**人工智能实验1-知识表示**

**实验内容：**

参照课程第二部分讲授的知识表示方法完成，包括产生式系统、框架系统、语义网络等（还可以选择其他方法）解决以下问题（不限于此），必要时上网查找有关参考文献。

**猴子摘香蕉问题：**

一个房间里，天花板上挂有一串香蕉，有一只猴子可在房间里任意活动（到处走动，推移箱子，攀登箱子等）。设房间里还有一只可被猴子移动的箱子，且猴子登上箱子时才能摘到香蕉，问猴子在某一状态下（设猴子位置为A，香蕉位置在B，箱子位置为C），如何行动可摘取到香蕉。

**人工智能实验2-搜索策略**

**实验内容：**

实验要求采用且不限于课程第四章内各种搜索算法此编写一系列吃豆人程序解决以下列出的问题1-7 （填充search.py和searchAgents.py这两个文件中的空缺代码），包括到达指定位置以及有效的吃豆等。

**简介：**

基本代码和支持文件可以从search.zip中获取。其中，一些需要参考的文件如下：

## 需要编辑的文件：search.py和searchAgents.py。这两个文件中有一些空缺代码需要大家补充。

## 需要参考的文件：

|  |  |
| --- | --- |
| [pacman.py](https://s3-us-west-2.amazonaws.com/cs188websitecontent/projects/release/search/v1/001/docs/pacman.html) | 吃豆人游戏的程序。 文件包括一个描述”吃豆人”gamestate的类型。 |
| [game.py](https://s3-us-west-2.amazonaws.com/cs188websitecontent/projects/release/search/v1/001/docs/game.html) | 吃豆人游戏的运行逻辑. 文件包括以下类型AgentState, Agent, Direction, and Grid. |
| [util.py](https://s3-us-west-2.amazonaws.com/cs188websitecontent/projects/release/search/v1/001/docs/util.html) | 搜索策略可以用到的数据结构. |

## 可以忽略的支持性文件：graphicsDisplay.py graphicsUtils.py textDisplay.py ghostAgents.py keyboardAgents.py layout.pyautograder.py testParser.py testClasses.py test\_cases/ searchTestClasses.py

**解压缩search.zip，在此目录下，运行以下指令可打开吃豆人游戏。**

***python pacman.py***

**运行*python autograder.py*可以帮助你对自己的程序打分**。

searchAgents.py中最简单的Agent叫做GoWestAgent，一路向西，偶尔能实现目标：

*python pacman.py --**layout testMaze --pacman GoWestAgent*

但是其不能实现转弯：

*Python pacman.py --layout tinyMaze --pacman GoWestAgent*

如果程序卡死，可通过CTRL-c来终止。

此项目中用到的指令也都储存在commands.txt文件中，可用于复制和粘贴。

## 问题1：应用深度优先算法找到一个特定的位置的豆

首先，运行一下命令测试SearchAgent是不是正常工作：

*Python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch*

然后，完成完整的通用算法帮助吃豆人规划路线。搜索算法的伪代码见附录。注意一个搜索节点不仅包含节点的状态，而且要包含构建搜索路径所需要的信息。

注意：所有的搜索函数必须返回一个从初始状态到目标状态的操作序列。所有操作必须合法（不能翻墙）。

注意：利用util.py文件中提供的Stack, Queue 和 PriorityQueue数据结构！这是自动评分系统的兼容性要求。

你的code应该能顺利解决以下问题：

*Python pacman.py -l tinyMaze -p SearchAgent*

*Python pacman.py -l mediumMaze -p SearchAgent*

*Python pacman.py -l bigMaze -z .5 -p SearchAgent*

注意：因为不同的搜索方法的不同之处仅仅在于open表的排序不同，因此请尽量定义一个通用的搜索算法解决问题1-3。提示：问题1-3的不同之处在于用不同的数据结构对open表进行排序。

## 问题2：宽度优先算法

利用宽度优先算法实现解决以上问题。并利用以下命令测试你的code：

*Python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs*

*Python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5*

## 问题3：A\* 算法

完成A\*搜索方法(search.py文件中的aStarSearch函数)，利用曼哈顿距离作为启发函数，用以下命令测试你得code：

*Python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic*

## 问题4：找到所有的角落

在角落迷宫的四个角上面有四个豆。这个搜索问题要求找到一条访问所有四个角落的最短的路径。

完成searchAgents.py文件中的CornersProblem搜索问题，你需要重新定义状态，使其能够表示角落是否被访问。用以下命令测试你的code：

*Python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem*

*Python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem*

提示：新的状态只包含吃豆人的位置和角落的状态。

## 问题5：角落问题（启发式）

构建合适的启发函数，完成searchAgents.py文件中的cornersHeuristic角落搜索问题。用以下命令测试你的code：

*Python pacman.py -l mediumCorners* *-p AStarCornersAgent -z 0.5*

## 问题6：吃掉所有的豆子

用尽可能少的步数吃掉所有的豆子。完成searchAgents.py文件中的FoodSearchProblem豆子搜索问题。此问题利用之前A\*算法可以很容易找到解，可用以下命令测试：

*Python pacman.py -l testSearch -p AStarFoodSearchAgent*

构建合适的启发函数，完成searchAgents.py文件中的foodHeuristic豆子搜索（启发式）问题。用以下命令测试你的code：

*Python pacman.py -l trickySearch -p AStarFoodSearchAgent*

## 问题7：次最优搜索

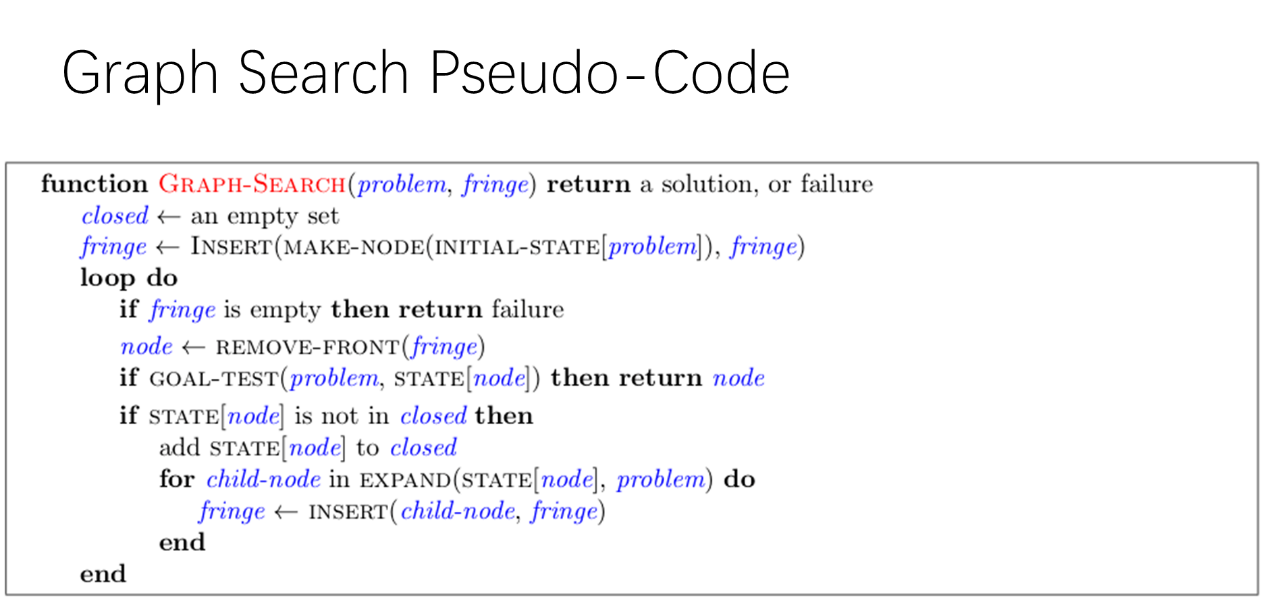
定义一个优先吃最近的豆子函数是提高搜索速度的一个好的办法。补充完成searchAgents.py文件中的AnyFoodSearchProblem目标测试函数，并完成searchAgents.py文件中的ClosestDotSearchAgent部分，在此Agent当中缺少一个关键的函数：找到最近豆子的函数。用以下命令测试你得code：

*Python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5*

# 附录

参考网址<http://ai.berkeley.edu/search.html>的实验内容。

**搜索算法的伪代码**



**人工智能-实验3不确定性推理**

**实验内容：**

参照课程第五部分讲授的贝叶斯网络完成。给定事件和事件之间的关系，并且给出每个事件的CPT图，编写程序自动读取文档实现基于贝叶斯网络的推理, 利用枚举或者变元消除法求出目标概率的精确解，利用采样方法求出目标概率的精确解。

**说明：**

在这里用到的贝叶斯算法是建立在有向无环图和CPT表的技术上实现的。

首先，给定的输入文件格式为：

N

rv0 rv1 ... rvN-1

0 0 1 ... 0

1 0 0 ... 1

...

0 1 1 ... 0

mat0

mat1

...

matN-1

在这里:

• N 是贝叶斯网络中随机事件的数目

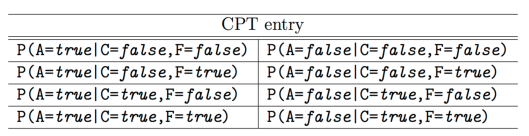
• rv 是随机事件的名字（字符串形式表示）

• mat是一个二维数组，分别表示从他的父亲到其本身的可能性概率。第一个元素表示发生的概率，第二个元素表示不发生的概率，显然两个元素相加为1

在上述中mat即为CPT表（Conditional Probability Table），其被设计为如下格式：

对于每个节点，如果他有N个父节点，则其CPT表中有2^N列，我们记为标号0 - （2N-1），其行序号的定义方法如下，利用二进制分别表示对应的父亲为是否发生，1为发生，0位不发生，将得到的二进制数转化为十进制代表其对应的行号。举例如下:

A有两个父节点C，F，则CPT表如下表所示:



其次，编写程序对应的查询格式为：P(rvQ | rvE1=val, rvE2=val, ...)

rvQ表示查询的条件的名字，即在rvE1=val, rvE2=val, ..发生的条件下，rvQ发生的概率。

RvEx表示条件的名称，而后面的val为true/false，分别表示发生和不发生。

最后，输出格式为两个数据分别表示P(QueryVar=true|...) 和P(QueryVar=false|...))。例如：

0.872 0.128

# 人工智能-实验4 基于Mindspore框架与ModelArts 平台的MNIST手写体识别实验

该实验包含了3部分内容：

1. **基于Mindspore框架的模型本地训练及预测**
2. **基于Modelarts平台和Tensorflow框架的模型训练及部署(请勿付费)**
3. **基于Mindspore框架的模型本地训练及预测**

## 实验介绍

本例子会实现一个简单的图片分类的功能，整体流程如下：

1、处理需要的数据集，这里使用了MNIST数据集。

1. 定义一个网络，这里我们使用LeNet网络。
2. 定义损失函数和优化器。
3. 加载数据集并进行训练，训练完成后，查看结果及保存模型文件。
4. 加载保存的模型，进行推理。

验证模型，加载测试数据集和训练后的模型，验证结果精度。

## 实验准备

在动手进行实践之前，确保，你已经正确安装了MindSpore。如果没有，可以通过MindSpore官网安装页面：<https://www.mindspore.cn/install/> ，将MindSpore安装在你的电脑当中。

同时希望你拥有Python编码基础和概率、矩阵等基础数学知识。

## 实验详细设计与实现

### 数据准备

我们示例中用到的MNIST数据集是由10类28\*28的灰度图片组成，训练数据集包含60000张图片，测试数据集包含10000张图片。

MNIST数据集下载页面：http://yann.lecun.com/exdb/mnist/。页面提供4个数据集下载链接，其中前2个文件是训练数据需要，后2个文件是测试结果需要。

将数据集下载并解压到本地路径下，这里将数据集解压分别存放到工作区的./MNIST\_Data/train、./MNIST\_Data/test路径下。

目录结构如下：

└─MNIST\_Data

├─ test

│ t10k-images.idx3-ubyte

│ t10k-labels.idx1-ubyte

│

└─ train

train-images.idx3-ubyte

train-labels.idx1-ubyte

为了方便样例使用，我们在样例脚本中添加了自动下载数据集的功能。

### 实验步骤

1. 导入Python库&模块并配置运行信息

在使用前，导入需要的Python库。

目前使用到os库，为方便理解，其他需要的库，我们在具体使用到时再说明。

详细的MindSpore的模块说明，可以在MindSpore API页面中搜索查询。

可以通过context.set\_context来配置运行需要的信息，譬如运行模式、后端信息、硬件等信息。

导入context模块，配置运行需要的信息。

parser.add\_argument('--device\_target', type=str, default=" Ascend", choices=['Ascend', 'GPU', 'CPU'], help='device where the code will be implemented (default: Ascend)')

在样例中我们配置样例运行使用图模式。再train.py和eval.py根据实际情况配置硬件信息，譬如代码运行在Ascend AI处理器上，则device\_target选择Ascend，代码运行在CPU、GPU同理。详细参数说明，请参见context.set\_context接口说明。使用本地和华为云处理器完成模型的训练。

1. 数据处理

#根据数据集存储地址，生成数据集，构建训练、验证函数

ds\_train = create\_dataset(os.path.join(args.data\_path, "train"), cfg.batch\_size)

ds\_eval = create\_dataset(os.path.join(args.data\_path, "test"), cfg.batch\_size,1)

数据集对于训练非常重要，好的数据集可以有效提高训练精度和效率。在加载数据集前，我们通常会对数据集进行一些处理。步骤二的以下内容里，mindspore 提供了更多数据处理的方法，供大家以后使用mindspore时参考：

/\*--------定义数据集及数据操作

我们定义一个函数create\_dataset来创建数据集。在这个函数中，我们定义好需要进行的数据增强和处理操作：

1. 定义数据集。

2. 定义进行数据增强和处理所需要的一些参数。

3. 根据参数，生成对应的数据增强操作。

4. 使用map映射函数，将数据操作应用到数据集。

5. 对生成的数据集进行处理。

#根据数据集存储地址，生成数据集

def create\_dataset(data\_dir, training=True, batch\_size=32, resize=(32, 32),

rescale=1/(255\*0.3081), shift=-0.1307/0.3081, buffer\_size=64):

#生成训练集和测试集的路径

data\_train = os.path.join(data\_dir, 'train') # train set

data\_test = os.path.join(data\_dir, 'test') # test set

#利用MnistDataset方法读取mnist数据集，如果training是True则读取训练集

ds = ms.dataset.MnistDataset(data\_train if training else data\_test)

#map方法是非常有效的方法，可以整体对数据集进行处理，resize改变数据形状，rescale进行归一化，HWC2CHW改变图像通道

ds = ds.map(input\_columns=["image"], operations=[CV.Resize(resize), CV.Rescale(rescale, shift), CV.HWC2CHW()])

#利用map方法改变数据集标签的数据类型

ds = ds.map(input\_columns=["label"], operations=C.TypeCast(ms.int32))

# shuffle是打乱操作，同时设定了batchsize的大小，并将最后不足一个batch的数据抛弃

ds = ds.shuffle(buffer\_size=buffer\_size).batch(batch\_size, drop\_remainder=True)

return ds

其中，

batch\_size：每组包含的数据个数，现设置每组包含32个数据。

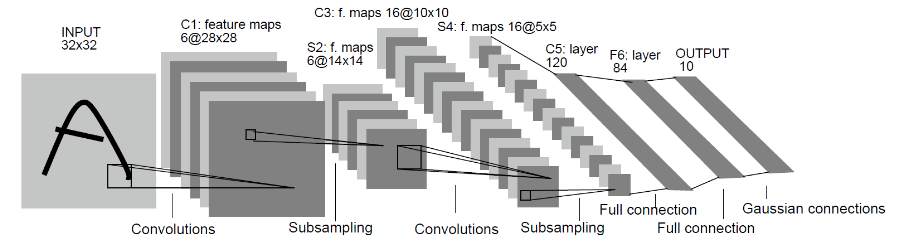
先进行修改图片尺寸，归一化，修改图像频道数等工作，再修改标签的数据类型。最后进行shuffle操作，同时设定batch\_size，设置drop\_remainder为True，则数据集中不足最后一个batch的数据会被抛弃。

MindSpore支持进行多种数据处理和增强的操作，各种操作往往组合使用，具体可以参考数据处理与数据增强章节。--------\*/

1. 定义网络

我们选择相对简单的LeNet网络。LeNet网络不包括输入层的情况下，共有7层：2个卷积层、2个下采样层（池化层）、3个全连接层。每层都包含不同数量的训练参数，如下图所示：

LeNet-5



更多的LeNet网络的介绍不在此赘述，希望详细了解LeNet网络，可以查询http://yann.lecun.com/exdb/lenet/。

使用MindSpore定义神经网络需要继承mindspore.nn.cell.Cell。Cell是所有神经网络（Conv2d等）的基类。

神经网络的各层需要预先在\_\_init\_\_方法中定义，然后通过定义construct方法来完成神经网络的前向构造。按照LeNet的网络结构，定义网络各层如下：

#定义模型结构，MindSpore中的模型时通过construct定义模型结构，在\_\_init\_\_中初始化各层的对象

class LeNet5(nn.Cell):

def \_\_init\_\_(self, num\_class=10, num\_channel=1):

super(LeNet5, self).\_\_init\_\_()

#定义卷积层，ReLU激活函数，平坦层和全连接层

#conv2d的输入通道为1维，输出为6维，卷积核尺寸为5\*5，步长为1，不适用padding

self.conv1 = nn.Conv2d(num\_channel, 6, 5, pad\_mode='valid')

self.conv2 = nn.Conv2d(6, 16, 5, pad\_mode='valid')

self.fc1 = nn.Dense(16 \* 5 \* 5, 120, weight\_init=Normal(0.02))

self.fc2 = nn.Dense(120, 84, weight\_init=Normal(0.02))

self.fc3 = nn.Dense(84, num\_class, weight\_init=Normal(0.02))

self.relu = nn.ReLU()

self.max\_pool2d = nn.MaxPool2d(kernel\_size=2, stride=2)

self.flatten = nn.Flatten()

#构建Lenet5架构，x代表网络的输入

def construct(self, x):

x = self.max\_pool2d(self.relu(self.conv1(x)))

x = self.max\_pool2d(self.relu(self.conv2(x)))

x = self.flatten(x)

x = self.relu(self.fc1(x))

x = self.relu(self.fc2(x))

x = self.fc3(x)

return x

1. 定义损失函数及优化器

在进行定义之前，先简单介绍损失函数及优化器的概念。

损失函数：又叫目标函数，用于衡量预测值与实际值差异的程度。深度学习通过不停地迭代来缩小损失函数的值。定义一个好的损失函数，可以有效提高模型的性能。

优化器：用于最小化损失函数，从而在训练过程中改进模型。

定义了损失函数后，可以得到损失函数关于权重的梯度。梯度用于指示优化器优化权重的方向，以提高模型性能。

定义损失函数

MindSpore支持的损失函数有SoftmaxCrossEntropyWithLogits、L1Loss、MSELoss等。这里使用SoftmaxCrossEntropyWithLogits损失函数。

MindSpore提供了callback机制，可以在训练过程中执行自定义逻辑，这里使用框架提供的ModelCheckpoint为例。 ModelCheckpoint可以保存网络模型和参数，以便进行后续的fine-tuning（微调）操作。

#设定loss函数

net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")

#设定优化器

net\_opt = nn.Momentum(network.trainable\_params(), cfg.lr, cfg.momentum)

#编译形成模型

model = Model(network, net\_loss, net\_opt, metrics={"Accuracy": Accuracy()})

# 训练网络 train.py

model.train(cfg['epoch\_size'], ds\_train, callbacks=[time\_cb, ckpoint\_cb, LossMonitor()], dataset\_sink\_mode=args.dataset\_sink\_mode)

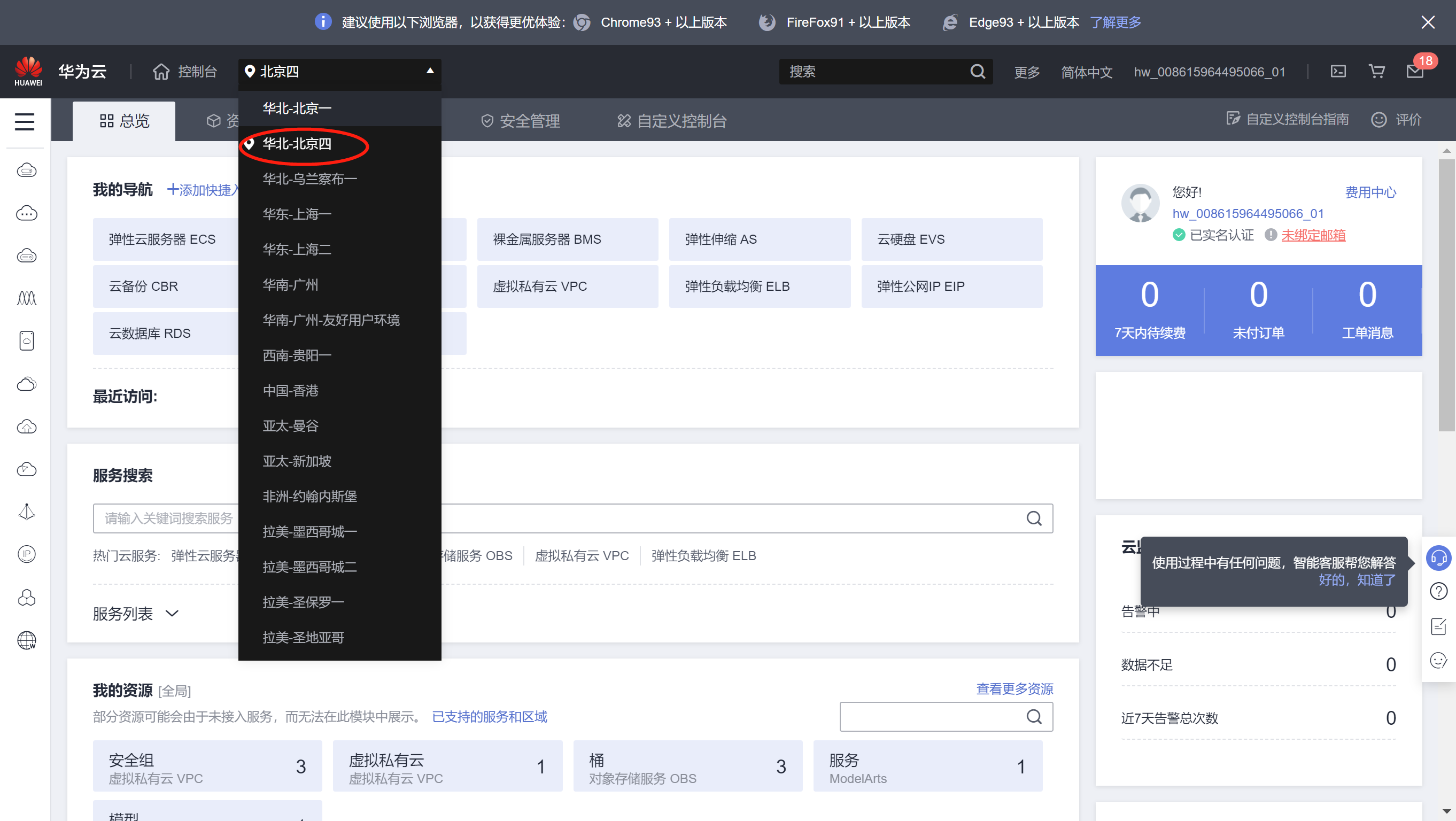
1. 开始训练及验证过程

利用train.py和eval.py完成模型的训练及验证过程。

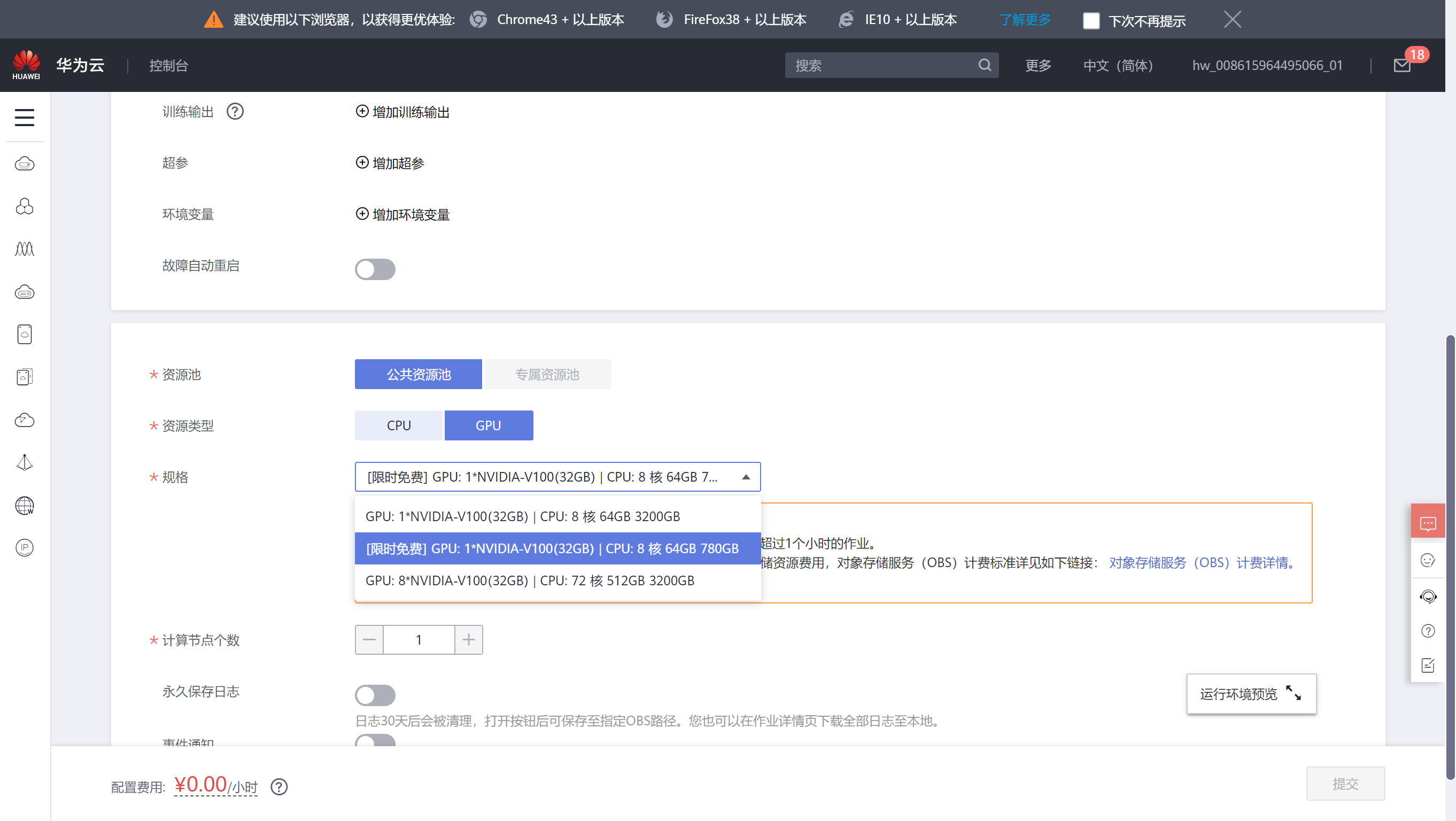
训练过程中会打印loss值。loss值会波动，但总体来说loss值会逐步减小，精度逐步提高。每个人运行的loss值有一定随机性，不一定完全相同。

1. **基于Modelarts平台和Tensorflow框架的模型训练及部署（请勿付费）**

**注：控制台服务器要选择：华北-北京四，如下图。**



**模型训练：请选择免费GPU资源，如下图。请勿付费。**



参考华为官方网页实例（<https://support.huaweicloud.com/bestpractice-modelarts/modelarts_10_0080.html>）在昇腾服务器上实现Tensorflow模型的训练部署及测试

## 此文档内容基于华为人工智能导论实验指导书改编，仅供哈尔滨工业大学《人工智能》课程学生实验使用。