Lab 2

历次修改
v1.0:
与问题相关的修正 1. answerLogisticRegression 15 行注释
@return: 输出标签为正的概率。 -> @return: wx+b 2. input/nn.Linear.in与nn.multilayer.in除去Logsoftmax前的relu。

与问题无关的修正 1. layout.py 146行的random.choice换成np.random.choice 2.增加了pdf版本的README

目录

介绍

在这个项目中，你将使用numpy实现课上学到的各种算法，以帮助pacman识别具有不同正负分数的豆子。

代码库发生了大幅改变，请从教学网上下载lab2。

本项目包括一个自动评分器，供你在机器上对答案进行评分。

用以下指令可以对所有题目进行评测。

```
python autograder.py
```

你需要修改的文件:

info.yaml 姓名学号

- 线性回归

answerLogisticRegression.py 线性回归模型

- 分类树

answerTree.py 树模型

answerRandomForest.py 随机森林模型

- 反向传播算法

autograd/BaseNode.py: 计算图上的结点

autograd/BaseGraph.py: 计算图类

answerMultiLayerPerceptron.py: 用反向传播实现MLP

- 给pacman添加视觉

MnistModel.py pacman agent的视觉模型

YourTraining.py 视觉模型的训练代码

你需要阅读的文件:

mnist.py 载入MNIST数据集的接口

debug_scalarfunction.py 用于debug 反向传播和评分的文件

debug_neuralnetwork.py 用于debug 反向传播和评分的文件

answerSoftmaxRegression.py：用反向传播实现SoftmaxRegression

你可以忽略的文件:

`pacman.py` 运行Pacman游戏的主要文件。这个文件还描述了一个Pacman `GameState` 类型。

`game.py` 吃豆人世界运作的逻辑。这个文件描述了几个支持类型，如AgentState、Agent、Direction和Grid。

`util.py` 实现算法时的可能会用到的数据结构。

`graphicsDisplay.py` Pacman图像显示界面

`graphicsUtils.py` 支持图像

`textDisplay.py` Pacman的ASCII编码

`ghostAgents.py` Ghosts程序

`keyboardAgents.py` 键盘接口

`layout.py` 导入地图的程序

`autograder.py` 自动评分器

需要修改并提交的文件: 提交所有需要修改的文件。你可以使用以下指令来打包答案文件。

```
python compressAnswers.py
```

评估: 我们使用 `autograder.py` 对你的提交进行评分，但包含的测试用例与本地给出的样例文件有所不同。你可以在本地运行评分器对你的代码进行评测，但仅限于帮助调试代码，该分数不等于最终分数。

学术诚信: 我们会将你的代码与课堂上其他提交的代码进行逻辑查重。如果你拷贝了别人的代码，并做一些微小的修改，我们会很容易发现，请不要尝试。我们相信你们会独立完成作业。

可能用到的一些操作

copy.deepcopy: 复制对象

numpy: https://numpy.org/doc/stable/user/absolute_beginners.html (特别注意array的indexing, slicing, random number随机数生成, transposing and reshaping，broadcast广播，np.unique(return_count=True))

A@B: 矩阵乘法 AB

注意以下操作的axis和keepdims参数
np.sum:
np.average:
np.std:

数据说明

本项目使用MNIST数据集。我们按照1000:69000将其分成训练集和验证集(MNIST/train_data.npy, MNIST/valid_data.npy)，并通过对MNIST数据集进行数据增强，得到了测试集(MNIST/test_data.npy)，用于pacman游戏中。目前只公布了测试集的一个子集。q1,q2,q3的所有的模型训练代码最终给出的准确率(acc)都是验证集上的，也请以验证集分数为依据调参。

mnist.py 提供了载入数据的统一接口。这一文件中你可能用到的变量有

```
num_feat = 784 #每张图片是28\*28的矩阵。在训练集和验证集中，图片已经展平成一个784维的向量。在测试集中没有。
num_class = 10 #MNIST的任务是识别0-9数字，也就是一个10分类任务
trn_X, trn_Y #训练集的图片与标签。每个标签是一个0-9的整数。
val_X, val_Y #验证集的图片与标签。
data, targets #测试集的图片与标签。
```

autograder.py可以自动评分。对于q1,q2,q3，评分基于训练的模型在验证集上的准确率（而不看pacman分数），给出的分数和最终的分数一样。对于q4，评分基于pacman在公开的测试集子集上得到的分数。而最终分数我们会使用提交的代码在完整的测试集上测试得到。

q1,q2,q3的数据划分是固定的。在q4中，你可以自由对MNIST数据(trn_X, trn_Y, val_X, val_Y)进行划分，但请不要使用公布的测试集子集进行训练或测试（这个子集只是给你们一个关于最终测试数据长什么样的印象）。请记住，永远不要用测试集来训练或调参。我们在计算最终分数时会在一个没有测试集数据的环境下进行训练。

新的pacman规则

与lab1相比，lab2的pacman中每个食物被赋予了一个数字，不同的数字对应不同的分数，有正有负。pacman只允许走有限步。只有当pacman在有限步内吃掉所有正分数的食物时获胜（+50分）。移动1步不扣分，吃到食物加上相应的分数。你不需要自己写agent，只需要设计图片分类模型让pacman能辨认数字。

地图文件存放在layouts文件夹下，一个典型的地图文件如下

```
%%%%%%%%%%
%87 32 2P0%
%65 6  1 4%
%4  10 993%
%%%%%%%%%%
# -10 -10 -10 -10 -10 10 10 10 10 10 False
```

其中P代表pacman的起始位置，%代表墙壁。数字代表食物的数字。根据数字，游戏会随机从数据集中抽取数字对应的图片。最后一行从前到后代表着0-9数字对应的分数。最后一个True/False代表着是否轮换数字。例如轮换后，地图左上角的数字8 （10分）可能会变为2（-10分）。换言之，你不能让你的模型只对特定的类识别特别准。

所有用于评分的地图已经公布。测试地图中都没有鬼。如果你想实现有鬼的情况，可参见选做的q5，不计分，不提交。

Question 1 (4 points): 逻辑回归

使用第9节课上的内容实现逻辑回归。在MNIST上定义一个线性模型以区分一个数字是否是0，并用梯度下降完成优化。

modelLogisticRegression.py给出了模型的训练流程。你需要修改answerLogisticRegression.py，完成模型定义和一次梯度下降过程，并进行调参。

写完后运行

```
python modelLogisticRegression.py
```

进行训练并保存模型。

最后运行

```
python pacman.py -p ReflexAgent -a model="LR" -l strict --maxstep 40
```

来看你的模型在pacman中的运行效果。这个例子使用layouts/strict.lay作为地图，只有数字0是正分，且只有一条严格的道路能吃完所有正分食物获胜。

可以在指令最后添加`-v`或者`--visualize`在地图中可视化模型对每个数字的预测结果（右下角红色数字）。

思考：你会发现虽然你的准确率可能很高，但实际pacman表现得很差。这可能是由于类别不均衡(class imbalance)问题导致的。MNIST中90%的数据是非零，因此一个很差的一直预测非零的分类器就可达到90%的准确率，但实际中它一个零也预测不出来。对于类别不均衡问题，准确率并不是个很好的指标，此时可以使用confusion matrix来更好地衡量模型性能。本题目中你不需要解决类别不均衡问题，我们最终仍然以准确率来评分。

运行

```
python autograder.py --q q1
```

来评分(0-100)。注意如果你使用默认的超参数可能分数会很差。

注意autograder可能会报错`python: command not found`，此时请修改autograder.py开头的PYTHON_PATH为你的python程序的绝对位置（Linux和苹果用户可以在shell中用`which python`，windows用户可以在powershell中用`gcm python`）

Question 2 (5 points): 树模型

modelTree.py给出了分类树模型的训练流程，你需要修改answerTree.py构建树模型对MNIST进行10分类，并进行调参。

写完后运行

```
python modelTree.py
```

进行训练并保存模型。

最后运行

```
python pacman.py -p ReflexAgent -a model="Tree" -l accuracy --maxstep 40
```

来看你的模型在pacman中的运行效果。这个例子使用layouts/accuracy.lay作为地图，偶数数字是正分，奇数数字是负分。

在树模型的基础上，你需要构建随机森林模型。modelRandomForest.py给出了训练流程，你需要修改answerRandomForest.py。

写完后运行

```
python modelRandomForest.py
```

进行训练并保存模型。

运行pacman:

```
python pacman.py -p ReflexAgent -a model="Forest" -l accuracy --maxstep 40
```

同样地，可以在指令最后添加`-v`或者`--visualize`在地图中可视化预测结果，下面的问题中也可如此设置。

运行

```
python autograder.py --q q2
```

来评分。

Question 3 (6 points): 反向传播

修改autograd/BaseNode.py与autograd/BaseGraph.py完成计算图定义，例如，在BaseNode.py中，你需要补全一些计算结点的前向或反向传播结果（对于反向传播，输入为上游梯度，输出为下游梯度）。在BaseGraph.py中，你需要为一个任意给定的计算图实现前向传播和梯度的反向传播。注意本题中只需要考虑每个结点只有一个输入和一个输出的情况（结点连成直线的计算图）。

你可以使用debug_NeuralNetwork.py与debug_ScalarFunction.py进行debug。它会通过有限元近似计算数值梯度。请确保你定义的计算图和计算结点通过反向传播得到的梯度和数值梯度大致相等，再进行模型的搭建。

```
python debug_ScalarFunction.py input/sf.relu.in --debug
```

其中`sf.relu.in`是一个定义标量计算图的文件，可以换成`sf.sigmoid.in`，`sf.tanh.in`，`sf.mix.in`. `--debug`会开启debug模式，输出各结点输入输出的矩阵形状。类似地，你也可以对神经网络计算图进行debug。

```
python debug_NeuralNetwork.py input/nn.BatchNorm.in --debug
```

其中的`nn.BatchNorm.in`可以换成`nn.CELoss.in`，`nn.Dropout.in`，`nn.Linear.in`，`nn.multilayer.in`，`nn.NLLLoss.in`.

基于反向传播框架，你可以实现Softmax回归与多层感知机，他们的训练流程分别在modelSoftmaxRegression.py与modelMultiLayerPerceptron.py。Softmax回归已经实现，参见answerSoftmaxRegression.py。请你在answerMultiLayerPerception.py中使用计算图定义一个多层感知机(MLP)使其能在MNIST上进行10分类，同时进行调参。

写完后运行

```
python modelSoftmaxRegression.py

python modelMultiLayerPerceptron.py
```

进行训练并保存模型。

运行

```
python pacman.py -p ReflexAgent -a model="SR" -l accuracy --maxstep 40

python pacman.py -p ReflexAgent -a model="MLP" -l accuracy --maxstep 40
```

来看你的模型在pacman中的运行效果。注意，即使你之前在验证集上获得极高的准确率(>90%)，有可能pacman仍然对测试集预测得不是很准，导致表现不好。你将会在下个问题中尝试解决它。具体可查看autograder.py。

运行

```
python autograder.py --q q3
```

来评分。本题的评分只看是否你实现的计算结点能通过有限元数值梯度的debug测试，以及你实现的MLP在验证集上的准确率。

Question 4 (5 points): 给pacman添加视觉

在本问题中，你可自由定义并训练任何模型，使得pacman能识别数字。在MnistModel.py中调用你训练好的模型。直接使用q1-q3中的模型、或暴力记住所有MNIST数字可能并不能取得很好的效果。

你可以调lr, wd, batch size, 网络结构等，甚至实现卷积算子。允许使用全部给你的数据集。允许模型集成。允许数据增强。评分时的测试集并没有放出。可以参考之前的model*.py构建训练流程，在YourTraining.py中训练模型。请控制训练时间在600秒以内（autograder会检查）。修改MnistModel.py中的modeldict["Your"]来放入你的模型。

注意我们测试时会首先在一个没有testdata的环境中运行YourTraining.py，然后在一个没有YourTraining.py的环境中测试Agent。由于限时，请不要在测试时进行训练，即不要在MnistModel.py中包含训练代码。

执行

```
python pacman.py -p ReflexAgent -a model="Your" -l smallPKU
```

来看看具备"Your" model赋予的视觉能力的pacman在smallPKU地图上表现如何。如果你的模型一开始表现不是很好，也不要气馁，这里的测试集都是经过数据增强的较难样本，需要你开动脑筋，努力提升你模型的泛化性能。

运行

```
python autograder.py --q q4 --q4training
```

来评分。`--q4training`表示会调用YourTraining.py进行训练。如果已经训练完成，可以去掉这一选项。评分会分别在"largeAccuracy", "onepath", "VeryLargePKU"三个地图上收集多次pacman的得分来计算最终得分。

Question 5 (0 points): 使用你自己的Agent

本题不提交，不计分。

本题允许使用你在lab1中创造的agent。可以将你的Agent加入MultiAgent.py中，并将原来的getAction改名称为getVisAction就行。

我们提供了一个有鬼的地图 ghostPKU，可运行如下命令测试你的Agent在有视力、对抗情况下的表现

```
python pacman.py -p YourAgent -a model="Your" -l ghostPKU
```

Acknowledgement

Designed and implemented by Xiyuan Wang, Fanxu Meng and Muhan Zhang from the GraphPKU Lab. Thanks Hongze Li, Zichen Liu and Jiahuan Zhou for module contributions, testing and code comments. Thanks all professors and TAs of the 2023 Introduction to AI team for the valuable discussions. The project is based on UC Berkeley CS188 course project.