深度学习大作业-2048

目录

1 题目描述

我们希望训练一个神经网络来在 2048 游戏中获得更多的分数。规则是控制所有方块向同一个方向运动，两个相同的数字方块撞在一起之后合并，成为其和，每次操作之后会随机产生一个 2 或者 4，每次合并能取得相应的分数。

课程组提供了一个完整的训练模型，将其建模为一个马尔可夫决策过程，通过强化学习方法来完成。训练过程中模型最高可达 8000 分，平均分 2000 左右。我们需要理解相关代码和模型，并调整其参数，修改模型，使得其平均得分尽可能的高。

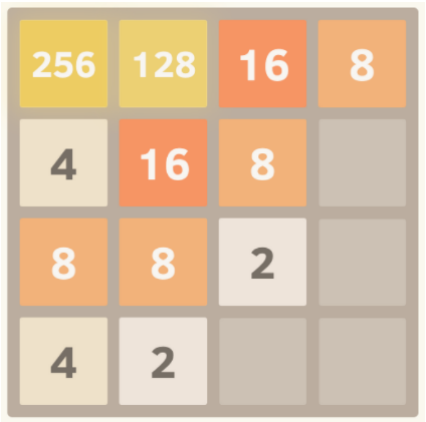


图 1: 2048 游戏界面

2 所做的工作

2.1 改变网络结构

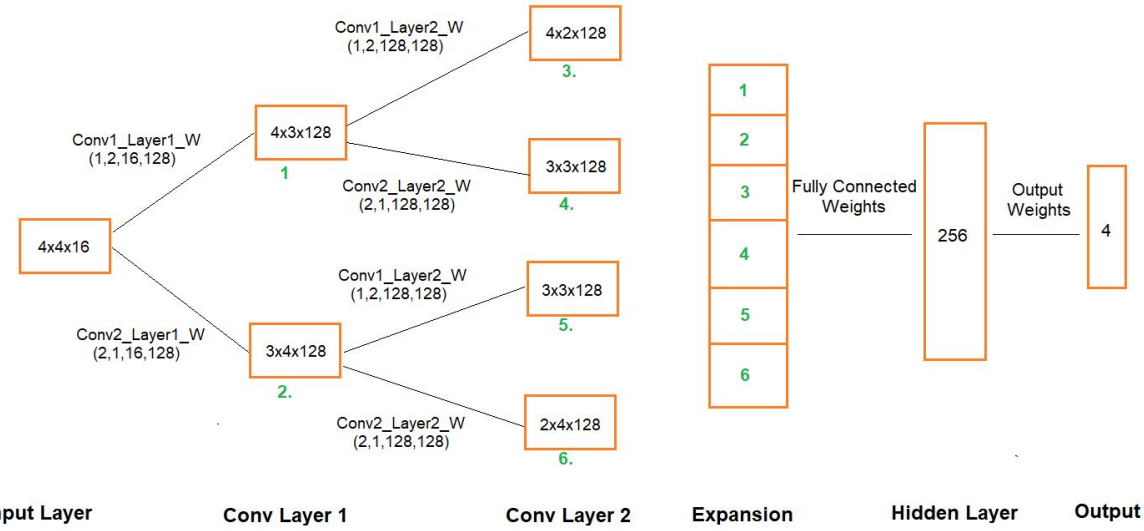
用于决策的网络是模型中的关键部分，我们对其进行了一些结构上的修改。

2.1.1 移动种类

模型的输出应该是四种移动方式对应的 Q 值，然而提供的代码中输出个数为 3，我们将其修改为 4。

2.1.2 网络结构

将 GitHub 项目 navjindervirdee/2048-deep-enforcement-learning 的网络结构使用 MegEngine 实现，如下图所示。



该网络深度较浅，且全连接层与每个卷积层都有直接连接，比 baseline 中的网络能够更好拟合决策函数。

2.2 Epsilon-greedy

2.2.1 简介

本文使用的 DQN 模型的原理就是在训练过程中获取更多经验，由于 2048 模型的状态可以说是无限的，不可能遍历所有的状态，我们就需要模型通过对于有限的样本的学习来在更多状态中选择出一条最优路径。但是在对于某种固定的状态，模型在第一次选择了路径之后就会固定下来，这显然不利于我们训练出更好的模型。

所以我们设定一个 Epsilon，然后在每次选择路径的时候先生成一个 0-1 的随机数，如果这个数字小于 Epsilon，就随机选择一条路径，如果大于 Epsilon，就按照原来的贪婪算法，选择 Q 值最大的路径。这样就可以保证模型有一定的几率探索到更多的路径。

在足够多次之后，再降低 Epsilon 的值，我们设置在训练过程的最后 25%Epsilon = 0，这样就可以保证在这些训练过程中可以选择最优路径，获得更高的分数。

2.2.2 代码实现

```
choice_greedy = F.argmax(values, 1) # 初始版本使用的贪心算法的选择
a = numpy.zeros([values.shape[0]], dtype=int)
e = 1. / (epoch / 100 + 1) # Epsilon
if epoch >= epochs * 0.75: # 当训练进行到最后 25\% 时将 Epsilon 设为 0，即使用贪心算法
    e = 0
for k in range(0, games_count):
    random_num = random() # 生成用于选择策略的随机数
    if random_num >= e:
        a[k] = int(choice_greedy[k])
```

```
10        else:
11            a[k] = int(randint(0, values.shape[1] - 1))
```

2.3 Prioritized Experience Replay-优先经验重播

2.3.1 简介

当我们对经验进行随机采样来让神经网络学习时，某些可以获得更高分数的经验是比其他经验更有价值的。在均匀采样的 DQN 中，所有经验都具有相同的采样概率。因此，在培训结束时，每种体验的使用次数将大致相同。而如果使用权重进行采样，就可以使一些更有用的经验平均被采样更多次，这个方法已经在论文中被验证过，证明可以取得更好的效果。

我们采用了 SumTree 的方式来实现经验的存储和获取。

2.3.2 SumTree 的存储与采样过程

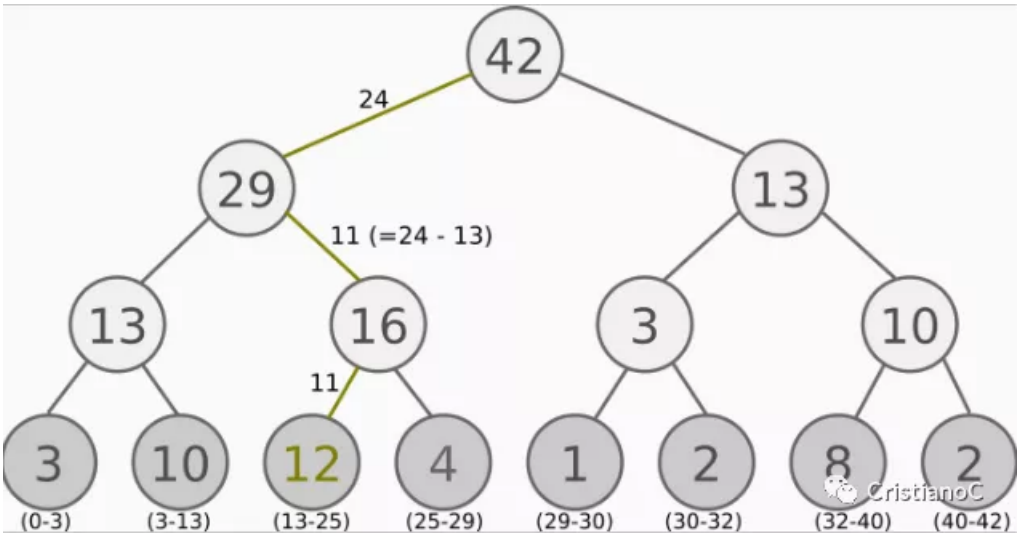


图 2: SumTree 的结构

图中的叶子节点为所有经验，数字为该经验对应的权值。其他节点为子节点的权值之和。

假设要抽取六个样本，则在 [0-7], [7-14], [14-21], [21-28], [28-35], [35-42] 区间中各随机挑选一个数字。比如在第 4 个区间 [21-28] 选到了 24，就按照这个 24 从最顶上的 42 开始往下搜索。首先看到最顶上 42 下面有两个子节点，左边的 29 比 24 大，则选择左边；然后 24 比 13 大，将现有权值更新为 24-13=11，并选择右边。这样最终可以获得一个样本。也实现了根据权重来抽取样本。

2.3.3 进行的修改

创建了 SumTree 和 Memory 类用于存储和获取样本，此外对数据格式进行了处理，在调用函数的地方进行了相应修改。

2.4 优化 reward 的计算

初始版本中的 reward 仅使用这一步操作中获得的分数作为 reward。

我们对这一计算方式进行了修改，使 reward 的值与前后两个状态的空格数、前后两个状态中数值最大的方块值有关。

2.5 调参

2.5.1 增加 epoch 数量

训练时观察训练曲线发现，10000 个 epoch 结束后，score 仍然有上升趋势，所以将 epoch 数量调整到 20000，使学习更加彻底。

2.5.2 学习率的指数衰减

学习率即步长，它控制着算法优化的速度。使用的学习率过大，虽然在算法优化前期会加速学习，使得损失能够较快的下降，但在优化后期会出现波动现象以至于不能收敛。如果使用的学习率偏小，那么极有可能训练时 loss 下降得很慢，算法也很难寻优。

代码中的 lr 为固定值 1e-4，我们分别尝试了使用 megengine.optimizer.MultiStepLR 和手动修改两种方法实现 lr 的衰减。

以下为使用 MultiStepLR 实现 lr 衰减的**部分**代码：

```
LR = 0.01
opt = Adam(model.parameters(), lr=LR)
scheduler = torch.optim.lr_scheduler.StepLR(opt, step_size=5, gamma=0.8)
```

手动实现 lr 衰减的**部分**代码，这里设置 lr 的初始值为 0.9，以后每次衰减为原来的 0.9。

```
params_g = opt.param_groups[0]
tmp = params_g["lr"] * 0.9
params_g["lr"] = tmp
opt._updates(params_g)
```

由于这两种方法实现 lr 的指数衰减后，平均分并没有提升，所以放弃了这一修改。

2.5.3 DDQN 算法的步长

Double DQN 算法中有一个步长参数，该值大于 1 能够有效避免 Q-learning 学习中“过估计”带来的影响，baseline 中的对应取值为 10，相关代码如下所示。在其他条件保持一致的情况下调整该值的大小，选取 5、10、15、20、30、40 等情况，结果如下表所示，可见调整该值效果不明显，最终我们放弃对该值的调整。

```
        '''double DQN'''
    if epoch % 10 == 0:
        mge.save(model, "1.mge")
        model_target = mge.load("1.mge")
```

步长	最大得分	平均得分
5	8620	3699.0
10	9402	3685.7
15	10044	3412.3
20	9284	3441.7
25	9110	3436.0
30	12506	3642.2
40	8912	3714.9

表 1: DDQN 步长调整结果

3 最终结果

使用提供的代码运行结果：
loss=0.02725, Q=5.55624, reward=5.93750, avg_score=2483.64691
maxscore:7392
avg_score:2483.6469064382013
我们最终版本代码运行结果如下：
loss=17.00991, Q=109.20171, reward=200.00000, avg_score=4026.19920
maxscore:10952
avg_score:4026.1992000784
有较为明显的提升。
代码附在 q1.diff 文件中。