星际争霸强化学习报告

王帅1800012995 区子锐1800012945　王鸿铖1800012939

一、作业内容简介

在星际争霸的七个小游戏当中，我们达到了如下成果

	MoToBea	CollMineSha	FiAndDeZerg	DeRoa	DeZergAndBan	CollMinsAndGa	BuiMari
ours	25.87	65.12	39（16.7，23.9）	98.16	63.8	3637.5	11.26（36.8，47.53）
deep	25	96	44	98	62	3351	<1
百分比	**103.4%**	67.83%	88.64%（54.3%）	**100.1%**	**102.9%**	**108.5%**	**1126%**
使用的网络与算法	FullyConv+TD3	FullyConv+TD3	Rulebase(Relational+A3C)	FullyConv+TD3	Relational+A3C	Relational+A2C	Relational+A3C(TD3,A2C)

在1、2、4、6五张图中，我们加入了一定程度的人类经验，手动掩盖了一些动作。在3图当中，我们写了一个rulebase agent，获得了39分，但是A3C方法的只有16分左右，换用PPO之后拿到了23.9分的成绩。在5图中，我们没有加入任何人类经验。在图7中，不加入任何人类经验拿到了11.26分，加入人类经验拿到了36.8分，加入少量人类经验和reward_guided拿到了47.53分。

我们在作业当中，使用了FullyConv和FullyConv+Relational两种网络架构，DDPG、TD3、A3C、A2C、PPO五种算法（或者说四种，A2CA3C本质上一样）。以上的成果是在这几种网络架构和算法的组合之下完成的。

二、网络架构

0、对动作的编码

无论是用Policy-based还是用Value-based的算法，如何对动作空间进行编码都是一个必须要解决的问题。我们在经过讨论以及尝试之后，采取了如下的两种编码方式

1.只对主动作$base_action$以及空间参数$spatial_args$进行编码：

对于$base_action$采用one_hot的方式进行编码，而对$[screen, minimap, screen_2]$等等位置参数，例如对于一个$screen = [x_{screen}, y_{screen}]$的参数，将其编码为$x_{screen} \cdot Nx_{screen} + y_{screen}$，并用一个列表来顺序存放参数。

对于其他的参数$nonspatial_args$，我们直接赋值0, 并且不考虑其作用。

这种编码的好处在于简单直接，对于$MoveToBeacon$等简单任务能进行学习；坏处在于无法用这些动作来产生更加复杂的策略。

2.对主动作$base_action$,空间参数$spatial_args$以及非空间参数$nonspatial_args$都进行编码

为了使动作空间更容易处理，我们将每一个动作对应的参数补充完整，最终的形式为$(base_action, spatial_args, nonspatial_args)$例如：

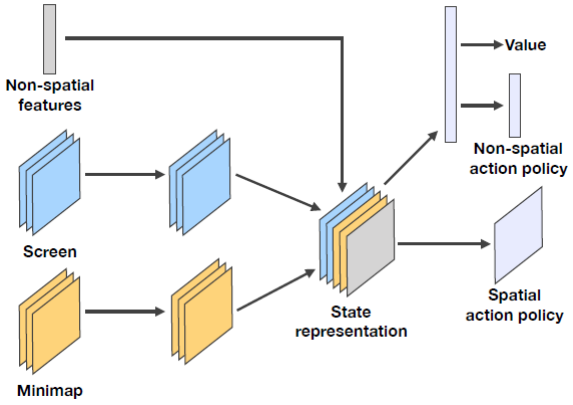
对于$no_op, [\ \ ]$，我们用$(0, [-1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1])$来进行编码，

对于$Move_Screen, [\ [0],\ \ [4,\ \ 4]\ \ ]$，我们用
$(0, [4 \times 64 + 4, -1, -1], [0, -1, -1, -1, -1, -1, -1, -1, -1, -1])$ 来进行编码

这种编码的好处在于无论是采用On-policy还是Off-Policy的方式，我们都可以更加容易地求解值函数，求解loss；坏处在于，由于大部分的动作其实只需要少部分的参数，换而言之，动作集合中的参数其实是很稀疏的，这样可能会在训练时增加内存/显存负担。

1、FullyConv

该架构直接参考deepmind论文当中所提出的架构。



在实际使用当中，将screen和minimap的$features_{3D}$先通过两个卷积网络，得到$output_{3D}$。然后该$output_{3D}$再分别输入到$Actor\ Critic$网络当中。效果上来看，FullyConv网络只能学到地图的浅层特征（如Beacon的位置），并不能学到一些连串的动作逻辑；比如在图七当中，FullyConv并没有学到如何造$marine$；在图六当中也没能学会制造$CommandCenter$

同时对于reward比较稀疏、滞后的地图而言，该网络的探索的作用较低。

2、Relational

在论文《Relational Deep Reinforcement Learning》中所提到的Relational网络

Relational网络对各种特征进行提取后，再将得到的特征向量输入Actor网络与Critic网络，从而得到相应的动作概率以及在当前状态下的值函数的估计值。

Relational网络最重要的改进之一，是在卷积之后加上了多头注意力机制$MultiHeadAttention$，这个模块被称为Relational模块。普遍认为$Attention$能够学习实体之间的关系，进行关系推理，从而能够解决一些比较棘手的关系问题。我们在部分地复现了Relational网络后，在没有任何人工先验知识的情况下，仅仅利用A3C算法便能在最后一张地图BuildMarines上面取得不错的成果。

这里要对Relational模块进行介绍，其主要的方式是通过$MultiHeadAttention-MLP$的方式进行堆叠，其中一个$AttentionHead$指的是将N个实体的一个编码向量矩阵
$E=（e_1,e_2,...e_N）\subseteq\mathbb{R}^{m\times N}$先用三个线性变换矩阵进行处理，得到$Query,Key,Value$这三部分，其中

$$
\begin{aligned}
Q&=M_Q\times E\\
K&=M_K\times E\\
V&=M_V\times E\\
M_Q,M_K,M_V&\in\mathbb{R}^{d\times m}
\end{aligned}
$$

再用

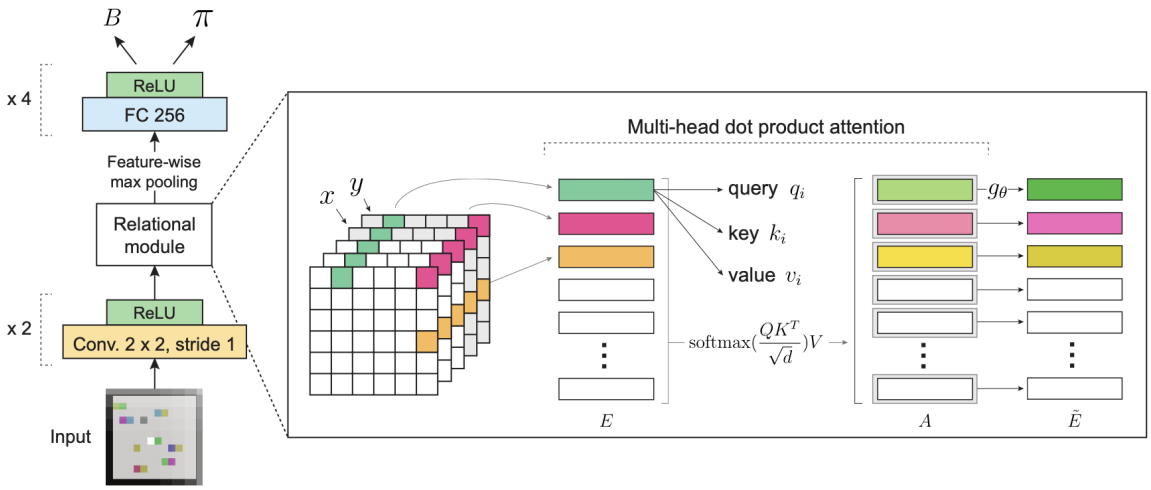
$$
\tilde{E}=Softmax(\frac{QK^T}{\sqrt{d}})V
$$

来得到对应实体的$Attention$编码，本质上是基于实体之间的关系权重，来进行编码的重新分配。不同的$Attention$之间的"关注点"很有可能是不同的，所以一般使用$Attention$都是通过多个头并行，将多个注意力机制模块得到的结果连接在一起，即

$$
\tilde{E}_{MHA}=MLP(concatenate(\tilde{E}_1,\tilde{E}_2,...\tilde{E}_k))
$$

其中$\tilde{E}_i$表示第i个Attention所输出的编码，这便是多头注意力最终得到的编码。

Relational模块在进行编码时，是要用相同的$MultiHeadAttention-MLP$模块来进行多次编码，叠加多次进行编码，很可能会产生类似于图神经网络中的"消息传递"的效果，使得实体之间的关系更加显著，从而使得关系推理成为可能。



Relational网络的架构如上，主要是如下的三个模块构成：

（1）状态编码：将原始的minimap, screen等等信息进行预处理：对于所有的分类变量，用Embedding层进行编码；而对于连续变量，我们采取取对数的方式将其数值减小。将预处理后的minimap与screen喂进两个顺序的残差网络块中，其中每一个块由一个核大小为4，步长为2的卷积层，以及2个核大小为3，步长为1的卷积层组成，其输出的channel数分别为16，32，32。在对minimap与screen状态编码后，将从原来$[channel_{in}, 64, 64]$的大小变为$[channel_{out}, 8, 8]$,其中$channel_{out} = 32$. 将通过残差模块后得到的minimap与screen在channel的维度进行粘贴，得到大小为$[\#channel_{minimap\ out} + \#channel_{screen\ out}, 8, 8]$的三维张量；将last_action通过Embedding后与player特征进行粘贴，并通过一个两层的MLP（每层的输出数为512，激活层为ReLU），得到向量$input_{2d}$。

（2）关系处理：处理后要得到的是$relational_spatial$以及$relational_nonspatial$两种不同的隐向量，分别用于$[screen, minimap, screen2]$坐标参数的选择，以及$base_action$，$nonspatial_args$的选择。我们将形状为$[channel_1, 8, 8]$的张量，先变为$[channel_1, 64]$的二维张量，再将张量放入Relational模块进行处理，其输出为一个$[channel_2, 64]$的二维张量。为了得到$relational_spatial$，我们将最后两维展开，得到的是形状为$[channel_2, 8, 8]$的张量；为了得到$relational_nonspatial$，我们对$[channel_2, 64]$中的第二维进行$max_pooling$，得到的新向量为$[channel_2, 1]$.

（3）输出处理：将$input_{2d}$与$relational_nonspatial$进行连接后，主动作的概率分布通过将这个新的向量输入到一个两层的MLP中获得，这个MLP的输出维度分别为256以及$\#base_actions$，激活函数为$ReLU$. 而当前状态的值函数则由一个两层MLP（输出维度分别为256，1，激活函数为$ReLU$)获得，其他的非空间参数的概率分布也是通过同样的方式获取。而对于空间参数$spatial_args$，则通过先对$relational_spatial$进行反卷积，得到大小为$[64, 16, 16]$的三维张量，再进行上采样获得$[64, 64, 64]$的三维张量后，通过$Softmax$层来获得位置参数的选择。

然而Relational网络在除了最后一个任务，在其他任务上表现，总体而言并没有在论文中那么惊艳，对此我们总结了一些可能的原因：

（1）原始的Relational网络相对比较大比较深，不采取LayerNorm/BatchNorm，残差链接等技巧的话，会在训练中出现梯度消失的现象.

（2）我们对于Relational网络的复现比较粗糙，由于实现得比较仓促，我们并没有对Relational加上ConvLSTM这个可能十分重要的模块。我们在实验中也观察到，不加入LSMT等等模块，可能会使agent产生"游移不定"的情况——例如，即使在训练的后期阶段，也偶尔地发生如下情况：在第k时刻向左走，在第k+1时刻又往右走了；或者是在两块矿石之间来回游走，却并不真正走近其中一块。这种毫无意义的动作会浪费agent进行探索以及完成任务的时间。

(3) 原始的论文《Deep Reinforcement Learning With Relational Inductive Biases》中使用了更为先进的算法IMPALA来进行计算，其核心在于利用并行的actor来收集数据，再返还给中心的learner由learner做off-policy的更新。总的而言，在这种相对复杂的任务中，off-policy算法要比on-policy的算法要更加的稳定，我们认为这也是Relational在我们的实验中相对不出彩的原因之一

(4) 可能是最重要的一点，由于算力，经费，时间，资源等等客观条件的限制，我们对于agent的训练次数远远达不到DeepMind这两篇论文中的训练次数。然而，从Relational的原文中我们可以看到，只有当训练的 episode数达到1e8的界限，对agent的训练才会有质的改变。在SC2LE与Relational中，所采用的进程数都远大于10个，我们并没有，也不大可能用这样的方式进行学习。我们在讨论后一致认为，这样的客观原因确实是不可忽视的。

三、算法简介

1、DDPG

DDPG全称是deep deterministic policy gradient，是AC框架的算法。

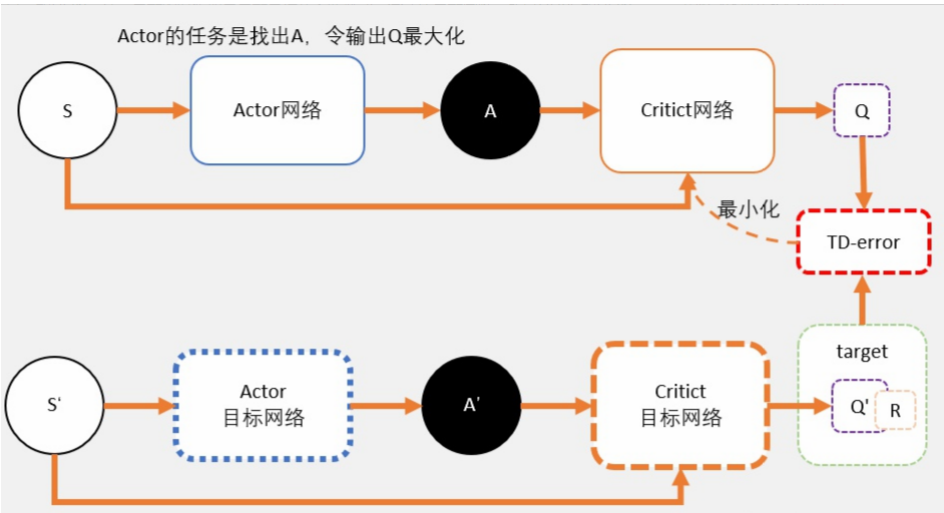
它的Critic网络需要输入动作和状态，然后传回一个$Q(s,a)$的值。Actor输出动作的概率，具体到星际争霸的环境里就是输出573个动作的概率，84*84个点的概率，然后选择一个动作和其参数。

可以得到Bellman方程为$Q^u(s_t,a_t)=E[r(s_t,a_t)+\gamma Q^u(s_{t+1},u(s_{t+1}))]$

Actor是要最大化度量值$J=\int_S p(s)Q^u(s,u(s))ds$

Critic的更新还是传统的TD-error。而Actor的更新是让Actor选出的动作是Q值最大的那个方向。

下图是运行的示意图



下图是DDPG的伪代码：

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2)$
    Update the actor policy using the sampled gradient:

$$\nabla_{\theta^\mu}\mu|_{s_i} \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

2、TD3

TD3全称是Twin Delayed Deep Deterministic policy gradient，和DDPG的差别有点类似于DDQN和DQN的差别。

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ
with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta_1' \leftarrow \theta_1$, $\theta_2' \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi(s) + \epsilon$,
  $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s) + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \min_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s,a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s,a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$
    $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
  end if
end for

从伪代码里可以看出，比起DDPG而言，TD3多了一个CriticQ网络，每次选择两个Q网络当中较小的那个：这就是Twin的来历。而且TD3的Actor网络并不是每次都更新，而是每个d个step才更新一次：这就是Delayed的来历。

从实际效果上看，TD3是远超DDPG的，因此在星际争霸游戏当中，我们直接弃用了DDPG。下面来分析一下为什么TD3远好于DDPG。

首先是两个Q网络，一般而言，从Q-learning起源的Q函数都会有过高估计的问题，double Q-learning当中把动作的选择器和评估器解耦来获得一个较好的缓解。但是TD3当中直接选择其中较小的一个，从一定程度上也能缓解过高估计的问题。

那为什么要delayed？在DDPG中提到，Actor的目的是找到Critic认为的最好的action。但是Critic在初期的估值不一定是正确的，也就是说，Actor好不容易找到最优动作，Critic就又更新出新的最高值了。所以让Actor更新的比Critic慢一些，让Critic学的更正确一点，Actor就能依从更好更正确的Critic来寻找最优的action。

3、PPO

尝试使用PPO2算法来进行model训练，PPO2是在TRPO的基础上发展而来，在Policy Gradient Methods中常用公式：$\hat{g} = \hat{E}_t[\nabla_\theta log\pi_\theta(a_t|s_t)\hat{A}_t]$进行梯度计算，其中$\pi_\theta$是随机策略，$\hat{A}_t$是t时刻的优势函数的估计，实际实现时目标函数为$L_{PG}(\theta) = \hat{E}_t[log\pi_\theta(a_t|s_t)\hat{A}_t]$,而在TRPO中训练的目标函数为:

$$max_\theta \hat{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] \tag{1}$$
$$\hat{E}_t[KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \leq \delta$$

其中添加了一个约束项，要求前后两个策略之间的KL散度小于某一阈值，但是用于求KL散度的θ_{old}是更新前策略参数的向量，因此要找到一个合适的值进行约束是比较困难的，这里我们用$r_t(\theta)$表示概率比则有$r_t(\theta) = \hat{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}]$,则TRPO优化的目标为$L_{CPI}(\theta) = \hat{E}_T[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_T)}\hat{A}_t] = \hat{E}_t[r_t(\theta)\hat{A}_t]$其中CPI意为conservative policy iteration,在没有约束项时最大化目标时可能会得到一个过大的更新，在PPO2论文中，通过惩罚使得$r_t(\theta)$远离1的策略更新，来达到对更新的约束效果，提出了目标函数：

$$L_t^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon))] \tag{2}$$

其中ϵ是一个超参数为裁剪比率，设置为0.2，以此将r_t限制在$[1-\epsilon, 1+\epsilon]$中，最终选择clip的r_t和没有clip的其中较小的与$\hat{A}_t$相乘得到最终的目标函数，在PPO中square-error计算对当前局面value估计的误差，记为L_t^{VF},通过entropy bonus进行增强促进探索，最后将三者相加得到PPO的loss function：

$$L_t^{CLIP+VF+S}(\theta) = \hat{E}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{3}$$

具体的代码实现上，对于advantage，实现了两种计算方式，分别为:

$1.\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T)$

$2.\hat{A}_t = delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-t}, \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_{t+1})$

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t))\right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

PPO利用重要性采样，解决了星际中采集到的数据在神经网络发生改变后不能重用的问题，一个episode中PPO可以训练多次，所以相比较A3C算法，PPO对于一局游戏的训练效率较高，而且由于训练过程中效果比较好的局面出现次数较少，因此对于一场效果好的游戏PPO能利用数据进行多次重复训练。就速度而言，在实验中训练速度相较于多线程的A3C明显较低。

由于PPO实现的比较晚，因此并没来得及训练出一个比TD3和A3C好的agent所以在7张图的最优结果当中并没有PPO。在作业当中，我们附加了PPO的代码。

4+5、A2C+A3C

在星际2学习环境中所采用的原始算法其实便是A3C，我们采用的是Actor与Critic共享参数的架构，在单个进程中，假设其在固定的时间t到$t+T$中，收集到如下的数据：

$$S=\{(s_t,a_t,r_t),(s_{t+1},a_{t+1},r_{t+1})\dots(s_{t+T-1},a_{t+T-1},r_{t+T-1}),s_{t+T}\}$$

用如下方式分别计算$A_i^\pi=A^\pi(s_i,a_i)=Q(s_i,a_i)-V(s_i)$

$$Q_i^\pi\approx\sum_{j=i}^{t+T-1}\gamma^{j-i}r_j+\gamma^{t+T-i}V^\pi(s_{s+T})$$

同样地，这也可以看作是对当前状态的价值函数$V^\pi(s_i)$的估计，从而用于计算Critic的loss的计算。

我们使用熵正则化的方式，来使得我们的agent保持一定的探索。

最后要优化的是如下的目标函数：

$$min_\theta\sum_{i=t}^{t+T-1}-(Q_i-V^\pi(s_i))log\pi_\theta(a_i|s_i)+\alpha\cdot(Q_i-V^{\pi_\theta}(s_i))^2-\beta\cdot Entropy(\pi_\theta(.|s_i))$$

在A2C中，我们使用$\alpha=0.5,\beta_0=10^{-3}$来进行训练。

在A3C中，我们使用10个进程进行探索，并对β采用线性减少的方式，使得最终学习能够收敛。

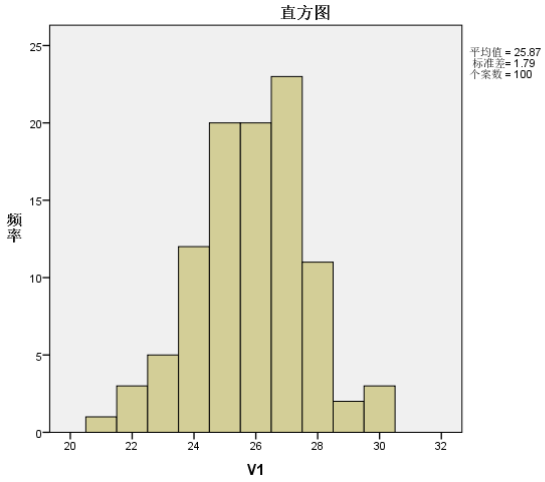
在训练过程中采用Adam优化器，用$10^{-3},10^{-4}$(FullyConv agent)，$10^{-4},10^{-5}$(Relational Agent)的学习率进行学习，每训练40-50轮进行10次测试，并挑选出性能较优的agent

四、训练过程以及地图总结

由于动作空间较大，我们组在实际训练过程当中，只保留了部分会影响reward的动作，其余同类型或者无影响的动作均被手动mask。对于剩余动作的选择和其参数（如地图坐标）的选择均使用强化学习的方法得到。

图一：MoveToBeacon

该图的任务是走到Beacon的位置。所需要的动作无外乎select和move两种。因此在实际操作当中，我们只保留这两个动作。该地图我们使用了FullyConv+TD3的配置。



从成绩分布可以看出，agent还是非常稳定的，得分聚集在24~27分之间，具体的得分差别还是得看Beacon的刷新位置。

第一张图的训练是比较简单的，在训练了20个episode之后就能比较合理的找到Beacon的位置。训练过程当中，我们先是使用了deepmind提供的scripted agent来产生行动数据，然后让TD3 agent进行学习；在学习了一定回合之后，TD3已经能稳定产生reward之后，让TD3 agent来产生数据和自己训练。之所以这样是因为如果让TD3直接进行训练，那么TD3很难拿到reward，产生的数据没有什么价值。经过scripted agent预训练之后，TD3上手就比较快捷。

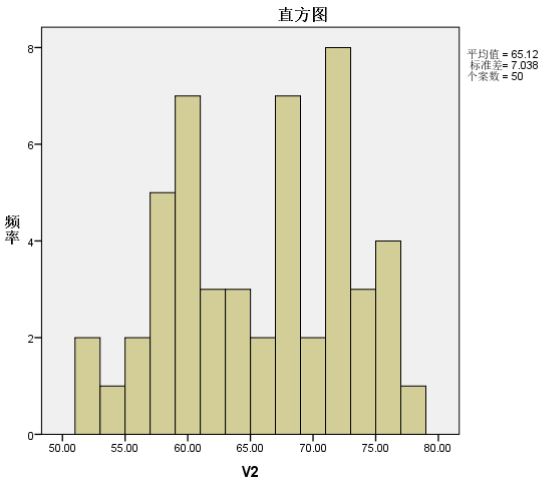
我们组之所以选择mask无关动作，是因为多个动作的会影响训练。在全动作下，agent一是会选择不到marine，于是marine会停滞在原地；二是会在move的时候，东点一下，西点一下，即使偶尔能选中Beacon的位置，也难以走到Beacon上。关于这里的第二个问题，我们考虑过让agent 对move这个动作执行"queued"而不是"now"，但是之后发现，动作会被堆积在一起，从而导致marine已经移动到了Beacon之后Beacon位置重新刷新，但是被堆积的动作仍然在执行，这些动作在这个时刻已经是废动作了（因为它们的生成与当前的Beacon无关）

图二：CollectMineralShards

该图的任务是依次收集$Minerals$。最好的策略是两个$marines$分开行动，比如分别收集屏幕左侧和右侧的$Minerals$。次一级的策略是一起行动，每次收集距离最近的$Minerals$。前者会带来的一个问题就是如何学习select动作，何时停下当前选择的$marine$去选择另一个来收集$Minerals$；后者会带来的问题就是，地图里的两个$marine$相当于被视为只有一个。

一开始我们组在训练的时候，开放了全动作。但是发现在select的学习上困难很大，一是由于图中的$features$过多导致选不中$marine$，而是一不小心就两个一起选中导致不能分开行动。即使选中之后，也会产生和地图一中一样的问题：东走一步，西走一步，然后在两个$Minerals$之间徘徊。

难以解决这样的问题之后，我们还是采用了模仿学习。使用了deepmind给出的scriped agent作为产生(s, a, r, s')的policy，然后使用FullyConv+TD3来学习这些数据。最终达到了scriped agent的大半性能。在test的时候发现，agent不会出现在两个$Minerals$之间徘徊的问题，但是它并没有学到收集距离最近的，因此最终的得分堪堪及格线。



统计

V2

个案数	有效	50
	缺失	0
平均值		65.1200
中位数		66.5000
众数		67.00
标准差		7.03820
偏度		-.087
偏度标准误差		.337
峰度		-1.162
峰度标准误差		.662
最小值		52.00
最大值		78.00

图三：FindAndDefeatZerglings

我们组认为这张地图是最难的。首先是地图的规则（击败所有怪物才能刷新下一轮怪物）要求我们必须对整张地图都探索到。

但是这会带来如下几个问题：一是如何保证把地图探索完毕，很多死角里面如果有怪物，而agent没能探索到，得分会卡在23、24分；二是如何减少重复探索，由于时间限制，重复探索势必会带来很多时间上的浪费。

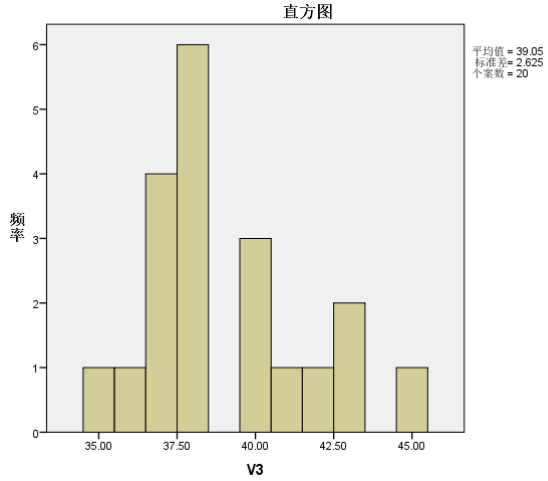
关于第一个问题，我们初期是想让3个$marine$分开行动以增加探索视野，但是发现这会被怪物直接击败。在观察了如何击败怪物后，我们发现，三个一起行动时，面对单个怪物可以刚刚保证无损击败，也就是说，如果少一个$marine$，那么很有可能会受到伤害而阵亡。因此关于探索，我们倾向于三个一起行动来探索。那么如何保证探索完毕？这个问题我们组无法解决，只能期待于agent自己学到。

关于第二个问题，减少重复探索。我们组在讨论过后，希望使用一个GRU来记忆之前走过的路，但是效果不佳。我们对此的解释是地图的features太像了，导致学不到什么记忆。

最终我们用Relational+A3C/TD3不加人类经验训练出来的agent普遍难以突破24分，也就是难以探索完整张地图。

之后我们认为将整个地图分成四块，那么只需要四个视野就足够覆盖整个地图，于是我们人为添加一个逻辑：当前视野探索完后，拖动窗口。窗口的坐标也被我们认为固定为pposx =[19, 19, 43, 43] pposy = [30, 50, 30, 50]。这样只要保证在当前窗口下完整探索，然后调整窗口即可。

但是一旦添加了这个逻辑，整个agent的RL浓度就急剧下降，我们几乎为agent做了最重要的一部分——探索。于是就有了下图的成绩表示。

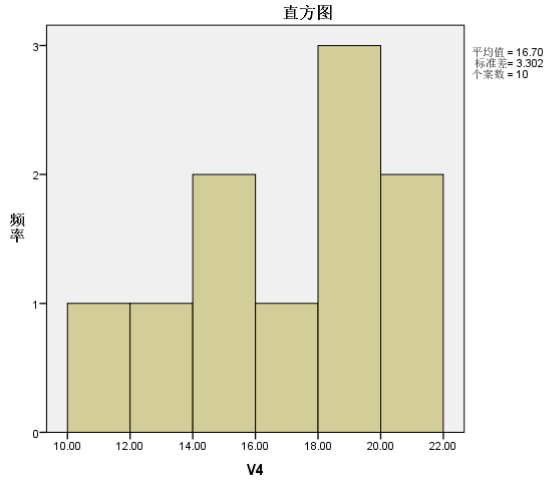


统计

V3

个案数	有效	20
	缺失	30
平均值		39.0500
中位数		38.0000
众数		38.00
标准差		2.62528
偏度		.754
偏度标准误差		.512
峰度		-.093
峰度标准误差		.992
最小值		35.00
最大值		45.00

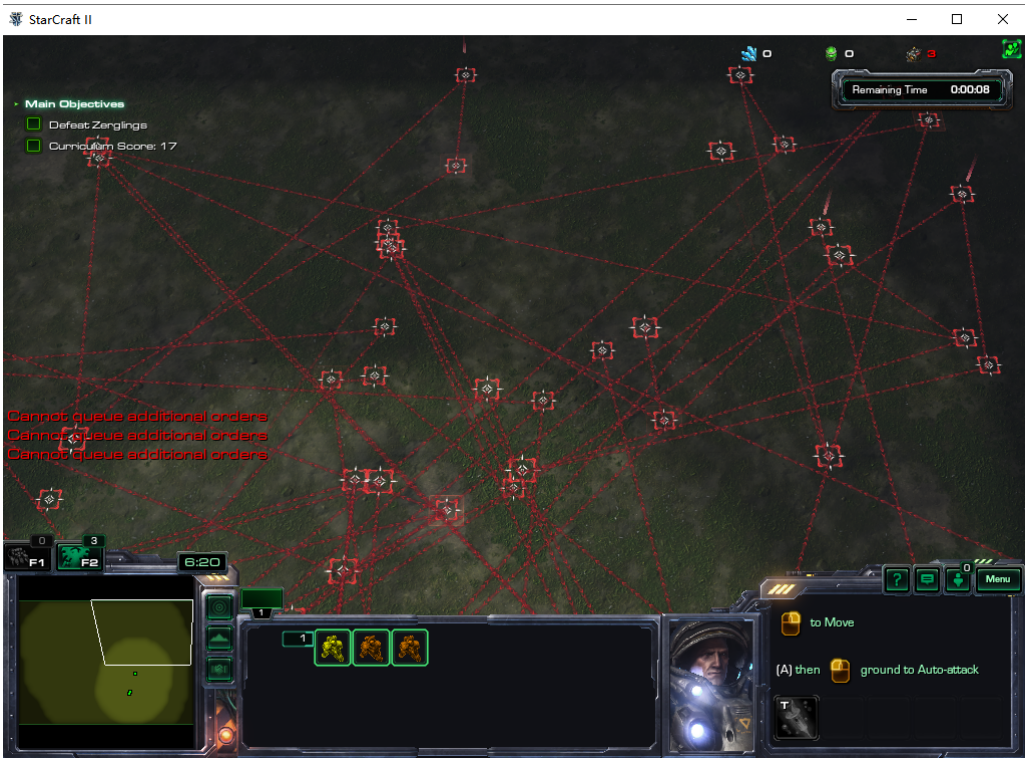
同时我们将纯Relational+A3C的结果贴在下方以供对比。



个案数	有效	10
	缺失	40
平均值		16.7000
中位数		17.0000
众数		15.00[a]
标准差		3.30151
偏度		-.272
偏度标准误差		.687
峰度		-.707
峰度标准误差		1.334
最小值		11.00
最大值		21.00

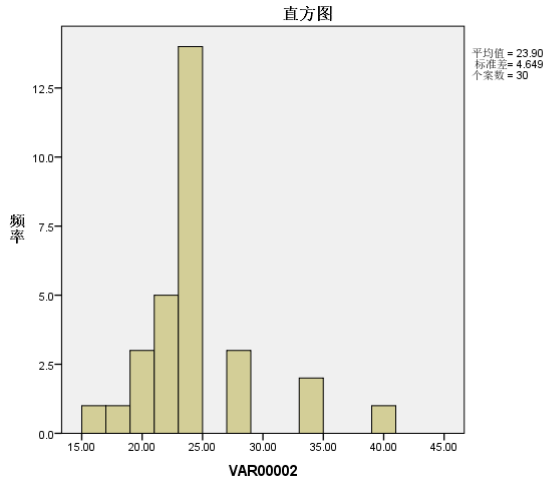
a. 存在多个众数。显示了最小的值。

纯A3C的游戏过程如下图，在游戏即将结束的时候，仍然有四个角落无法探索到，这就是限制突破24的最重要的原因。



随后我们尝试了利用rulebase agent进行产生数据来学习。然而agent没有学会探索。

随后又利用PPO训练第三张图，由于agent只有在消灭25只虫族后才能进入下一张图，所以训练时常常由于不能完全消灭地图中的虫族而导致训练结果卡在22、23之间，我们尝试使用了reward reshaping的方法，将reward达到20之后的奖励放大，并且根据minimap中的visual信息计算地图中每个坐标的状态改变频率，当频率过低时给agent一个负值，以此来驱动agent进行地图的探索，但最后训练结果没有明显的提升。结果大致如下。



统计

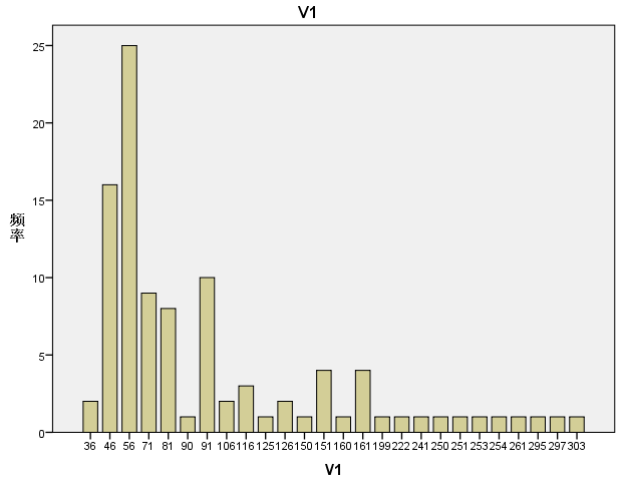
VAR00002

个案数	有效	30
	缺失	0
平均值		23.9000
中位数		23.0000
众数		24.00
标准差		4.64869
偏度		1.452
偏度标准误差		.427
峰度		3.344
峰度标准误差		.833
最小值		16.00
最大值		39.00

图四：DefeatRoaches

该图的任务是击败怪物。有一个特点是每次将地图中的怪物击败后会补充兵力，所以这会产生一个滚雪球的效应：如果初期打的好，士兵剩下的多，那么中后期就是几十个兵来打$Roaches$，于是分数很容易积累到200。因此整个得分的方差会很大，分布图呈现左偏态。

为了让尽可能多的士兵在一次战斗中活下来，较好的策略是尽可能集火其中一只$Roaches$，然后逐渐积累兵力来获得更大的胜利。然而更好的策略是使用1个士兵来吸引$Roaches$的火力，剩余的士兵进攻，这样每次都用一个士兵的损失来获得更好的reward。



统计

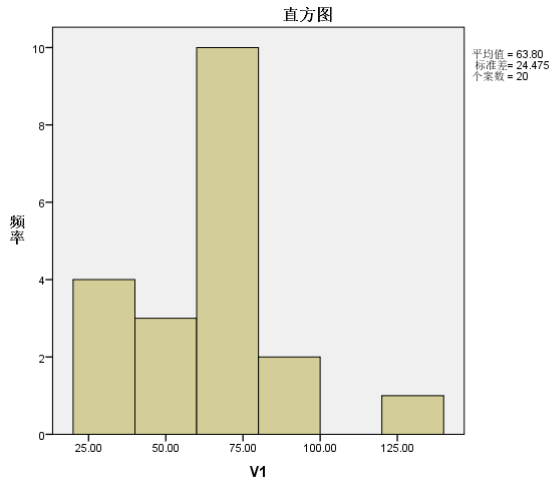
V1

个案数	有效	100
	缺失	0
平均值		98.16
中位数		71.00
众数		56
标准差		65.862
最小值		36
最大值		303

我们试图让agent学会更好的策略。通过大量的探索（A3C）来探索这种比较特殊的策略可能性比较低且难以学会，如果用一个rulebase agent来进行模仿学习，出于对pysc2环境的不熟悉，也很难写一个这样策略的agent出来。最终我们使用了FullyConv+TD3和deepmind提供的集火策略scripted agent来进行预训练，随后再进行自己学习。

图五：DefeatZerglingsAndBanelings

该图的任务是击败两种怪物。我们认为需要先集火一只怪物，不能分散火力。实际操作当中，我们使用了Relational+A3C的组合，并且没有mask任何动作，即训练过程当中使用的动作维度为环境给定的动作。



统计

V1

个案数	有效	20
	缺失	0
平均值		63.8000
中位数		68.5000
众数		72.00
标准差		24.47469
偏度		.116
偏度标准误差		.512
峰度		1.060
峰度标准误差		.992
最小值		21.00
最大值		123.00

从结果上可以看出，该图的结果并不稳定，标准差较大。中位数和众数都大于deepmind的最差结果。

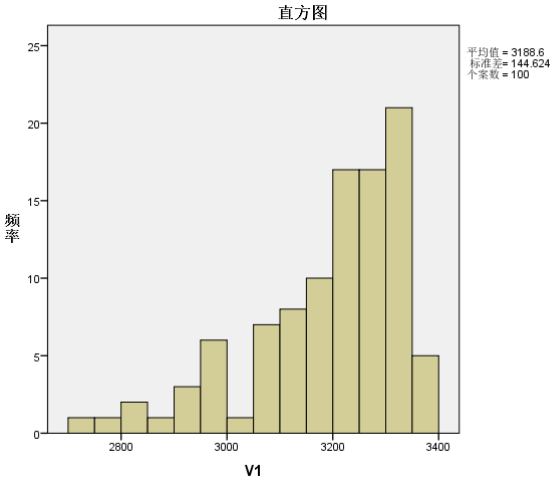
一开始，我们使用TD3进行训练的时候，发现很难集中击杀一个怪物，也难以学到一个比较好的策略（集火一个怪物）。即使对其mask动作之后，也无济于事；之后分析原因的时候发现主要是因为mask动作并不能改变attack这个动作附带参数依旧难以学到刚刚提到的较好的策略。对于参数的学习我们认为要么是给出比较好的数据（如rulebase agent）来进行学习，要么就从大量的采样当中学到。因此我们之后改用A3C来进行训练。

在换用A3C之后，由于该地图并没有mask任何动作，也没有一个合适的rulebase agent来进行预训练。于是我们直接在A3C，开了16个线程进行训练；一个晚上过后，均值基本稳定在52、53附近；但是模型的标准差还是很大。由于经费有限，因此我们停止了训练。

图六：CollectMineralsAndGas

这张地图经过我们的分析，认为如果纯采矿是可以达到3200分的程度，这我们猜测大概是deepmind两个较差结果的策略。而如果开始造一定的SCV，就可以获得超过3500的得分，这大概是deepmind最好的结果。在我们的参考论文使用的Relational网络当中，获得了7000余分的成绩，我们猜测是学习到了建造一个新的$CommandCenter$。

因此我们组的思路是先写一个纯采矿导向的用来保底交作业，然后开始思考如何引导制造SCV和$CommandCenter$。下图就是一个纯采矿导向的agent的成绩。



统计

V1

个案数	有效	100
	缺失	0
平均值		3188.60
中位数		3235.00
众数		3305
标准差		144.624
最小值		2710
最大值		3380

在训练的初期，我们人为提高一些动作的概率（如建造SCV）。但是我们发现很多问题，比如要想建造$CommandCenter$，它的位置限定很死，就只能在一块很小的可行域上建。

我们理一下逻辑链开采$Minerals$ → 选择合适的区域 → 建造$CommandCenter$，然而即使$Minerals$足够，agent也会更倾向于建造$SupplyDepot$而使用掉$Mineral$导致无资源建造$CommandCenter$（我们试图手动降低建造$SupplyDepot$的概率）。当我们手动mask了中期之后建造$SupplyDepot$的对应动作，发现$CommandCenter$已经没有位置建造了。也就是说，如果要用建造$CommandCenter$来提高reward，那么我们实际的逻辑链并没有那么简单。如下图，要想找到合适的位置制造司令部很难，如果不在这里加入人工经验，我们认为学习位置是一件很难的事情。

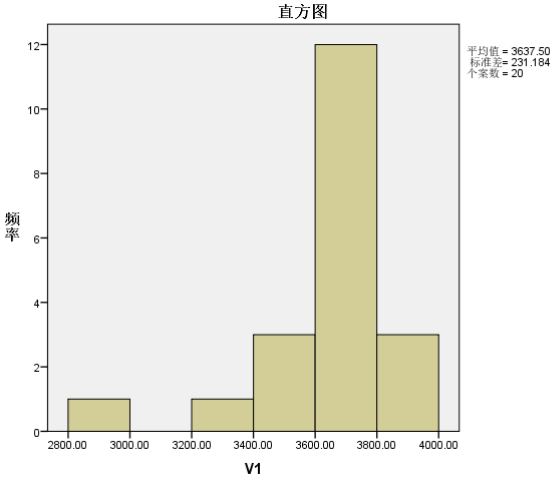
在随后尝试过程当中，我们直接给agent提供了$CommandCenter$的建造坐标，因此我们的agent能造出$CommandCenter$，但是造出的时间点太晚、agent难以学到如何分配SCV在两个$CommandCenter$上交接导致它的效率并没有得到很好的提升。另一个问题是$CommandCenter$造出的SCV很难参与到挖矿当中，它们就停在$CommandCenter$附近（如上图$CommandCenter$旁边那个被指出的SCV）。

于是我们组将目光放到了制造SCV。我们的agent确实学到了制造SCV，但是随之而来的问题是，额外制造出的SCV很难被select，导致制造出来没有意义。在加入了人类经验（select动作的区域）之后，额外制造出的SCV也能参与到挖矿。但是我们训练次数较多之后，发现agent沉迷于select而不去采矿，reward一直为0。我们分析是因为select之后才能挖矿，所以即使有$discount\ \gamma = 0.99$，select这个动作本身因此的价值依然很高。

随后我们将调小至$discount\ \gamma = 0.9$，并且对一些动作的参数和动作mask时机进行一定的约束。同时我们发现agent很喜欢建造$SupplyDepot$，而且到训练到后期疯狂造兵而不知采矿。于是我们人为的给动作一些reward来引导agent采取某些动作。

```
if base_act == 44:
    Comre /= 1000 //第一次建造CommandCenter我们给予大量的reward(1万)，之后就基本无reward。（这里初始值为10000000，所以除以1000之后为1万）
else:
    Comre = 0
if base_act == 490:
    SCVre = -10
else:
    SCVre = 0
if base_act == 264:
    Havre = 200 //采矿动作也有额外的奖励，这是为了避免后期无限select
else:
    Havre = 0
if base_act == 91:
    Supre *= 20 //SupplyDepotReward初始值为-100，我们认为造1个就足够采矿用了。
else:
    Supre = 0
```

最终训练出了满意的agent



统计

V1

个案数	有效	20
	缺失	0
平均值		3637.5000
中位数		3685.0000
众数		3645.00[a]
标准差		231.18402
偏度		-1.935
偏度标准误差		.512
峰度		5.187
峰度标准误差		.992
最小值		2895.00
最大值		3960.00

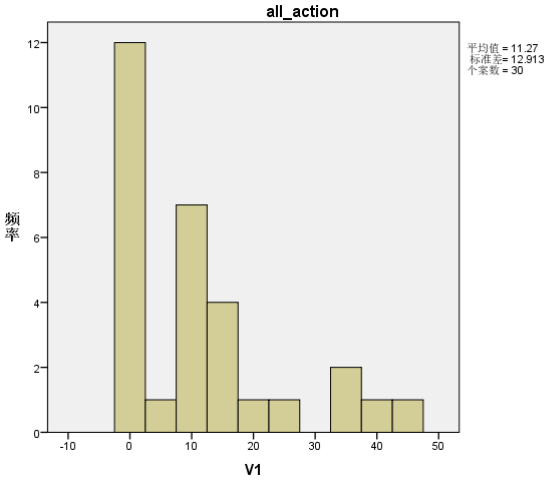
a. 存在多个众数。显示了最小的值。

图七：BuildMarines

这张图的逻辑链其实很长，挖矿 $\rightarrow$ 制造$Barracks$ $\rightarrow$ 制造$Marines$，这是主分支，期间还需要制造$SupplyDepot$。同时reward很稀疏，只有在最后制造出了$Marines$后才会获得。我们的思路有三条，一是通过大量的探索来获得正确的逻辑，二是通过mask动作来人为设定逻辑链，三是给一些中间环节也设定一些虚假的reward以驱使它去制造。

我们尝试了三条路，而且均获得了超过deepmind最好结果（6分）的成绩，详细分析如下。

all_action



all_action

V1

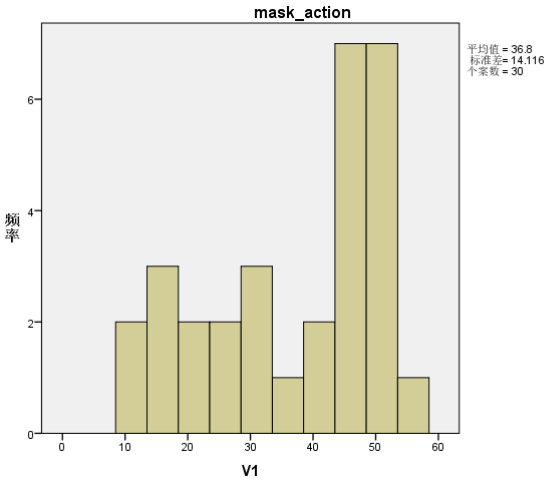
个案数	有效	30
	缺失	0
平均值		11.27
中位数		8.50
众数		0
标准差		12.913
最小值		0
最大值		43

上图是第一条思路的结果。我们使用了Relational+A3C的方式，直接进行大量探索。在得分分布当中，可以发现很多时候都会爆0，说明这个学习得到的逻辑链并不稳定。第二众数集中在10附近，说明还是能在特定的情况下学到了上述逻辑链。在训练的初期本机跑实验的时候，我们发现它训练初期会造很多$SupplyDepot$，以致于$Barracks$很少出现。当放到服务器上跑实验的时候，我们发现训练结果不能复现，也就是不能保证每一次训练都能训练出模型；很多时候开多个tmux同时跑多个A3C，会偶尔有一个两个A3C能获得reward，进而学到逻辑链。

该方法不稳定的原因在于它会采用很多不必要的动作，既不去采矿也不去建造，导致时间被浪费在随意游走上。这一方面是训练量不够所带来的弊端，另一方面是

所以如果将不必要的动作mask，我们认为效果会很好。于是我们测试了mask动作之后的情况如下。

mask_action



统计

V1

个案数	有效	30
	缺失	0
平均值		36.80
中位数		43.50
众数		51
标准差		14.116
最小值		11
最大值		54

上图是第三条思路的结果。我们使用了Relational+TD3的方式。在mask掉很多不必要的动作和参数之后，可以发现该模型能稳定产生$Marines$。我们组认为过多的mask动作训练出的模型并不优雅，使用的人类经验过多。该方法的测试的时候，agent在前5分钟会不停建造$SupplyDepot$，在后5分钟开始建造$Barracks$，最后才开始制造$Marines$。这一方面是因为Relational网络确实对于潜在逻辑关系有着FullyConv不可比拟的优势，另一方面是因为动作少了，逻辑学习更加容易，最后的难度就在select Barrack上。

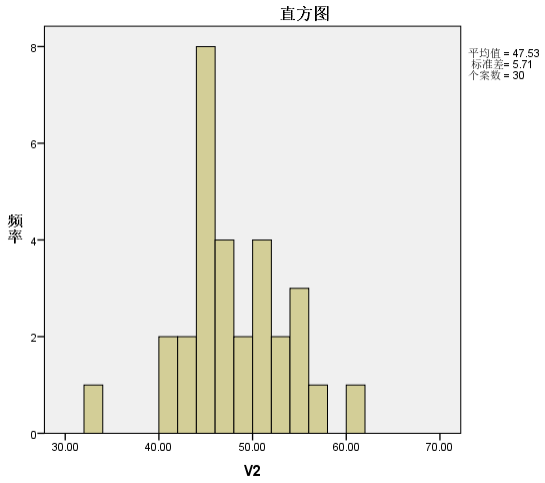
reward_guided

由于mask动作所起到的效果过于强大，我们大幅度地降低了mask动作的含量，并添加了额外的reward引导，希望通过额外的reward来敦促agent建造$Barracks\&Marines$。

```
if base_act == 42:
    Barre = 500
else:
    Barre = 0

if base_act == 477:
    Marre = 10000
else:
    Marre = 0
if base_act == 264:
    Haver -= 50
if base_act == 490:
    SCVre = 30
else:
    SCVre = 0
```

通过reward来学习逻辑链我们组认为是一种比较优雅的方法。这样解决了原本该地图下reward稀疏的问题，而且学出来的逻辑链更加稳定、快速。如下图的，最终得分达到了47分，而且更加稳定。



统计

V2

个案数	有效	30
	缺失	0
平均值		47.5333
中位数		46.5000
众数		44.00[a]
标准差		5.70984
偏度		.032
偏度标准误差		.427
峰度		.469
峰度标准误差		.833
最小值		33.00
最大值		60.00

a. 存在多个众数。显示了最小的值。

五、分工

王帅：PPO（100%），TD3（100%），训练、调试、添加人工经验、reward_guided（10%），框架搭建（15%），报告第三部分中PPO撰写。

区子锐：A2C、A3C（100%），FullyConv、Relational(100%)，训练、调试、添加人工经验（10%），框架搭建（15%），报告第二部分Relational、第三部分中A2CA3C撰写

王鸿铖：DDPG（100%），训练、调试、添加人工经验（80%），框架搭建（70%），报告中第一、四部分全部以及第二部分中FullyConv、第三部分中DDPG、TD3的撰写