Homework 3

21215068

蔡云龙

Task MPE（Multi-agent Particle Environment)

(coding)在`MPE`的`simple spread`环境中实现MADDPG、VDN、QMIX三个算法，对于三个算法有如下要求：

1. 所有算法均采用参数共享。
2. 不能修改原始环境的任何文件。不能修改环境返回的奖励值。QMIX需要全局状态，可通过对obs的理解自行定义。
3. 奖励值最终至少收敛到-5.5。
4. MADDPG、VDN与环境交互的episode数量不超过50000，QMIX与环境交互的次数不做限制。
5. 详细分析不同全局状态对QMIX算法性能的影响，类似于多智能体课件中MAPPO对全局状态的分析一样。

MADDPG

原论文：[1706.02275] Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments (arxiv.org)

简介

传统单智能体的算法不能用于多智能体强化学习的主要原因就是每个agent的策略在训练过程中都是不断变化的，因此对每个agent来说，每次训练环境都是不稳定的（因为其他智能体的随机性），而在这种不稳定的环境中学习到的策略自然也是无意义的。

MADDPG即为Multi-Agent Deep Deterministic Policy Gradient。Multi-Agent表示多智能体强化学习；Deep表示使用深度网络，类似DQN，使用目标网络和经验回放等机制；Deterministic表示直接输出确定性的动作；Policy Gradient表示基于策略Policy来做梯度下降从而优化模型。

MADDPG其实是在DDPG的基础上做的修改,而DDPG可以看作在DPG的基础之上修改而来,DPG是由DQN和Policy Gradient两者结合后得到的;也可以把DDPG理解为让DQN可以扩展到连续控制动作空间的算法。

基础思想是：集中式训练，分布式执行(Centralized Train and Decentralized execution, CTDE)，其实也算简单粗暴的方法，即在训练时有个全局的critic指导actors的训练，每个actor在做决策时只需根据局部观测采取动作。

所以MADDPG考虑使用一个全局的critic。当知道所有agent的动作时，无论策略如何变化，环境都是稳定的：

A primary motivation behind MADDPG is that, if we know the actions taken by all agents, the environment is stationary even as the policies change, since $P(s'|s, a_1, ..., a_N, \boldsymbol{\pi}_1, ..., \boldsymbol{\pi}_N) = P(s'|s, a_1, ..., a_N) = P(s'|s, a_1, ..., a_N, \boldsymbol{\pi}'_1, ..., \boldsymbol{\pi}'_N)$ for any $\boldsymbol{\pi}_i \neq \boldsymbol{\pi}'_i$. This is not the case if we do not explicitly condition on the actions of other agents, as done for most traditional RL methods.

MADDPG算法的三点特征：1. 通过学习得到的最优策略，在应用时只利用局部信息就能给出最优动作。 2. 不需要知道环境的动力学模型以及特殊的通信需求。 3. 该算法不仅能用于合作环境，也能用于竞争环境。

三个技巧：1. 集中式训练，分布式执行（CTDE）：训练时采用集中式学习训练critic与actor，使用时actor只用知道局部信息就能运行。critic需要其他智能体的策略信息。2. 改进了经验回放记录的数据。3. 利用策略集合优化（policy ensemble）：对每个智能体学习多个策略，改进时利用所有策略的整体效果进行优化。以提高算法的稳定性以及鲁棒性。

实现

MADDPG的分布式actor和中心式critic结构如下：

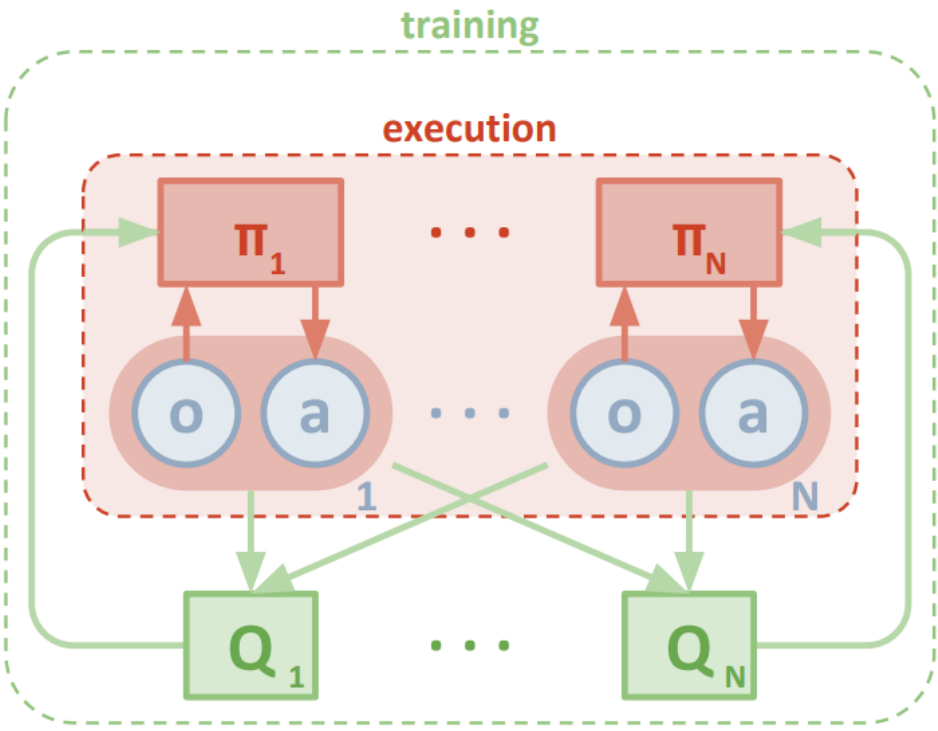


Figure 1: Overview of our multi-agent decentralized actor, centralized critic approach.

这里π为智能体策略，设定其参数为θ，$J(\theta)$为第i个智能体的累计期望奖励$E[R_i]$，由此第i个agent的梯度可以表示为

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s\sim p^{\boldsymbol{\mu}}, a_i\sim \boldsymbol{\pi}_i}[\nabla_{\theta_i} \log \boldsymbol{\pi}_i(a_i|o_i) Q_i^{\boldsymbol{\pi}}(\mathbf{x}, a_1, ..., a_N)]. \tag{4}$$

o_i表示第i个智能体的观测，$x = [o_0, \ldots, o_i]$ 即状态。 Q_i^{π}为critic_network。由于每个智能体独立的学习自己的Q_i^{π}，所以，MADDPG可应用于合作或竞争任务。

对于确定性策略，梯度可以表示为：

$$\nabla_{\theta_i} J(\boldsymbol{\mu}_i) = \mathbb{E}_{\mathbf{x}, a\sim\mathcal{D}}[\nabla_{\theta_i} \boldsymbol{\mu}_i(a_i|o_i) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, ..., a_N)|_{a_i=\boldsymbol{\mu}_i(o_i)}], \tag{5}$$

可以利用该梯度更新actor网络，对于critic网络，需要计算其与目标网络的均方差作为loss来更新参数。通过以下方式更新中心式动作值函数Q_i^{μ}：

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\mathbf{x}, a, r, \mathbf{x}'}[(Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, \ldots, a_N) - y)^2], \quad y = r_i + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}', a_1', \ldots, a_N')\big|_{a_j'=\boldsymbol{\mu}_j'(o_j)}, \tag{6}$$

对于上述的技巧3，即策略集合优化(policies ensemble)，MADDPG提出的策略集合的思想是，第i个智能体的策略 μ_i由一个具有`K`个子策略的集合构成，在每一个训练episode中只是用一个子策略$\mu_{\theta_i^{(k)}}$ （简写为 $\mu_i^{(k)}$）。对每一个智能体，最大化其策略集合的整体奖励$J_e(\mu_i) = E_{k\sim unif(1,K),s\sim\rho^{\mu},a\sim\mu_i^{(k)}}[\sum_{t=0}^{\infty}\gamma^t r_{i,t}]$。并且为每个子策略k构建一个记忆存储：$D_i^{(k)}$。优化策略集合的整体效果，因此针对每一个子策略的更新梯度为：

$$\nabla_{\theta_i^{(k)}} J_e(\boldsymbol{\mu}_i) = \frac{1}{K}\mathbb{E}_{\mathbf{x},a\sim\mathcal{D}_i^{(k)}}\left[\nabla_{\theta_i^{(k)}}\boldsymbol{\mu}_i^{(k)}(a_i|o_i)\nabla_{a_i}Q^{\boldsymbol{\mu}_i}(\mathbf{x},a_1,\ldots,a_N)\Big|_{a_i=\boldsymbol{\mu}_i^{(k)}(o_i)}\right]. \quad (9)$$

算法流程:

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)|_{a'_k=\boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\boldsymbol{\mu}_i(o_i^j)\nabla_{a_i}Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\Big|_{a_i=\boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$$

  end for
end for

实验

MADDPG实验部分代码见`hw3_code/MPE_maddpg_and_vdn`目录。相关参数写在了`utils/arguments.py`文件中，设定最大运行episode为50000；算法实现见`algorithms/maddpg.py`文件。运行结果会保存在`results/maddpg`文件夹中，包括模型、图片和logs。

运行:

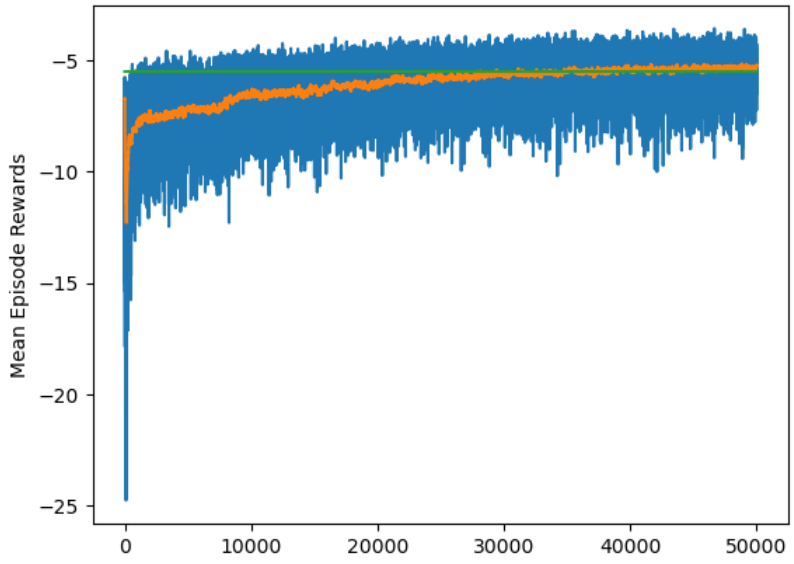
```
python main.py --algo maddpg
```

实验结果:

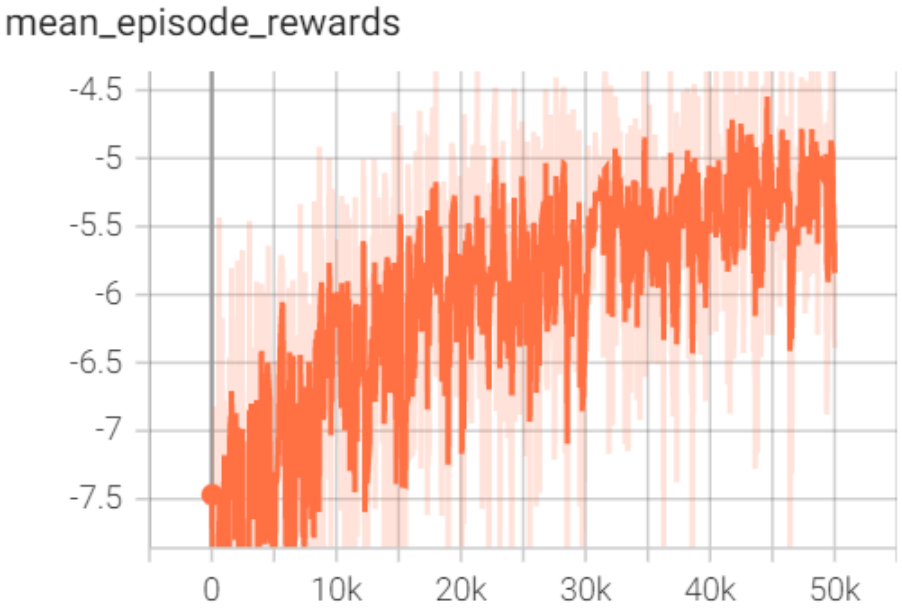
运行过程log结果：

```
Episode 49973 of 50000 : Total reward -139.49 , Mean reward -5.58 , last_100_mean -5.22
Episode 49974 of 50000 : Total reward -137.70 , Mean reward -5.51 , last_100_mean -5.22
Episode 49975 of 50000 : Total reward -129.15 , Mean reward -5.17 , last_100_mean -5.22
Episode 49976 of 50000 : Total reward -132.84 , Mean reward -5.31 , last_100_mean -5.23
Episode 49977 of 50000 : Total reward -120.92 , Mean reward -4.84 , last_100_mean -5.23
Episode 49978 of 50000 : Total reward -121.78 , Mean reward -4.87 , last_100_mean -5.23
Episode 49979 of 50000 : Total reward -114.06 , Mean reward -4.56 , last_100_mean -5.22
Episode 49980 of 50000 : Total reward -143.71 , Mean reward -5.75 , last_100_mean -5.22
Episode 49981 of 50000 : Total reward -146.68 , Mean reward -5.87 , last_100_mean -5.22
Episode 49982 of 50000 : Total reward -118.99 , Mean reward -4.76 , last_100_mean -5.21
Episode 49983 of 50000 : Total reward -152.37 , Mean reward -6.09 , last_100_mean -5.22
Episode 49984 of 50000 : Total reward -123.99 , Mean reward -4.96 , last_100_mean -5.21
Episode 49985 of 50000 : Total reward -130.53 , Mean reward -5.22 , last_100_mean -5.22
Episode 49986 of 50000 : Total reward -114.27 , Mean reward -4.57 , last_100_mean -5.21
Episode 49987 of 50000 : Total reward -151.45 , Mean reward -6.06 , last_100_mean -5.22
Episode 49988 of 50000 : Total reward -129.00 , Mean reward -5.16 , last_100_mean -5.22
Episode 49989 of 50000 : Total reward -180.95 , Mean reward -7.24 , last_100_mean -5.23
Episode 49990 of 50000 : Total reward -175.84 , Mean reward -7.03 , last_100_mean -5.24
Episode 49991 of 50000 : Total reward -123.47 , Mean reward -4.94 , last_100_mean -5.24
Episode 49992 of 50000 : Total reward -139.61 , Mean reward -5.58 , last_100_mean -5.25
Episode 49993 of 50000 : Total reward -130.19 , Mean reward -5.21 , last_100_mean -5.24
Episode 49994 of 50000 : Total reward -149.54 , Mean reward -5.98 , last_100_mean -5.25
Episode 49995 of 50000 : Total reward -171.83 , Mean reward -6.87 , last_100_mean -5.26
Episode 49996 of 50000 : Total reward -133.60 , Mean reward -5.34 , last_100_mean -5.26
Episode 49997 of 50000 : Total reward -146.44 , Mean reward -5.86 , last_100_mean -5.26
Episode 49998 of 50000 : Total reward -123.82 , Mean reward -4.95 , last_100_mean -5.26
Episode 49999 of 50000 : Total reward -129.04 , Mean reward -5.16 , last_100_mean -5.24
Episode 50000 of 50000 : Total reward -159.78 , Mean reward -6.39 , last_100_mean -5.26
```

奖励与最后100轮平均奖励：



logs：



VDN

原论文：[1706.05296] Value-Decomposition Networks For Cooperative Multi-Agent Learning (arxiv.org)

简介

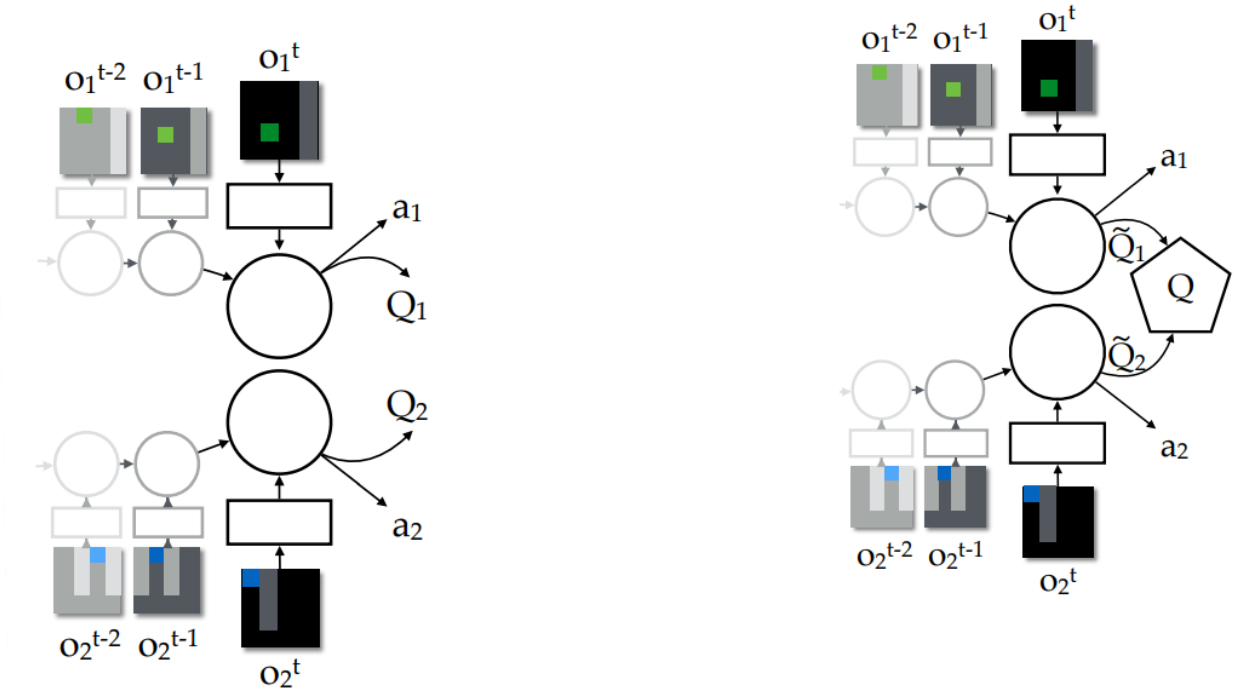
由于每个智能体都是局部观测，对其中一个智能体来说，其获得的团队奖励很有可能是其队友的行为导致的。也就是说该奖励值对该智能体来说，是“虚假奖励 (spurious reward)”。因此，每个智能体独立使用强化学习算法学习 (即independent RL) 往往效果很差。与这种虚假奖励相伴随的现象即"lazy agent"，当团队中的部分智能体学习到了比较好的策略并且能够完成任务时，其它智能体不需要做什么也能获得不错的团队奖励，这些智能体就被称作"lazy agent"。

不管是“虚假奖励”还是“lazy agent”，本质上还是credit assignment问题。如果每个智能体都会根据自己对团队的贡献，优化各自的目标函数，就能够解决上述问题。基于这样的动机，作者提出了“值函数分解”(Value Decomposition)的研究思路，将团队整体的值函数分解成N个子值函数，分别作为各智能体执行动作的依据。

实现

VDN为每个agent单独设置一个Q_i，然后取每个agent最大的Q值（同时采取产生最大Q值这个动作，但有 ϵ 的机率选择随机动作进行探索），然后将$maxQ_i$**直接加和**作为整体的 Q_{total} ，最后Q_{total}与团队奖励的误差反向学习Q_i，从而可以体现每个agent的影响大小，也就是集中训练，分散执行。

与传统的独立智能体架构对比如下：



左边为Independent agent architecture，每个agent有自己单独的Q网络，它们分别计算每个agent的Q值并单独进行自己的更新；而文献中提到的Value-decomposition individual architecture如右侧图所示，将每个智能体的子Q函数求和得到系统的Q函数，即如下方法：

假设 $Q((h^1,h^2,\ldots,h^d),(a^1,a^2,\ldots,a^d))$ 是多智能体团队的整体Q函数，d 是智能体个数， h^i 是智能体i的历史序列信息， a^i 是其动作。该Q函数的输入集中了所有智能体的观测和动作，可通过团队奖励 r 来迭代拟合。为了得到各个智能体的值函数，提出如下假设：

$$Q((h^1,h^2,...,h^d),(a^1,a^2,...,a^d)) \approx \sum_{i=1}^{d} \tilde{Q}_i(h^i,a^i)$$

即团队的Q函数可以通过对d个智能体的子Q函数求和得到，且每个子Q函数的输入为该对应智能体的局部观测历史信息和动作，互不影响。其中$\tilde{Q}_i(h^i,a^i)$是每个agent根据自己local observation得到的Q值，在VDN中，由于每个agent按照贪心选择Q值最大的动作，因此也就相当于整体的 Q_{total} 也是取最大值，使训练与执行的单调性相同。对每个$\tilde{Q}_i$取 `argmax` 函数就能得到每个智能体根据自己的局部Q函数进行决策，得到a^i。

实际上上式成立需要满足：$r(\boldsymbol{s},\boldsymbol{a}) = \sum_{i=1}^{d} r(o^i,a^i)$。即总的reward是由所有智能体的reward相加产生。论文也举了两个智能体的例子进行说明，有：

$$Q^{\pi}(\mathbf{s},\mathbf{a}) = \mathbb{E}[\sum_{t=1}^{\infty}\gamma^{t-1}r(\mathbf{s}_t,\mathbf{a}_t)|\mathbf{s}_1=\mathbf{s},\mathbf{a}_1=\mathbf{a};\pi]$$

$$= \mathbb{E}[\sum_{t=1}^{\infty}\gamma^{t-1}r_1(o_t^1,a_t^1)|\mathbf{s}_1=\mathbf{s},\mathbf{a}_1=\mathbf{a};\pi] + \mathbb{E}[\sum_{t=1}^{\infty}\gamma^{t-1}r_2(o_t^2,a_t^2)|\mathbf{s}_1=\mathbf{s},\mathbf{a}_1=\mathbf{a};\pi]$$

$$=: \bar{Q}_1^{\pi}(\mathbf{s},\mathbf{a}) + \bar{Q}_2^{\pi}(\mathbf{s},\mathbf{a})$$

where $\bar{Q}_i^{\pi}(\mathbf{s},\mathbf{a}) := \mathbb{E}[\sum_{t=1}^{\infty}\gamma^{t-1}r_1(o_t^i,a_t^i)|\mathbf{s}_1=\mathbf{s},\mathbf{a}_1=\mathbf{a};\pi], i=1,2.$

进而验证了下式：

$$Q^{\pi}(\mathbf{s},\mathbf{a}) =: \bar{Q}_1^{\pi}(\mathbf{s},\mathbf{a}) + \bar{Q}_2^{\pi}(\mathbf{s},\mathbf{a}) \approx \tilde{Q}_1^{\pi}(h^1,a^1) + \tilde{Q}_2^{\pi}(h^2,a^2)$$

也就说明了需要满足上面的公式：$r(\boldsymbol{s},\boldsymbol{a}) = \sum_{i=1}^{d} r(o^i,a^i)$。

论文给出的网络结构如下：

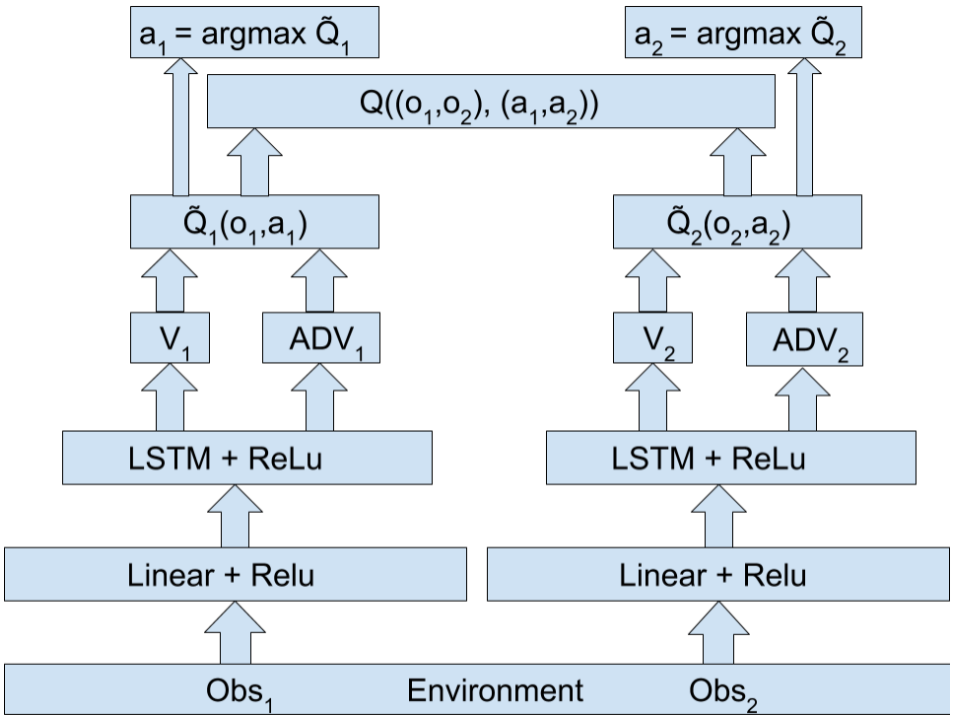


Figure 15: Value-Decomposition Individual Architecture

其他变种比如在底层和高层加入通信的结构分别见论文的 **Figure16,17,21,20** 等。

实验

VDN实验部分代码见`hw3_code/MPE_maddpg_and_vdn`目录。相关参数写在了`utils/arguments.py`文件中，设定最大运行episode为50000；算法实现见`algorithms/vdn.py`文件。运行结果会保存在`results/vdn`文件夹中，包括模型、图片和logs。

运行:

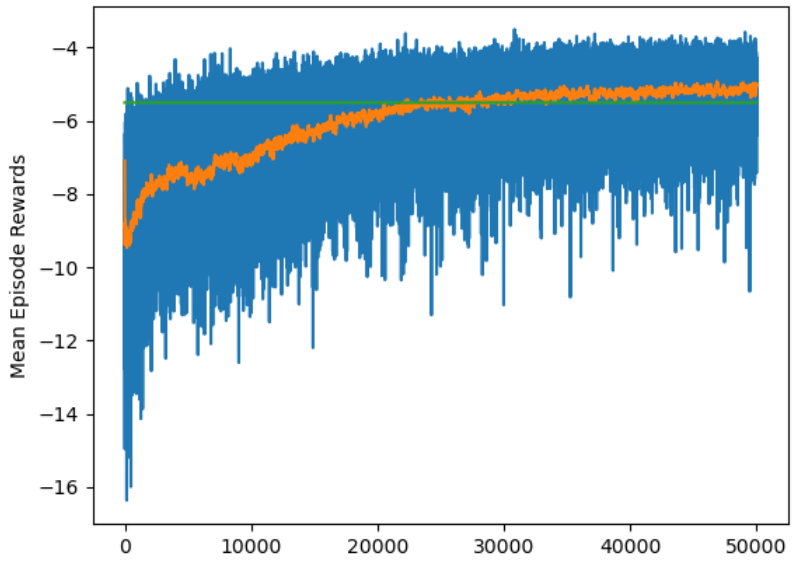
```
python main.py --algo vdn
```

实验结果:

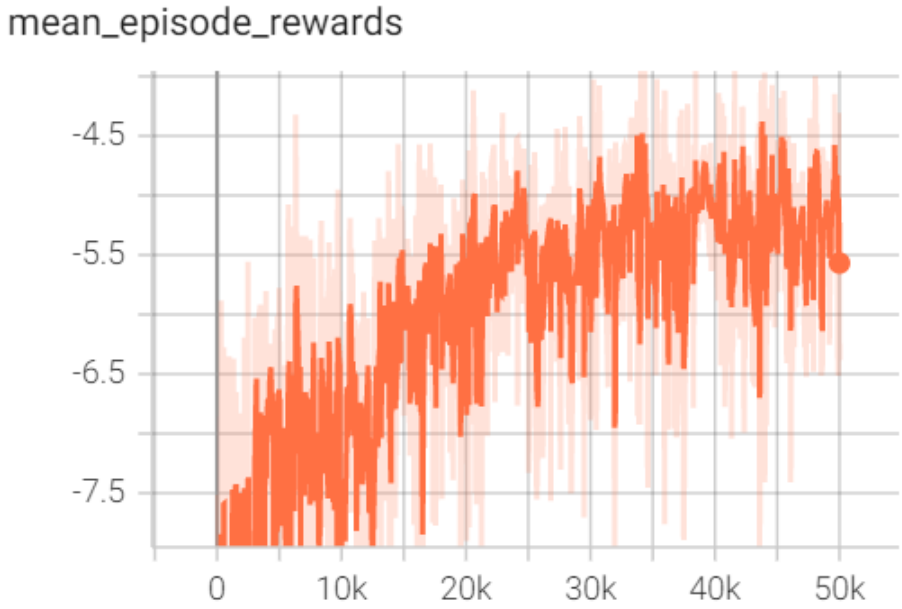
运行过程log结果:

```
Episode 49973 of 50000 : Total reward -123.78 , Mean reward -4.95 , last_100_mean -5.00
Episode 49974 of 50000 : Total reward -108.00 , Mean reward -4.32 , last_100_mean -5.00
Episode 49975 of 50000 : Total reward -109.69 , Mean reward -4.39 , last_100_mean -4.99
Episode 49976 of 50000 : Total reward -134.24 , Mean reward -5.37 , last_100_mean -5.00
Episode 49977 of 50000 : Total reward -120.71 , Mean reward -4.83 , last_100_mean -5.01
Episode 49978 of 50000 : Total reward -134.52 , Mean reward -5.38 , last_100_mean -5.00
Episode 49979 of 50000 : Total reward -126.29 , Mean reward -5.05 , last_100_mean -5.00
Episode 49980 of 50000 : Total reward -118.70 , Mean reward -4.75 , last_100_mean -5.00
Episode 49981 of 50000 : Total reward -106.66 , Mean reward -4.27 , last_100_mean -5.01
Episode 49982 of 50000 : Total reward -156.73 , Mean reward -6.27 , last_100_mean -5.02
Episode 49983 of 50000 : Total reward -185.72 , Mean reward -7.43 , last_100_mean -5.05
Episode 49984 of 50000 : Total reward -127.79 , Mean reward -5.11 , last_100_mean -5.06
Episode 49985 of 50000 : Total reward -133.79 , Mean reward -5.35 , last_100_mean -5.07
Episode 49986 of 50000 : Total reward -126.97 , Mean reward -5.08 , last_100_mean -5.07
Episode 49987 of 50000 : Total reward -119.92 , Mean reward -4.80 , last_100_mean -5.07
Episode 49988 of 50000 : Total reward -112.72 , Mean reward -4.51 , last_100_mean -5.06
Episode 49989 of 50000 : Total reward -127.94 , Mean reward -5.12 , last_100_mean -5.06
Episode 49990 of 50000 : Total reward -112.73 , Mean reward -4.51 , last_100_mean -5.05
Episode 49991 of 50000 : Total reward -113.70 , Mean reward -4.55 , last_100_mean -5.05
Episode 49992 of 50000 : Total reward -134.17 , Mean reward -5.37 , last_100_mean -5.05
Episode 49993 of 50000 : Total reward -123.59 , Mean reward -4.94 , last_100_mean -5.03
Episode 49994 of 50000 : Total reward -113.37 , Mean reward -4.53 , last_100_mean -5.01
Episode 49995 of 50000 : Total reward -118.24 , Mean reward -4.73 , last_100_mean -5.02
Episode 49996 of 50000 : Total reward -121.41 , Mean reward -4.86 , last_100_mean -5.01
Episode 49997 of 50000 : Total reward -133.04 , Mean reward -5.32 , last_100_mean -5.02
Episode 49998 of 50000 : Total reward -112.47 , Mean reward -4.50 , last_100_mean -5.01
Episode 49999 of 50000 : Total reward -116.60 , Mean reward -4.66 , last_100_mean -5.02
Episode 50000 of 50000 : Total reward -159.60 , Mean reward -6.38 , last_100_mean -5.02
```

奖励与最后100轮平均奖励:



logs:



QMIX

原论文：[1803.11485] QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning (arxiv.org)

简介

QMIX是VDN的一种扩展，由于VDN只是将每个智能体的局部动作值函数求和相加得到联合动作值函数，虽然满足联合值函数与局部值函数单调性相同的可以进行分布化策略的条件，但是其没有在学习时利用状态信息以及没有采用非线性方式对单智能体局部值函数进行整合，使得VDN算法还有很大的提升空间。

QMIX就是采用一个混合网络对单智能体局部值函数进行合并，并在训练学习过程中加入全局状态信息辅助（通过hypernetwork处理全局状态信息），来提高算法性能。为了能够沿用VDN的优势，利用集中式的学习，得到分布式的策略。也就是要对联合动作值函数取 $argmax$ 等价于对每个局部动作值函数取 $argmax$ ，其单调性要相同，即：

$$\underset{\mathbf{u}}{\operatorname{argmax}}\, Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \operatorname{argmax}_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \operatorname{argmax}_{u^n} Q_n(\tau^n, u^n) \end{pmatrix}. \quad (4)$$

因此分布式策略就是贪心的通过局部Q_i获取最优动作。QMIX将上式转化为一种Q_{total}与每个Q_a之间的单调性约束，为：

$$\frac{\partial Q_{tot}}{\partial Q_a}, \forall a \in A.$$

实现

QMIX的网络结构如图所示：

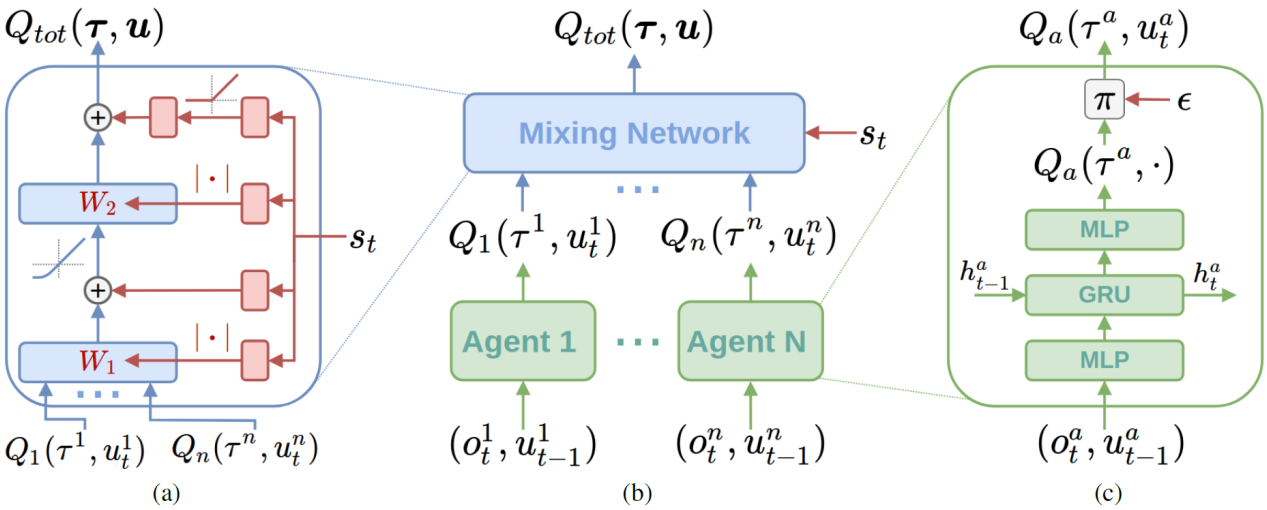


Figure 2. (a) Mixing network structure. In red are the hypernetworks that produce the weights and biases for mixing network layers shown in blue. (b) The overall QMIX architecture. (c) Agent network structure. Best viewed in colour.

主要有三部分网路：

- `agent networks`：对于每个智能体都要学一个独立的值函数$Q_a\left(\tau^a, u^a\right)$，单个智能体的网络结构采用DRQN的网络结构，在每个时间步，接收当前的独立观测O_t^a和上一个动作μ_{t-1}^a，如上图(c)部分所示。
- `mixing network`：其输入为每个DRQN网络的输出，即每个智能体的输出：$Q_n(\tau^n, u_t^a)$，输出联合动作值函数$Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u})$。为了满足上述的单调性约束，混合网络的所有权值都是非负数，对偏移量不做限制，这样就可以确保满足单调性约束。
- `hypernetworks`：hypernetworks网络去产生mixing network的权重，将系统的全局状态s作为输入，输出Mixing网络的每一层的超参数向量。为了保证权值的非负性，采用一个线性网络以及绝对值激活函数保证输出不为负数。对偏移量采用同样方式但没有非负性的约束，混合网络最后一层的偏移量通过两层网络以及ReLU激活函数得到非线性映射网络。

最终的代价函数为，其中b表示从经验记忆中采样的样本数量：

$$\mathcal{L}(\theta)=\sum_{i=1}^{b}\left[\left(y_i^{tot}-Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s; \theta)\right)^2\right], \qquad (6)$$

y^{tol}为：

$$y^{tot}=r+\gamma \max_{\mathbf{u}'} Q_{tot}(\boldsymbol{\tau}', \mathbf{u}', s'; \theta^-)$$

$Q_{tot}(\boldsymbol{\tau}^{'}, \boldsymbol{u}^{'}, s^{'}; \theta^{-})$为目标网络，$\theta^{-}$为类似DQN中目标网络的参数。

由于满足上文的单调性约束，对 Q_{tot}进行 $argmax$ 操作的计算量就不在是随智能体数量呈指数增长了，而是随智能体数量线性增长，极大的提高了算法效率。

实验

QMIX实验部分代码魔改自[marlbenchmark/off-policy(github.com)](), 见`hw3_code/qmix`目录。相关参数写在了`arguments.py`文件中，设定最大运行episode为100000；算法实现见`algorithms/qmix/qmix.py`文件。对mpe稍微进行了修改，使得运行时传入的参数能够传入到环境中。运行结果保存在`results/qmix`文件夹中，包括模型和logs。

QMIX实验使用的全局信息s是所有智能体的local observations的拼接(concatenate)，每个智能体的local observations包括：自身的速度、自身的位置、自身相对于地标的位置、其他智能体相对于自身的位置以及与其他智能体communication的数据。全局信息s作为hyper network的输入。

运行：直接运行qmix文件夹里的main.py即可，参数默认

```
python main.py
```

实验结果：

运行过程log结果:

```
NO.99971: simple_spread | qmix | timesteps: 2499275/2500000 | average_episode_rewards is -4.59
NO.99972: simple_spread | qmix | timesteps: 2499300/2500000 | average_episode_rewards is -4.47
NO.99973: simple_spread | qmix | timesteps: 2499325/2500000 | average_episode_rewards is -4.85
NO.99974: simple_spread | qmix | timesteps: 2499350/2500000 | average_episode_rewards is -4.73
NO.99975: simple_spread | qmix | timesteps: 2499375/2500000 | average_episode_rewards is -5.20
NO.99976: simple_spread | qmix | timesteps: 2499400/2500000 | average_episode_rewards is -4.62
NO.99977: simple_spread | qmix | timesteps: 2499425/2500000 | average_episode_rewards is -4.82
NO.99978: simple_spread | qmix | timesteps: 2499450/2500000 | average_episode_rewards is -5.41
NO.99979: simple_spread | qmix | timesteps: 2499475/2500000 | average_episode_rewards is -4.84
NO.99980: simple_spread | qmix | timesteps: 2499500/2500000 | average_episode_rewards is -5.73
NO.99981: simple_spread | qmix | timesteps: 2499525/2500000 | average_episode_rewards is -4.42
NO.99982: simple_spread | qmix | timesteps: 2499550/2500000 | average_episode_rewards is -4.43
NO.99983: simple_spread | qmix | timesteps: 2499575/2500000 | average_episode_rewards is -4.56
NO.99984: simple_spread | qmix | timesteps: 2499600/2500000 | average_episode_rewards is -5.14
NO.99985: simple_spread | qmix | timesteps: 2499625/2500000 | average_episode_rewards is -4.71
NO.99986: simple_spread | qmix | timesteps: 2499650/2500000 | average_episode_rewards is -6.45
NO.99987: simple_spread | qmix | timesteps: 2499675/2500000 | average_episode_rewards is -6.01
NO.99988: simple_spread | qmix | timesteps: 2499700/2500000 | average_episode_rewards is -5.59
NO.99989: simple_spread | qmix | timesteps: 2499725/2500000 | average_episode_rewards is -5.57
NO.99990: simple_spread | qmix | timesteps: 2499750/2500000 | average_episode_rewards is -5.03
NO.99991: simple_spread | qmix | timesteps: 2499775/2500000 | average_episode_rewards is -4.16
NO.99992: simple_spread | qmix | timesteps: 2499800/2500000 | average_episode_rewards is -4.88
NO.99993: simple_spread | qmix | timesteps: 2499825/2500000 | average_episode_rewards is -4.79
NO.99994: simple_spread | qmix | timesteps: 2499850/2500000 | average_episode_rewards is -4.46
NO.99995: simple_spread | qmix | timesteps: 2499875/2500000 | average_episode_rewards is -4.61
NO.99996: simple_spread | qmix | timesteps: 2499900/2500000 | average_episode_rewards is -4.45
NO.99997: simple_spread | qmix | timesteps: 2499925/2500000 | average_episode_rewards is -4.72
NO.99998: simple_spread | qmix | timesteps: 2499950/2500000 | average_episode_rewards is -5.45
NO.99999: simple_spread | qmix | timesteps: 2499975/2500000 | average_episode_rewards is -4.66
NO.100000: simple_spread | qmix | timesteps: 2500000/2500000 | average_episode_rewards is -5.03
eval_average_episode_rewards is -5.07
```

logs:

不同全局状态的影响：

1. 所有智能体的局部观测拼接成的全局状态：这种方式的全局状态s相对没有什么冗余信息，代表性较强，但如果是较复杂的智能体环境，或者智能体个数较多时，则全局状态的维度也会变得很大，导致hypernetwork的参数较多，会稍微增大训练时间。
2. 仅用一些全局的信息作为全局状态：比如所有地标的绝对位置，所有智能体的绝对位置和速度。这种确实是全局的信息，并且维度相对第一种来说小一点，能加快训练速度；但由于缺少智能体和地标之间的相对信息，代表性可能不足，最终训练的效果可能并不是很好。
3. 将上面两种合并，即将所有智能体的局部观测和全局的信息都拼接起来。这种方式几乎能够包含所有的信息，理论能够达到较好的结果，但是输入数据维度也更大，会减缓训练速度。
4. 对1进行改进，由于1中是拼接所有智能体的局部观测，然而这种局部观测是有冗余信息的，比如A对B的相对位置和B对A的相对位置就是一种冗余信息，可以将这部分冗余信息仅保留一个，因此可以降低输入数据的维度同时并不丢失太多全局信息。效果应该与1差不太多并会稍微减少训练时间。当然4 这种方式也可以与2拼接构成 3' 作为另一种全局状态。

参考：

> [1706.02275] Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments (arxiv.org)
>
> 多智能体强化学习入门（四）——MADDPG算法 - 知乎 (zhihu.com)
>
> 从代码到论文理解并复现MADDPG算法(基于飞桨的强化学习套件PARL)*Mr.郑先生*的博客-CSDN博客 maddpg算法
>
> MARL学习篇----MADDPG*昨日啊萌的博客-CSDN博客*maddpg
>
> [1706.05296] Value-Decomposition Networks For Cooperative Multi-Agent Learning (arxiv.org)
>
> VDN算法解析: Value-Decomposition Networks For Cooperative Multi-Agent Learning - 知乎 (zhihu.com)
>
> Qmix相关算法1：VDN笔记 - 知乎 (zhihu.com)
>
> 多智能体强化学习(一) IQL、VDN、QMIX、QTRAN算法详解*小小何先生的博客-CSDN博客*vdn算法
>
> 多智能体强化学习入门（五）——QMIX算法分析 - 知乎 (zhihu.com)
>
> openai/multiagent-particle-envs(github.com)
>
> openai/maddpg(github.com)
>
> philtabor/Multi-Agent-Deep-Deterministic-Policy-Gradients(github.com)
>
> ShAw7ock/MPE-Multiagent-RL-Algos(github.com)
>
> starry-sky6688/MARL-Algorithms(github.com)
>
> **marlbenchmark/off-policy**(github.com)