EE228 课程大作业

基于集成学习训练深度卷积神经网络控制2048游戏

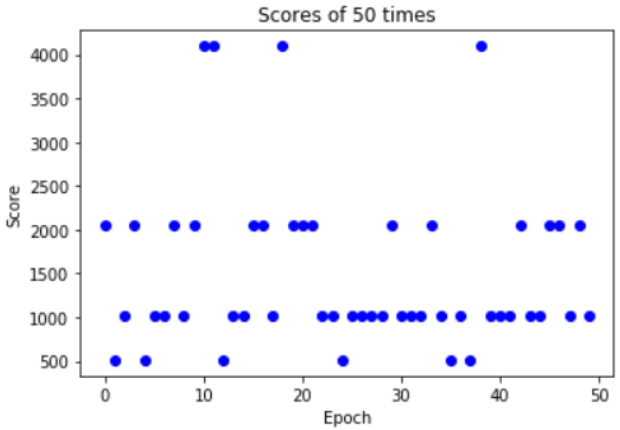
姚逸然 518021910646

2020年06月20日

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

项目1完成情况

- 50轮平均分数（无限制）：1515

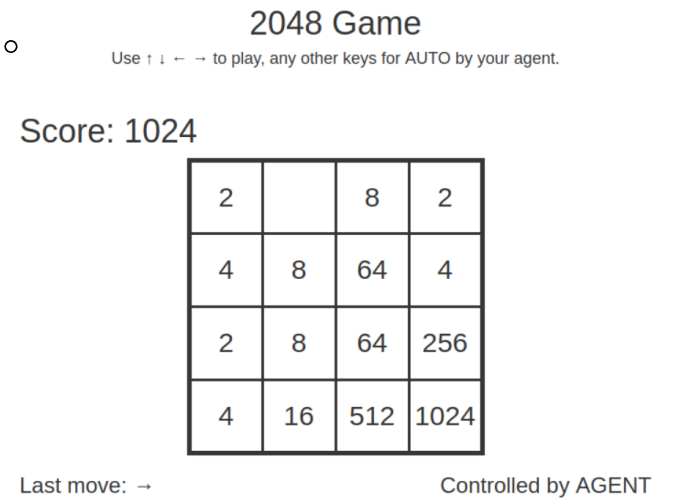


- 10轮平均分数（有限制）：1331
- 方法简述：基于集成学习训练深度卷积神经网络控制2048游戏
- 主要使用的代码框架：Keras
- 模型大小（MB）：446（共7个模型，实际使用时选择3-5个即可）
- 亮点：集成学习， CNN
- 加分项：尝试过使用强化学习，但是模型很不稳定
- 代码链接：https://github.com/SpeagleYao/2048-api

问题描述

- 本项目要求使用监督学习来进行2048游戏。

- 2048游戏使用方向键让4*4棋盘内的所有方块整体上下左右移动。若两个数字相同的方块在移动中碰撞，则会合并为一个新的方块，且其数字为两者之和。每次移动后，会随机生成一个值为2或4的新方块。当棋盘中出现值为2048的方块时，游戏胜利。当棋盘被填满而无法生成新方块时，游戏失败。



问题描述

- 接下来，我们将问题描述转换为机器学习的语言。

- 所求模型是基于监督学习的分类模型。数据集应包含输入和输出。
- 所求模型的输入是当前4*4的棋盘。
- 所求模型的输出是上下左右四个方向中的一个。

- 我们希望得到的模型是能够通过不断输入当前棋盘状态并返回输出方向以游玩2048游戏并最终取胜的。

模型设计

- 由于2048游戏的每一步为向四个方向中的一个移动，则棋盘上的所有方块都只能在其所在的横纵直线上移动。因此，4*4的棋盘作为输入是具有空间信息的。这启发我使用CNN（卷积神经网络）来解决这个问题。

- 一个经典的CNN可细分为一下五个部分：
- **输入层 —— 卷积层 —— 池化层 —— 全连接层 —— 输出层**

- 接下来就分别对上述五个部分进行设计，即可得到模型。

模型设计

- **输入层**

- 如果直接将4*4的棋盘作为输入，单看棋盘上的2、4、8、16...这些数字会显得不明所以。经过测试，模型确实很难从这样的输入中学到东西。
- 因此，我将棋盘输入转为One-hot Vector表示。首先，将棋盘上的空余部分用0表示。之后，0转变为[1, 0, ..., 0]，2变为[0, 1, ..., 0]，以此类推。如此一来，如果棋盘上会出现的最大数字是n，则One-hot vector的长度为$c \triangleq \log_2 n + 1$。这样一来，输入的维度变为了4*4*c。这个c可类比为处理图片时每个图片的信道。
- 使用One-hot vector表示输入有两个好处：
- 1） 将每个数字转为对应向量，明确了这些数字的含义
- 2） 每个信道所表示的是某数字在4*4的棋盘上的分布，空间信息凸显

模型设计

- **卷积层**
- 考虑到模型主要应学到每行、每列及整个棋盘所含的空间信息，因此设计三种大小的卷积核，分别为1*4，4*1，4*4，共同作用于卷积层。在基本模型中，每种卷积核的数量设计为64个。

- **池化层**
- 池化层一般是用来进一步提取特征，增强模型泛化能力的，可以看做是对现有信息的浓缩。考虑到每一信道大小仅有4*4，无需压缩，且压缩后信息意义不明，因此模型中不引入池化层。经过测试，引入池化层后的模型确实表现下降。
- 为与之后的全连接层对接，在此会将卷积层的输出改变为一维向量。

模型设计

- **全连接层**
- 全连接层即是最基本的神经网络，用以进一步处理卷积层是输出。在基本模型中，全连接层设计为2层512个神经元的隐层。

- **输出层**
- 由于这是一个多分类模型，即将每个输入分为4个方向中的一种。因此，输出层选用含有4个神经元的SoftMax层。

- 以上即为基本模型设计。
- 基本模型中可训练的参数约有185W个。

```
Total params: 1,874,116
Trainable params: 1,866,948
Non-trainable params: 7,168
```

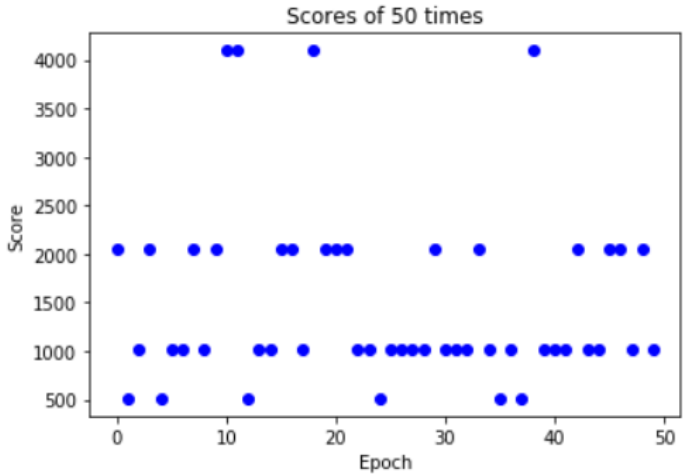
模型设计

- **数据集**
- 模型学习中所使用的数据集基于项目代码中提供的ExpectiMaxAgent生成。
- 所使用的数据集是以8192为胜利条件的。共有约1500W组数据。

- **集成学习**
- 经过测试，单个模型在50轮2048游戏中的平均分数约为850，仍为达到要求。因此，我进一步训练了多个进行了少量改进（如引入2*2、3*3卷积核，加深加大全连接层，更换非线性层等）的模型，并在预测时采用多个模型投票选最多的方式得出结果。如此做，在模型结构不变的基础上大大提升了游戏的平均分数。
- 我共生成了7个模型，实际运行时选取3-5个即可。

性能分析

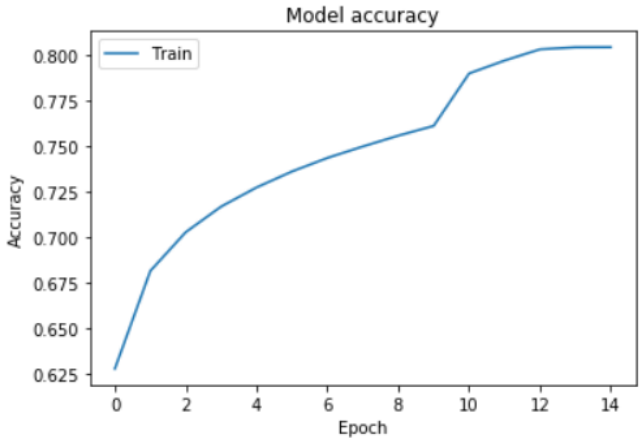
- Agent在50轮中的成绩分布
- 如右图所示，可以看到在50轮中，Agent在多数情况下达到了1024分，部分情况下达到了2048分，但也有个别的低分（512）和高分（4096）。

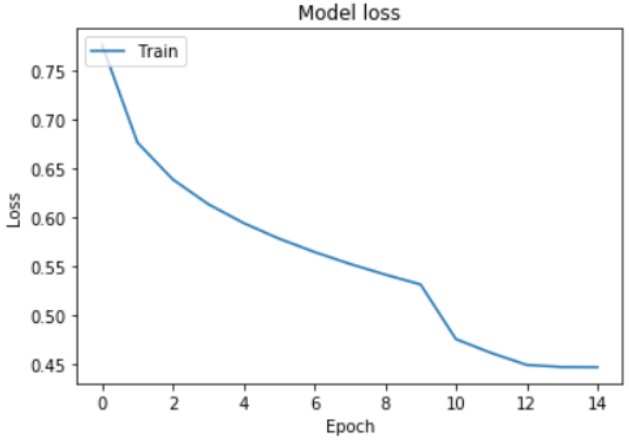


- Agent的稳定性相对较好，但是仍有少数低分存在。考虑到每轮棋盘的随机性，这很难避免。
- Agent测试50轮所花时间为7分50秒。据此粗略估计，在1s内模型内预测约80步，即每8s可以完成一个达到2048分的棋盘。由于每一步预测只需计算7个模型的前向传播，因此模型单步预测时长较短。
- 如果选用5个模型进行预测，所用空间约为300MB。

技巧设计

- 在训练过程中，我设置batch_size = 512，optimizer = ‘Adadelta’。这两项即缩短了单轮训练时长，同时还加速了收敛。
- 在每轮训练后，我会监测验证集的loss。如果验证集的loss下降较小，则说明模型当前已接近收敛，因此在下一轮训练时，模型的learning_rate会自动降为原来的1/10。这会使得模型精确度迅速提高，加快收敛速度。
- 在多轮训练结束后，我会输出训练集的acc和loss，调整模型以提高准确性并加速收敛。在反复尝试后，我才确定了现在的模型参数设置。





讨论

- 在这次项目完成过程中，我充分感受了深度学习领域的“大力出奇迹”。在我的测试中，数据集的大小和模型参数大小每增大10倍，训练结果中验证集的准确度就会提升约10个点，最终平均分数约翻一倍。从验证集准确度最初的45%提升到最终的80%，这个进步主要是依靠“大力”完成的，而精细的参数修改在其中只占了小部分。这让我体会到了为何现在的深度学习更像是一门工程学，而不是数学。

- 在最初卷积神经网络效果并不好的时候，我尝试使用强化学习中的DQN算法来解决这个问题（代码一并上传到了github）。但是无论我怎么修改奖励函数，我训练出的网络始终不稳定，每轮游戏的分数会在64~256分间反复徘徊。我推测这是因为2048游戏的随机性相对较强，且表示出的棋盘不同状态间联系较弱，致使本就难以收敛的DQN更难以收敛。也许之后可以采取更稳定的强化学习算法再做尝试。

Thank You