Hyper-parameter Tuning and Coreset Distillation in Reinforcement Learning

Anonymous submission

Paper ID

Abstract

深度学习以其强大的端到端学习能力和在不同任务上的泛化能力而在各个领域上得到了广泛的应用，在端到端的训练流程中，数据的质量和超参数的选取尤为关键，对于数据质量而言，高质量的数据可以让深度神经网络训练得更加平稳和快速；对超参数而言，合理的超参数可以让深度神经网络优化时向着正确的方向前进。而深度强化学习这一深度学习的分支对于这两点要求相对于普通深度学习任务而言更加严格，这不仅因为强化学习任务所使用到的数据需要通过智能体（也就是深度神经网络）和环境交互产生，产生数据的质量容易参差不齐；更因为强化学习任务的神经网络非常容易收敛到局部最优解，即如果不能在较短时间内获得有效的监督信号，智能体就会陷入局部最优，进而表现为不再积极的和环境交互获得新的信息，只依赖于已经获得的信息进行决策。因此我们尝试在强化学习任务中进行数据压缩和超参数自动调整，从而更好的辅助原本的强化学习算法。

1. Introduction

深度强化学习是强化学习中非常重要的一个分支，其对于无人驾驶，机器人控制，多智能体学习等领域都具有重要的现实意义，传统的深度强化学习 任务可以被抽象为一个马尔可夫决策过程(Markov decision process)，这个决策过程通常可以用一个四元组来表达，例如$(\mathcal{S}, \mathcal{A}, p, r)$。在这里 $\mathcal{S}$通常表示状态空间，也即智能体可能处于的所有状态，$\mathcal{A}$通常表示动作空间，也即智能体可以做出的所有动作，在我们此次的实验中状态空间和动作空间 都是连续的。p是一个$\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^+$的映射，表示在当前的状态下执行对应的动作，从而转移到接下来的某一个 状态的概率密度，值得一提的这里使用概率密度而不是概率的原因是状态空间是连续的，r通常代表一个有界的奖励值(reward)，这是环境能够给到的最直观的监督信号，也和我们要优化的目标紧密相关。

在深度强化学习中，通常的优化目标是最大化智能体所获得的累积奖励值，也即$\sum_t \mathbb{E}_{(s_t,a_t)\backsim\rho_t}[r(s_t,a_t)]$， 其中$\rho_t(s_t,a_t)$代表从某一状态的初始分布开始，并以$\pi(a_t|s_t)$作为策略生成的所有轨迹中t时刻对应$(s_t,a_t)$的总概率。这个目标直观上是希望智能体能够在 和环境的交互中尽可能多地获取奖励值，和人类在完成任务时的目标是一致的。在这个目标的驱使下，我们一般需要做的首先是在环境中进行一定程度的探索从而发现高奖励值的(s,a)对，然后再根据已有的对环境 的认识找到最优策略，也即在任何s上都能选出有助于最大化总奖励值的动作a。

1.1. Soft Actor Critic Algorithm

Soft-Actor-Critic [5]是深度强化学习中的一个baseline算法，它的主要思想是通过值函数决定每一个状态s下应该做出的动作a是什么，值函数的更新则是通过智能体与环境交互的结果和时序差分(Time-Differential) 的方式进行的。

具体的，SAC会维护两个值函数(通常也是用深度神经网络拟合得到)，一个是Q函数，$Q(s_t,a_t) = r(s_t,a_t) + \mathbb{E}_{s'\backsim p(s_t,a_t)}[V(s')]$，代表的是当前处于$s_t$下执行动作$a_t$可以在未来获得的 总的奖励值，另一个是V函数，$V(s) = \sum_t \mathbb{E}_{s_0=s,a_t\backsim\pi(a_t|s_t)}[r(s_t,a_t)]$，代表的是当前处于$s_t$下并以$\pi$作为策略生成的轨迹在未来所获得的总的 奖励值。明确了这两个值函数的含义之后，只需要使用时序差分的方法，自举地利用和环境交互的信息不断更新这两个值函数就可以获得相对准确的值函数估计。

在获得了相对准确的值函数估计之后，我们想要从值函数中推导出策略就只需要在任何一个状态下执行$Q(s,a)$值最大的行动就可以了，这是因为Q函数本身就代表了在当前状态下执行某个行为可以在未来获得的总的奖励值，但是事实上这么做通常是有问题的， 原因在于Q函数只是作为未来总的奖励值的一个估计值，实际上通过自举的方法学出来的Q和V函数的值都有很大的偏差，因此一个常见的缓解这种问题的做法是引入最大熵强化学习(Max Entropy Reinforcement Learning)，最大熵强化学习在要求所获得的总奖励值最大的基础上， 还要求策略具有一定的探索性，优化目标变为$\mathcal{J}(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\backsim\rho_\pi}[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))]$，可以发现优化目标除了希望总的奖励值大， 还希望策略π的熵大，这能够一定程度上保证策略的探索性，从而避免由于值函数带来的偏差而导致策略过分相信值函数，做出的行为并不是实际上的最

优行为。

一般来说，只要能够保证策略具有一定的探索能力，随着策略和值函数训练过程的交替进行，策略和值函数都会越学越好，是不容易陷入局部最优值的。但是如果α 的选取不正确的话就会产生十分严重的后果，α太小会导致策略过分相信值函数，从而容易陷入局部最优，α太大会导致策略过分强调探索，从而很难收敛。 因此α的选取对于Soft-Actor-Critic是至关重要的，这也正是为什么我们选择在这个算法上进行超参数学习的辅助。

1.2. Hyper-parameter Tuning

上文提到，SAC算法中α的值至关重要，事实上α决定了最终策略的形式，也即决定了深度强化学习中有关策略的监督学习部分的目标，事实上最大熵强化学习的目标策略可以写作$\pi^{\star}_{MaxEnt}(a_t|s_t) \propto exp\left(\frac{Q^{\star}_{soft}(s_t,a_t)-V^{\star}_{soft}(s_t)}{\alpha}\right)$， 由于$\alpha$是指数内部的因子，$\alpha$的变动会极大影响目标策略，哪怕是$\alpha$很小的偏差也可能导致算法直接失效。

目前主流的超参数学习算法可以通过贝叶斯估计，Bandit算法求解，Meta Learning等框架实现，本次实验我们选择的框架是Meta Learning的框架，这不仅仅是因为Meta Learning可以受益于深度神经网络强大的拟合能力，更是因为Meta Learning相比于其他经典算法 具有更好的泛用性，可与更好地结合到本次的强化学习算法中。我们主要基于Bootstrapped Meta Learning的框架进行超参数学习，这是Meta Learning领域内最近提出的一个超参数学习算法框架，具体算法讲解会在下文详细展开。

1.3. Data Compression in Buffer

在现如今的模型训练过程中，数据集的样本选择是非常关键的环节之一。特别是深度强化学习此类在训练过程中采样的学习方式，如何选取采样后样本进行学习的方法可以直接影响训练时间、损失值的收敛程度和最终效果、模型表现等等。

深度强化学习模型的样本选择方法通常为基于Replay Buffer的数据采样。Replay Buffer中存储的是模型在此前的训练过程中所获取到的样本信息，其中每个样本包含(s_t, a_t, r_t, s_{t+1}) 即t时刻的状态、选择的动作、获得的奖励和过渡到的$t+1$时刻状态。在完整的训练流程中，智能体结合探索机制，从buffer中的样本进行采样，使用历史样本信息组来决策下一步的行为，将探索得到的环境反馈信息再次存入buffer中。随着训练轮数的增加，buffer中的样本不断扩大，智能体也逐渐掌握环境反馈的分布特征，减小探索比重、增大经验价值，从而学习到更优的响应方案。

不难发现，当训练轮数上升，buffer所需要的空间也越来越大。特别是在复杂的任务中，智能体如果需要达到较高的表现水平，消耗的缓存空间也是非常大的。这在一方面导致了存储空间的占用，尤其不利于没有后端支持的小型真实环境机器人体，它们通常使用轻量级的芯片等实现，存储空间较小。

另外一方面，智能体在学习中，通常较近的知识具备更高的参考价值，这是由于最近的行为隐式包含先前学习到的知识信息，通过总结先前经验分布、选取更优蓝本的方法实现了行为优化迭代。因此随着训练的进行，前期的样本重要性下降，有理由对其进行进一步筛选压缩，在节省空间的同时改善buffer中样本比例，给予后期经验相对更高的分布值。

由此，在训练过程中，阶段性地对buffer进行数据压缩，可以在轻量化存储空间地同时改善优质样本分布，提高模型效果。

2. Related Works

本次试验的相关工作主要从三个方向进行展开，首先是介绍深度强化学习方向的一些相关工作，然后是介绍超参数学习方向的一些相关工作，最后会介绍数据压缩方向的一些相关工作。

首先对于深度强化学习而言，本次实验的主要强化学习算法框架是SAC和discrete-SAC，前者在上文也有提到，后者则是SAC算法在离散状态空间和动作空间下的一个变种。SAC [5] 作为深度强化学习领域内标志性的突破，不仅以off-policy的方式极大提升了sample-efficiency，实验也证明SAC在很多环境下都能有很不错的表现，比如mujoco环境 [8]，D4RL环境 [3]等等。

正因为SAC具有如此强大的泛用性，我们才可以以SAC为基础，尝试以SAC中的一些参数为待学习的超参数进行超参数调整，而Tuomas Haarnoja等 [4]指出，SAC算法中作为temperature parameter的α 至关重要，直接影响了最终策略的形式，因此我们此次实验主要探究把超参数学习的算法加入传统SAC算法之后对于算法各方面性能的提升，我们希望的结果是使用了超参数学习的方法之后SAC算法可以在不同任务上完成α的自适应，从而更好地平衡智能体对于探索和利用现有知识的侧重。

超参数学习是早在深度学习兴起之前就已经有所发展的方向，在深度学习出现之前，超参数学习往往是通过一些传统算法，例如把超参数学习看成一个Bandit的优化问题去求解，即便是深度学习出现之后，这种Bandit的优化方法 [6] 也是屡见不鲜。除了这些传统算法之外，Meta Learning是当下超参数学习的主流框架之一，Meta Learning侧重于把超参数学习的过程和深度学习端到端的特性相结合，利用超参数在优化过程中产生的梯度信息结合优化目标 进行超参数的优化，具体的，Zhongwen Xu [9]首次在强化学习中引入Meta Gradient的超参数学习方法，即把整体的优化目标看作二级的优化过程，当二级的优化过程执行了数步之后，整体的优化目标相对于超参数而言 也能计算出对应的梯度，这个时候再执行一级优化过程，也即对超参数进行以整体优化目标作为优化目标的梯度下降。除了Meta Gradient之外，MAML [1]这类通过多任务ensemble从而找到最合适的初始化参数的位置的方法 也在Meta Learning中占有举足轻重的地位，只是由于我们此次任务并不需要用到超参数初始化，所以我们在实现的时候并没有考虑用MAML进行超参数

学习。

虽然Meta Gradient给了我们一个泛用性很高的框架来优化超参数，但是Meta Gradient算法在执行的过程中可能遇到近视，整体优化目标过于困难等问题，进而影响超参数的学习，考虑到这些实际的问题之后，Sebastian Flennerhag [2] 提出了Bootstrapped Meta Learning来解决上述问题，BMG相比于MG而言有着更好的泛化性，作者在文中证明了特定情况下BMG可以退化为MG，并且可以一定程度上缓解近视和整体优化目标的对于超参数优化的影响，尽量将超参数优化的过程和 整体目标的优化过程分开，从而防止由于整体目标优化过于困难而导致超参数优化也过于困难的情况发生。

正是由于BMG不仅在理论上有一定的保证，同时在实际实现算法的时候也能受益于其泛用性，可以更好的结合到SAC中，我们这次实现的超参数学习部分使用的算法便是BMG，至于BMG的具体做法和我们在实验中的具体实现将在下一个版块进行详细的讲解。

在机器学习任务中，Incremental gradient(IG)方法通常用于大规模优化，需要通过数据压缩的方式从完整数据集中遴选出训练子集，从而高效地利用数据，同时尽量保证子数据集在理论上和应用中都与完整数据集表现基本相当。据此，CRAIG [7]提出了一种选择训练数据的加权子集的方法，通过最大化子模块函数的方式来密切估计完整梯度。文中证明了该子集在IG下能够收敛到最优解，并且将训练速度提升到了3 6倍。我们从此种数据处理方法中得到启发，尝试将其搬移至强化学习任务上，观察是否同样能够提升数据利用率。

3. Methodology

3.1. Bootstrapped Meta Gradient

前文中提到Meta Gradient虽然思想上简单易懂，但是在实际任务中往往会遇到两个严重的问题：近视和受限于整体优化目标。具体的，近视是指Meta Gradient是根据历史优化过程中整体优化目标关于超参数的梯度进行超参数的梯度下降，对于超参数的优化过程是没有用到任何和未来的优化过程有关的信息的，也就是说这么做只能保证更新之后的超参数有助于从过去的优化过程中的某一步优化到现在，而不能保证这样优化之后的超参数可以在未来也有利于整体优化过程，这个问题产生的根本原因是Meta Gradient没有用到除了历史优化过程以外的任何信息，要想克服这个问题就必须想办法把未来的优化过程的信息给用上。

受限于整体优化目标是指Meta Gradient算法中用来更新超参数的梯度是整体优化目标对于超参数的梯度，那么当原本的优化目标较为困难时，整体优化目标对于网络参数的梯度常常容易梯度消失或者梯度爆炸，或者没办法在正确的方向上对 网络参数进行更新，这时整体优化目标对于超参数的梯度也往往是畸形的，我们很难指望原本的优化目标对于超参数梯度的质量远远高于对于网络参数的梯度的质量。要想克服这个问题就必须要让更新超参数的梯度来源独立于更新网络参数的梯度来源。

巧妙的是，Bootstrapped Meta Learning利用自举产生超参数优化目标的方式解决了上述问题。BMG同样把整个优化过程分成两级优化，次级优化过程是对于网络参数的优化，在这一级的优化过程中也会相对应的记录和超参数有关的梯度信息，但是BMG对于第一级的优化方式选择了截然不同的一种，在次级优化过程进行了K步之后，我们考虑这时的网络参数为X^K，BMG会先根据某个网络参数到网络参数的映射$\xi : \chi \mapsto \chi$(对应Bootstrap这一操作)生成目标网络参数，在具体实现过程中这种网络参数到网络参数的映射通常是把某时刻的网络参数X^K映射到以相同的次级优化过程再优化L步所得到的网络参数X^{K+L}，也即$\xi(x^{(K)}) = x^{(K+L-1)} - \alpha\nabla f(x^{(K+L-1)})$。有了目标网络参数 之后，BMG会最小化当前网络参数X^K和目标网络参数之间的某种距离，也即$w' = \text{argmin}_w \mu(\tilde{X}, X^{(K)}(w))$，在实现的过程中距离$\mu$通常选用欧氏距离或者KL-div，且并不需要直接最小化两个 网络参数之间的距离，有时可以通过最小化两个网络输出的分布之间的KL-div来达到类似的效果。

直观上来说，BMG想要达成的效果是通过优化超参数，使得原本需要$K + L$步才能优化到的目标，使用新的超参数之后仅仅通过K步就能优化得到。从这个直观的角度来看，BMG显然理论上可以解决之前提出的两个MG的问题，就近视的问题而言，BMG由于 是最小化当前网络参数和未来优化产生的网络参数间的距离，如果未来的优化过程和过去的优化过程差别很大，BMG会优先考虑未来的信息，$K + L$步优化得到的网络参数可以提前告知超参数如何变化才能让未来的优化过程受益。就整体优化目标困难的情况而言，由于BMG中超参数更新的梯度来源不再是整体优化目标，哪怕是整体优化目标提供的梯度逐渐消失的情况下，L步网络参数的更新也可以为当前网络参数和目标网络参数之间拉开差距，从而更可能通过最小化二者之间的距离从而获得更好的超参数，进而帮助整体优化目标 脱离梯度消失的困境，这种独立的超参数优化目标在优化过程中是更加鲁棒的。

具体算法如下：

Algorithm 1 N-step RL actor loop

Require: $N, x \in \mathbb{R}^{n_x}, s$ ▷ Rollout length, Poilicy Parameters and Environment state
$\mathcal{B} \leftarrow (s)$ ▷ Initialize rollout
for $t = 1, 2, \cdots, N$ **do**
  $a \backsim \pi_x(s)$ ▷ Sample action
  $s, r \leftarrow env(s, a)$ ▷ Step forward environment
  $\mathcal{B} \leftarrow \mathcal{B} \cup (a, r, s)$ ▷ Add rollout to replay buffer
end for
return $s, \mathcal{B}$

可以发现在算法的实际实现中我们并没有直接最

Algorithm 2 K-step off-policy learning loop

Require: $N, K, x \in \mathbb{R}^{n_x}, v \in \mathbb{R}^{n_v}, w \in \mathbb{R}^{n_w}, s$ ▷ Rollout length, meta-update length, policy, value function, meta parameter and environment state
for $k = 1, 2, \cdots, K$ **do**
$s, \mathcal{B} \leftarrow ActorLoop(x, s, N)$ ▷ Algorithm 1
$(x, v) \leftarrow \phi((x, v), \mathcal{B}, w)$ ▷ Policy update step, here is SAC
end for
return $s, x, v, \mathcal{B}$

Algorithm 3 Soft-Actor-Critic with BMG

Require: $N, K, L, x \in \mathbb{R}^{n_x}, v \in \mathbb{R}^{n_v}, w \in \mathbb{R}^{n_w}, s$ ▷ Rollout length, meta-update length, bootstrap length, policy, value function, meta parameter and environment state
$u \leftarrow (x, v)$ ▷ Initialize policy and value function
while training not end **do**
$s, u(K), \leftarrow InnerLoop(u, w, s, N, K)$ ▷ K-step inner loop, Algorithm 2
$s, u(K+L-1), \mathcal{B} \leftarrow InnerLoop(u^K, w, s, N, L-1)$ ▷ L-1 step inner loop, Algorithm 2
$\tilde{u} \leftarrow u^{K+L-1} - \gamma \nabla_u l(u^{K+L-1}, \mathcal{B})$ ▷ Gradient step on objective l for one step
$w \leftarrow w - \beta \nabla \mu(\tilde{u}, u^K(w))$ ▷ BMG outer step
$u \leftarrow u^{K+L-1}$ ▷ Continue from the most resent parameters
end while
return u

小化目标网络参数和优化K步时得到的网络参数的距离，而是利用replay buffer $\mathcal{B}$中的数据，把一定数量的状态输入到策略网络中，然后最小化目标网络和当前网络在这些状态下的 动作分布。事实上这里可以选用的计算距离的方式有很多种，我们只是选用了比较便于实现且效果还可以的一种。其次就是内部循环中的SAC的具体更新流程并没有在这里给出，因为过于繁琐，并且这一部分算法 [5]可以在原文中查看。

3.2. CRAIG

CRAIG的算法尝试从完整数据集V中提取出一定数目的子数据集$S \subseteq V$，其中S的每个元素s_j均有对应的stepsize γ_j以衡量其在整个完整数据集分布空间上所代表的大致梯度（误差值为ϵ），并且针对所有可能的优化参数$\omega \in \mathcal{W}$。

$$S^* = \text{argmin}_{S \subseteq V, \gamma_j \geq 0 \forall j} |S|, s.t.$$
$$\max_{\omega \in \mathcal{W}} || \sum_{i \in V} \nabla f_i(\omega) - \sum_{j \in S} \gamma_j \nabla f_j(\omega) || \leq \epsilon \quad (1)$$

对于该子集S^*和相应权重值γ_j，可以确保梯度在子集上的更新与在V上的更新基本相同。这部分的理论证明具体见原文 [7]。

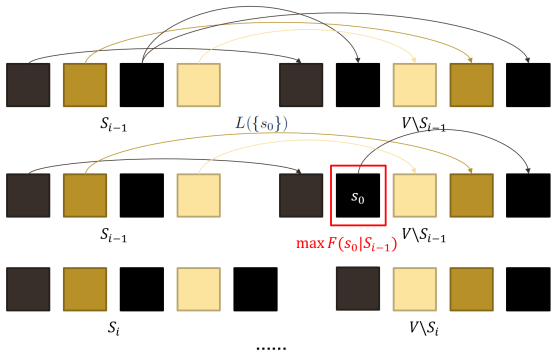


Figure 1. CRAIG迭代简要图示

在介绍筛选出子集（以下简称coreset）的具体算法之前，首先要说明CRAIG的实现是通过一个贪心最大堆迭代更新遴选实现的。最大堆每次推出堆顶元素，将其加入S_i后，查看堆二、堆三……与S_i的相似度评分情况。如果相似度评分比S_{i-1}更高，则选取堆顶元素入子集，并更新堆中评分；否则，将堆顶元素压回堆，取出堆二元素重复评分操作。具体的评分函数F见下：

$$F(S) = L(\{s_0\}) - L(S \cup \{s_0\})$$
$$\text{where } L(S) = \sum_{i \in V} \min_{j \in S} d_{ij} \leq \epsilon \quad (2)$$

在CRAIG的压缩算法中，每次将根据$F(S)$得到的新元素加入当前子集S_i，此时剩余的样本集$V \setminus S_i$中的每个样本都对应且仅对应一个S_i中的元素。对应关系表明了该两个元素在特征分布上较为相似，S_i中的元素可以代表其对应的$v \setminus S_i$中的元素们。在迭代结束之后，我们将剩余$V \setminus S$的权重对应赋给S中的代表变量。由此，子数据集S在保留数据分布特点的情况下实现了数据集的精简压缩。注意这种对应关系在迭代过程中将动态变化。

原论文 [7]伪代码见下。

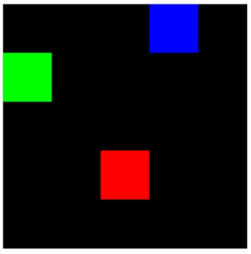
Algorithm 4 CRAIG: CoReset for Accelerating IG tasks

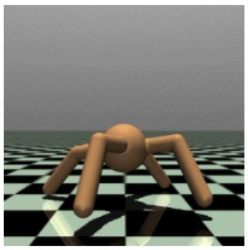
Require: Set of component functions f_i for $i \in V = [n]$.
$S_0 \leftarrow \emptyset, s_0 = 0, i = 0$
while $F(S) < L(\{s_0\}) - \epsilon$ **do**
$j \in \arg\max_{e \in V \backslash S_{i-1}} F(e|S_{i-1})$
$S_i = S_{i-1} \cup \{j\}$
$i = i + 1$
end while
for $j = 1$ to $|S|$ **do**
$\gamma_j = \sum_{i \in V} I[j = \arg\min_{s \in S} \max_{\omega \in \mathcal{W}} ||\nabla f_i(\omega) - \nabla f_s(\omega)||]$
end for
return $S, \{\gamma\}_{j \in S}$

4. Experiment

为了验证BMG为传统SAC算法提供自动调整超参数α的能力之后能否提升SAC算法的效果，并检验对于超参数α的学习效果，我们主要在以下两个环境进行实验，一个是简单的Grid World环境，另一个则是mujoco环境中的 ant环境，下面我们将先简单介绍一下这两个环境，再对这两个环境下的实验结果进行展示和分析。值得注意的是我们称α是自适应的当α是定义出来的某个神经网络的输出，并且这个神经网络是在整体优化过程中不断更新的。

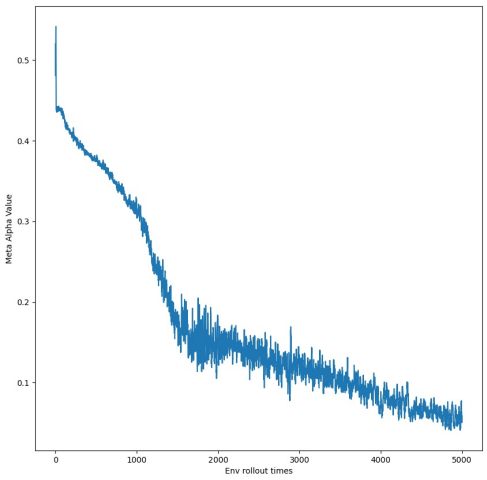
首先对于Grid World环境而言，我们这次使用的Grid World较为简单，仅仅是Two Color Grid World，在5×5的方格图中，仅有三种不同颜色的方格，分别是绿色，红色和黑色，在任意一种颜色的方格上智能体都可以沿着上下左右四个方向行走一个方格，如果智能体(显示为蓝色方格)在行走的过程中到达了绿色方块所在区域，则会获得+1的奖励值，反之如果行走的过程中到达了红色方块所在区域，就会获得-1的奖励值，而行走在黑色方块上会获得-0.04的奖励值，在整个方格图中，红色和绿色的方块都只有一个，黑色的方块 却占满剩下的所有方格，在随机初始化的方格图和智能体初始位置下，智能体需要在规定的行进步数之内获得最高的奖励值。

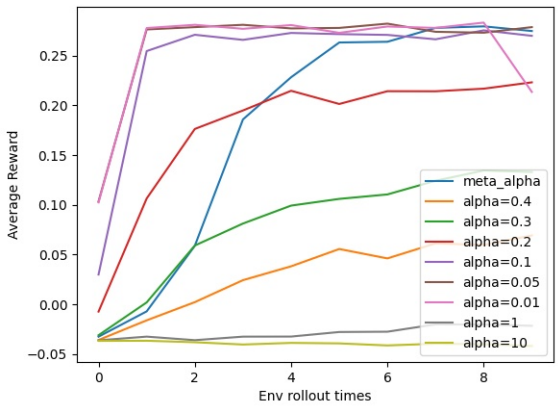




其次对于ant环境而言，ant环境的主要目标是控制一个拥有四条腿的机器人进行行走，作为mujoco环境库中的经典环境之一，ant首先区别于上述Two Color Grid World的是它具有连续的状态空间和动作空间，这意味着我们对于策略输出的建模不能仅仅是不同 动作之间的一个离散概率分布，而必须是整个连续的动作空间下的概率分布，我们此次实验选择的建模策略的方式是使用GMM作为策略来输出动作概率分布，也正是因为连续的动作空间使得BMG算法在实现的过程中没办法直接计算目标策略函数和当前策略函数输出的概率分布之间的 KL-div，这在一定程度上影响了BMG算法的效果，这一部分的具体实验结果会在后面详细给出。

介绍完了这两个环境之后，我们首先看一下在简单的环境，也即Two Color Grid World上面的实验结果。实验结果图如下，meta alpha展示了随着智能体学习的过程中超参数α的变化过程，average reward展示了不同种α下智能体在学习过程中获得的平均奖励值，这里的平均奖励值指的是在环境中进行一定步数的采样之后所获得的奖励的平均值。





这个环境下算法的表现如何呢？首先我们可以分析一下理想状态下的智能体在这个环境中会做什么，首先由于智能体在刚刚初始化完成的时候并不知道绿色的方格和红色的方格的分布，结合上文我们对α这个参数对于SAC算法的影响的分析可知，在整个学习过程的初始阶段，智能体应该先选用较高的α来保证自己对于环境的探索程度，从而尽可能快的获取高奖励值区域(绿色方块)和低奖励值区域(红色方块)的位置信息，而在进行了一段时间的探索之后，智能体将熟悉环境中绿色方格和红色方格的位置信息，并已经可以根据当前观测到的环境 (这里是整个Grid World)做出最正确的动作，也即以最短的步数到达绿色方块，并在行进过程中避免遇见红色方块。这个时候智能体不需要给自己的策略带来太多的不确定性，也即智能体应该降低自己的探索程度，也即降低α，这在策略的表达式$\pi^{\star}_{MaxEnt}(a_t|s_t) \propto exp\left(\frac{Q^{\star}_{soft}(s_t,a_t)-V^{\star}_{soft}(s_t)}{\alpha}\right)$体现为值函数大的动作将会主导整个策略的概率分布，也即soft-max操作更加偏向于max而不是偏向于soft。

事实上我们的算法的最终实验结果和理想状态下的智能体相差无几，可以发现α先以一个较快的速度下降

到合适的区间，然后逐渐减缓变小的速度并收敛到一个相对稳定的区间之内，这恰好对应了我们之前对智能体的分析：前期凭借较大的α进行对环境的探索，后期凭借较小的α 以确定的最优路径在每次初始化之后直接绕过红色方块的情况下到达绿色方块。

那么我们的算法学习得到的α是否有意义呢？又或者说我们学习得到的α能否作为一个合适的固定的α值来引导整个学习过程呢？为了解决这个疑惑，我们对于不同的固定的α值进行了消融实验，仅仅改变不同的α的取值，在相同的环境，相同的参数下 进行相同的实验，实验结果如上图所示，可以发现，首先SAC对于α的要求是十分严格的，α=0.1和α=0.2这两个不同的超参数设置下跑出来的结果就会相差很多，更不要说相差一个数量级的α取值了，因此在SAC算法中选用自适应的α值是很有意义的， 其次我们可以发现最稳定的α取值范围应该在0.05这个区间附近，因为在这个区间附近的α值不仅能让智能体快速获得较高的平均奖励值，也能不随着学习过程的进行过拟合，并产生表现下降的现象。这和我们的算法所学习到的α区间是一致的，因此我们有把握说这个算法学到了 正确的α值。 至于ant环境下的超参数学习，我们在此次实验中并没有调试出很好的结果，这里就不给出ant环境下的超参数学习结果了，可以通过提供的代码相对快速的跑出实验结果。我们认为BMG在复杂环境下效果较差的原因有以下几点:

1.连续环境下难以选择合适的距离来衡量BMG中目标策略和当前策略的差距。我们知道BMG算法的核心就在于通过最小化自举产生的目标策略函数和当前策略函数之间的某种距离μ，也即$w' = \text{argmin}_w\, \mu(\tilde{X}, X^{(K)}(w))$来根据次级优化过程中保存的有关超参数梯度的信息来优化超参数， 在离散环境下由于策略的输出直接就是离散的概率分布，可以用张量来直接表示这个概率分布并计算对应的距离(这里我们选用的是KL-div)；然而在ant这种连续的环境下，通过GMM表达的策略输出没办法直接计算两个概率分布之间的这种距离，只能取其次，转而优化GMM参数之间的某种距离(在此次实验中我们选择了优化GMM均值和方差的欧式距离)， 最小化这种替代的距离往往不能真正起到相应的效果，甚至可能提供错误的梯度，从而让超参数向着错误的方向更新。

2.复杂环境下直接最小化自举产生的目标策略函数和当前策略函数之间的某种距离可能并不能代表超参数更新的正确方向。我们知道BMG的直观理解就是根据未来的优化过程的信息，调整当前超参数的设置从而让K步的优化过程能够达到和$K+L$步优化过程相似的结果。由于策略网络的参数是十分高维的，我们无法用一种简单的方式判断出最小化某种距离 并只进行数步优化能否缩短二者的距离，因为高维空间下利用局部线性化进行梯度更新可能并不能朝着正确的方向更新，只有当更新的步数足够多之后才能有相对应的保证，然而我们又不可能在超参数更新的过程中耗费太多步数，否则将会严重拖慢原本网络参数的学习过程，这样就和我们最开始的目标背道而驰了。而这种局部线性化带来的不确定性也是 深度学习端到端的学习方式自带的缺陷之一。

4.1. Coreset for Sac-ant

sac-ant是sac任务中较为经典的子任务之一，我们用该任务实验CRAIG算法在RL上的表现。

首先，考虑直接将coreset移植到buffer上的方法。将buffer的空间大小设置在较小可达的值，若在存储样本环节检测到buffer剩余空间不足时，进行CRAIG数据压缩得到coreset，再将buffer清空并替换为coreset中的样本。具体而言，在CRAIG的评估函数中，根据样本的四元组(s_t, a_t, r_t, s_{t+1})的值进行相似度评估。

由于在实际模型中，s_t即当前状态量的维度较多，而sac-ant中a_t, r_t均为单量，故在原来的维度上动作-激励对的影响将被状态量削弱。考虑到我们希望coreset能够更多地参考动作-激励对地影响，在数据压缩时调整rewards actions的比重，使其对于评估函数更加重要。由此，可以筛选出更多基于动作-激励对的有价值代表样本。

除此之外，由于sac-ant的最终目的是尽量获取高的激励，在coreset选取时也应当更加关注那些激励较高的“好样本”，减少探索尝试产生的激励较低的“坏样本”的比重。这在后续基于buffer的采样中应当是有益的。因此，调整相似度函数$L(S) = \sum_{i \in V} \min_{j \in S} d_{ij} + r_j * \alpha$，其中$r_j$为样本的激励，$\alpha$为激励幅度因子。

最后，考虑到RL任务的探索尝试机制，CRAIG等在IG等深度学习领域大展身手的数据压缩算法很可能在RL领域无法取得较好的效果。因此，我们也考虑了不丢弃完整buffer信息、而是通过coreset提高有价值样本分布比重的方法。这样的考虑是出于在智能体选择动作、从buffer中采样时，能够更多地参考有价值的coreset数据信息，但同时也不丢失原始完整数据经验。

该方法类似插值，故称之为压缩插值法。这种做法虽然牺牲了实验动机中轻量化数据集的目的，但是从动机二的数据分布优化的角度上仍然是成立的。

实验结果画图如下(Figure 2)。实验0为原始sac-ant表现。可以发现单纯的coreset方法并不理想，激励波动较为严重，且最终所得(Return)持平，模型并没有从压缩数据集中学习到行动方案。

对于其中的周期性波动解释分析为，每次buffer压缩coreset后，采样数据空间分布又更加集中了，这不利于接下来的行为探索，故每次压缩coreset后rewards均会下跌。但是在每个压缩后的增长过程中，激励的提升速度很快且略高于原始实验（以初始增长作为对比），猜测这可能是由于压缩后的coreset虽然在分布上较为单一，无法提供完整全面的环境状态信息，但是较少的数据仍然保留了其高价值性，因此在每次重新学习中能够更加迅速地探索、学习到更高的水平。

基于压缩插值法的实验因为某些服务器环境的原因被迫中断，但是在展示的三个不完整过程中，可以注意到它们不仅没有像简单CRAIG那样无法保持高激励

学习信息，而且还表现出了基于原始任务更加出色的学习能力：即激励和最终所得的增长高于原始任务。分析认为这是coreset提炼出的高价值数据分布优化所带来的改进。在模型采样时，提高高价值高代表性样本的分布从而提高其被采样到的概率，类似于从最优数据和总体数据之间作插值，一方面不丢失完整数据信息的前提下，一方面提高了模型利用“好样本”的效率。

References

[1] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. 2017. 2

[2] S. Flennerhag, Y. Schroecker, T. Zahavy, H Van Hasselt, D. Silver, and S. Singh. Bootstrapped meta-learning. 2021. 3

[3] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. 2021. 2

[4] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies, 2017. 2

[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. 1, 2, 4

[6] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Poczos. Multi-fidelity bayesian optimisation with continuous approximations, 2017. 2

[7] B. Mirzasoleiman, J. Bilmes, and J. Leskovec. Coresets for data-efficient training of machine learning models. 2019. 3, 4

[8] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012. 2

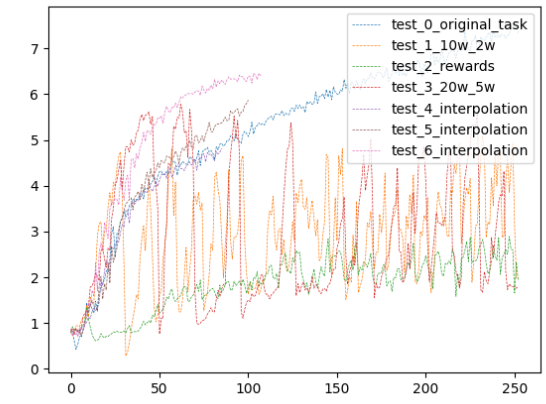
[9] Z. Xu, H Van Hasselt, and D. Silver. Meta-gradient reinforcement learning. 2018. 2

5. Conclusion

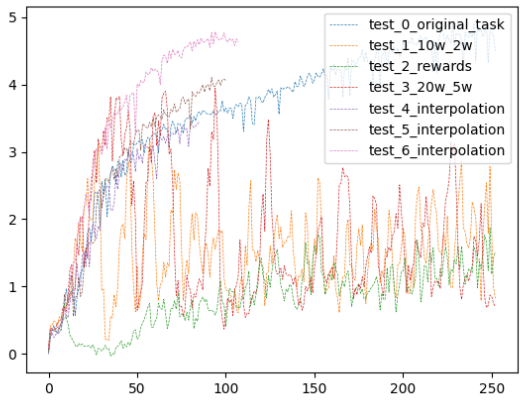
我们这次项目主要想探索超参数学习和数据压缩这两个在深度学习中得到广泛应用的技术是否能在强化学习场景下也起到令人满意的效果，从实验结果的角度来看，在简单的环境下这些深度学习中的技巧是可以 起到辅助强化学习算法的作用的，并且也具有可解释性。但是随着任务变得复杂，强化学习算法本身带来的不确定性以及超参数学习和数据压缩这两个十分依赖任务本身的技巧可能导致性能下降，甚至没有什么提升， 这种现象的产生并不是偶然，超参数学习本就是深度学习中非常前沿的一类任务，本质上是想要解决“如何让神经网络学会学习”这件更加抽象的事情；而数据压缩这个操作对于端到端的深度学习框架而言本来就是 风险很大的，在没办法解释深度学习的原理之前，我们其实并不知道什么数据是适合这个任务的，什么数据是不适合这个任务的，在算法的实现中也可以发现，更多是凭借一些经验性的原则去挑选数据。

在上文有关数据压缩的观察实验中，我们发现照搬DL的数据压缩方法给RL是很难生效的，这是因为RL任务对探索试错的要求。coreset的压缩会将大部分的试错样本都归类为“坏样本”并且丢弃，对于DL这是一种有效的小样本学习方法，但是对于RL任务而言则会丢失掉很多信息。然而，压缩插值法的优良表现也证明了coreset压缩后数据本身的价值确实是很有用的，这给了我们更多的改进思路。

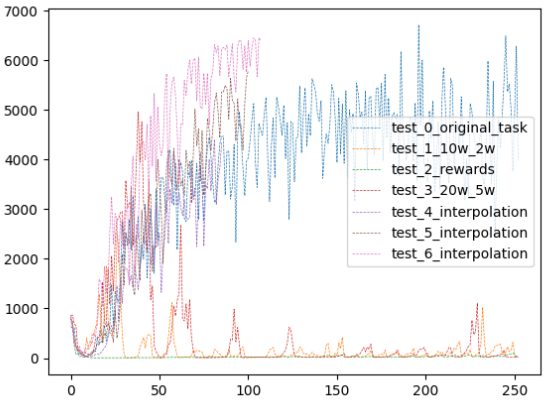
综上所述，这次实验还是给我们带来了很多惊喜，比如Two Color Grid World环境下合理的超参数学习结果和不错的最终表现，以及数据压缩的压缩插值法给算法框架带来的活力。希望能在后续的学习和研究中进一步深入的探索， 发现更多更有价值的内容。



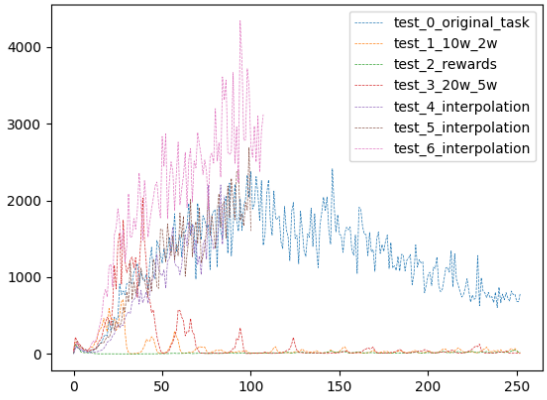
(a) Test Rewards Mean



(b) Exploration Rewards Mean



(c) Average Return



(d) Exploration Returns Mean

Figure 2. CRAIG实验结果