到径向公路的可达性指数
TAX	int	全值财产税率
PTRATIO	float	学生与教师的比例
B	float	1000(BK-0.63)^2，其中BK是城镇中黑人的比例
LSTAT	float	低收入人群占比

最后一个字段就是输出，也就是房价了。

接下来我们把这个数据放在项目文件夹下，先编写代码，查看以下数据情况：

```
import numpy as np  # 用于处理数据的模块，paddle自带

dataset = np.fromfile('housing.data.txt', sep=' ')  # 从文件中读取数据，sep：分隔符
print(dataset)
print(dataset.shape)  # 输出数据的形状
```

输出：

```
[6.320e-03 1.800e+01 2.310e+00 ... 3.969e+02 7.880e+00 1.190e+01]
(7084,)
```

第一行就是部分数据了，第二行指的是数据的形状，这里是一个长为7084的一维数据，也就是一条数据。但是，我们需要它是一个14列506行的二维表格，因此，我们可以使用np.reshape函数变换形状：

```
import numpy as np  # 用于处理数据的模块，paddle自带

dataset = np.fromfile('housing.data.txt', sep=' ')  # 从文件中读取数据，sep：分隔符
dataset = dataset.reshape((506, 14))
print(dataset)
print(dataset.shape)  # 输出数据的形状
```

这样它就成功变成了二维数据了，因此输出：

```
[[6.3200e-03 1.8000e+01 2.3100e+00 ... 3.9690e+02 4.9800e+00 2.4000e+01]
 [2.7310e-02 0.0000e+00 7.0700e+00 ... 3.9690e+02 9.1400e+00 2.1600e+01]
 [2.7290e-02 0.0000e+00 7.0700e+00 ... 3.9283e+02 4.0300e+00 3.4700e+01]
 ...
 [6.0760e-02 0.0000e+00 1.1930e+01 ... 3.9690e+02 5.6400e+00 2.3900e+01]
 [1.0959e-01 0.0000e+00 1.1930e+01 ... 3.9345e+02 6.4800e+00 2.2000e+01]
 [4.7410e-02 0.0000e+00 1.1930e+01 ... 3.9690e+02 7.8800e+00 1.1900e+01]]
(506, 14)
```

其实，dataset是由numpy模块实现的一个更快的列表，我们一般称之为数组。这样的二维数据称之为二维数组。

在paddle中，自定义数据集需要创建一个继承自 `paddle.io.Dataset` 类的子类，然后必须重写 `__init__` 、 `__getitem__` 和 `__len__` 方法：

```
class HouseDataset(paddle.io.Dataset):
    def __init__(self):
        super().__init__()

    def __getitem__(self, idx):
        pass

    def __len__(self):
        pass
```

接下来我们创建这样的子类，读取数据集：

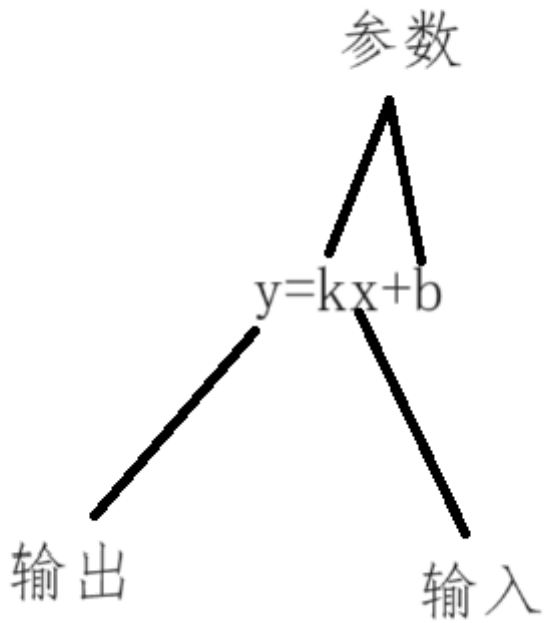
```
# 文件名：dataset.py
import paddle
import numpy as np


class HouseDataset(paddle.io.Dataset):
    def __init__(self, mode):
        super().__init__()
        dataset = np.fromfile('housing.data.txt', sep=' ', dtype=np.float32)
        self.dataset = dataset.reshape((506, 14))
        # 划分训练集和测试集
        if mode == 'train':
            self.dataset = self.dataset[: int(506*0.8)]
        else:
            self.dataset = self.dataset[int(506*0.8): ]

    def __getitem__(self, idx):
        return self.dataset[idx][:-1], self.dataset[idx][-1]  # 返回值是元组形式，第一个数据是特征(X)，第二个数据是预测值(Y)

    def __len__(self):
        return len(self.dataset)  # 获取数据长度（数据有几条）
```

在数据集子类中，我们编写了划分训练集和测试集的代码，用于以后的训练和测试。另外，在读取数据时，我们设置数据类型为float32，以便paddle使用。

模型训练

模型

在paddle中，机器需要先知道计算方法，再通过训练调节参数。模型就是告诉paddle应该采用什么样的计算方法，如下图所示：



在y=kx+b中，y是输出，x是输入，k和b是参数，而y=kx+b本身就是模型。
在paddle中，可以使用paddle自带的模型，也可以自定义模型。由于paddle自带模型多是关于图像处理的，因此我们采用自定义模型。
自定义模型采用paddle.nn.Sequential类，通过该类创建一个对象，这个对象就是模型：

```
layers = paddle.nn.Sequential(
    paddle.nn.Linear(13, 30),  # 全连接层
    paddle.nn.ReLU(),
    paddle.nn.Linear(30, 1)
)
```

这里的ReLU是激活函数，它用于防止模型线性，也就是防止模型停留在y=kx+b上，允许模型拟合更广泛的数据。
ReLU(x)定义为：当x<0时，ReLU(x) = 0，当x≥0时，ReLU(x) = x.

训练

Paddle的训练非常简单，只需要几行代码即可实现：

```
import paddle
import dataset

ds_train = dataset.HouseDataset('train')
ds_test = dataset.HouseDataset('test')

# 创建模型
net = paddle.nn.Sequential(
    paddle.nn.Linear(13, 30),  # 全连接层
    paddle.nn.ReLU(),
    paddle.nn.Linear(30, 1)
)

# 将模型包装，以使用Paddle高级API
model = paddle.Model(net)

# 设置模型的优化器和损失函数
model.prepare(paddle.optimizer.Adam(parameters=model.parameters()),  # Adam优化器
              paddle.nn.MSELoss())  # 均方误差损失函数

# 模型训练
# epochs: 训练过程中完全训练多少次训练集
# batch_size: 训练过程中，将多少条数据同时训练
# verbose: 设置日志打印格式
model.fit(train_data=ds_train, epochs=20, batch_size=8, verbose=1)

# 模型测试
model.evaluate(eval_data=ds_test, verbose=1
```

最后输出：

```
Epoch 1/20
step 51/51 [==============================] - loss: 1164.6094 - 15ms/step
Epoch 2/20
step 51/51 [==============================] - loss: 864.6916 - 710us/step
Epoch 3/20
step 51/51 [==============================] - loss: 111.0113 - 727us/step
Epoch 4/20
step 51/51 [==============================] - loss: 542.9614 - 708us/step
Epoch 5/20
step 51/51 [==============================] - loss: 225.0129 - 726us/step
Epoch 6/20
step 51/51 [==============================] - loss: 110.6434 - 745us/step
Epoch 7/20
step 51/51 [==============================] - loss: 112.8198 - 922us/step
Epoch 8/20
step 51/51 [==============================] - loss: 92.5391 - 763us/step
Epoch 9/20
step 51/51 [==============================] - loss: 53.8990 - 747us/step
Epoch 10/20
step 51/51 [==============================] - loss: 81.2532 - 706us/step
Epoch 11/20
step 51/51 [==============================] - loss: 25.8484 - 726us/step
Epoch 12/20
step 51/51 [==============================] - loss: 62.0762 - 823us/step
Epoch 13/20
step 51/51 [==============================] - loss: 281.1697 - 902us/step
Epoch 14/20
step 51/51 [==============================] - loss: 50.4519 - 824us/step
Epoch 15/20
step 51/51 [==============================] - loss: 105.5184 - 706us/step
Epoch 16/20
step 51/51 [==============================] - loss: 65.0720 - 667us/step
Epoch 17/20
step 51/51 [==============================] - loss: 24.2344 - 941us/step
Epoch 18/20
step 51/51 [==============================] - loss: 111.5248 - 882us/step
Epoch 19/20
step 51/51 [==============================] - loss: 27.7191 - 804us/step
```

```
Epoch 20/20
step 51/51 [==============================] - loss: 44.8798 - 667us/step
Eval begin...
step 102/102 [==============================] - loss: 312.0678 - 382us/step
Eval samples: 102
```

这就是训练和测试过程的日志。

损失函数

在训练过程中，机器需要知道自己的预测值与真实值的差距，因此就有了损失函数。在回归问题中，通常使用均方误差损失函数。

回归问题就像这种问题一样，它的输出是连续的，也就是说，它可能取到24，也可能取到24加一个非常小的数，这意味着它的输出是有无限种可能的。

预测

模型的预测很简单，只需要使用 `model.predict_batch` 函数即可。需要注意的是，给这个函数传入的数据应该是一个二维数组，因为它用于预测一个批次的数据，也就是多条数据：

```
import numpy as np
pred = model.predict_batch(np.array([[0.27957, 0.00, 9.690, 0, 0.5850, 5.9260, 42.60,
2.3817, 6, 391.0, 19.20, 396.90, 13.59]], dtype=np.float32))
print(pred[0])
```

输出：

```
[[28.436752]]
```

模型保存和加载

当训练好一个模型后，我们希望将其保存，需要用到 `model.save` 函数：

```
model.save('model')
```

加载模型时，需要用到 `model.load` 函数：

```
model.load('model')
```