关于多智能体强化学习中适应对手(Opponent Adaptation)算法的相关调研

一、 问题描述

在多智能体强化学习中，由于环境中包含和自己一样可以采取不同决策的智能体，训练的结果往往会和对手的行为产生过拟合[1]，因而不适应策略可能千变万化的对手；因此，让智能体学会适应对手的行为，并及时做出调整就尤为重要。

在讨论有哪些“适应对手”的方法之前，首先需要明确对手的性质：对手是否可以随着训练过程同样变化策略以针对我们？我们只需要面对一个固定的对手（这个对手可能会不断学习来针对我们），还是要面对许多对手（这些对手可能有实现固定的策略）？面对许多对手，是以同样的策略去面对还是可以针对每个对手分别学习？

这篇调研对第三个问题的答案是“针对每个对手分别学习”，因为这样更符合“adaptation”的定义。大部分与 opponent adaptation 相关的综述[6][8][9][10]都从 MARL 环境的 non-stationarity 入手，提到对方 agent 的策略可能会变化；其中一篇综述[6]将解决方法按照“忽略”（当成静态环境处理）、“忘记”（即只顾学习当前环境）、“对固定策略”（一般假设对手满足一些性质，如理性或有限理性）、“对手建模”和“机器思维”（假设对手也会对自己建模，具有推理链）五个层次分类。可以看出，这种切入点更倾向于上面所说的第一种可能性（对手会不断学习）；而我自己的研究目前更需要对第二种可能性（未知而多样的对手，但每个对手策略固定）的解决方案。因此，这篇调研从不同的应用技术来入手分类，尽量兼顾到上述的两种可能性，并且倾向于对抗环境而非合作环境的适应。

总的来说，对 opponent adaptation 这一问题，大致有以下五种思路：

1. 将相同环境下不同对手看成一族类似任务的 meta-learning；
2. （在零和博弈中）将对手视为是“攻击”目标的 Adversarial training；
3. 以纳什均衡、best response 为基础的 game-theoretic；
4. 通过（非博弈论的）建模得到对对手行为的 belief，在此基础上进行针对的 Opponent Modeling；
5. 其他工程上的技巧。

其中，前两个方向只有少量的工作，而建模的方法是最多的。下面将介绍这些思路，并重点论述 meta-learning 方向的 Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environment[3]；对对手建模的 Bayesian Opponent Exploitation in Imperfect-Information Games 和 Variational Task Embeddings for Fast Adaptation in Deep Reinforcement Learning[47] 三篇文章。

用 Meta-learning 解决 opponent adaptation 的想法是：对于要做 adaptation 的 agent 来说，环境是基本不变的，变化的只有对手的 policy，因此可以将任务看成单智能体面对“一族”有少部分差异的环境，其相似性在于环境，差别在于对手的 policy 不同。目前在 agent adaptation 上有应用的 meta-learning 算法主要是 MAML[3][11]，其中[3]是最成功的文章；具体论述可见“重点文章论述”部分。

一般来说，adversarial RL 是指研究通过对输入/reward 函数/环境转移的微小扰动来大幅度降低目标表现，以及如何防御这种攻击的问题。在单个 agent 的 RL 中，一种常见的解决方法是将扰动行为建模成另一个与原 agent 进行零和游戏的对手，进行对抗训练。而在原本

就是 multi-agent 的环境中，adversarial 是指通过自己操作的 agent 的行为去干扰对方的决策——这个“干扰”并不是通常意义上的对抗，而是抓住对手潜在的弱点进行利用，其特点就是针对攻击对象高度特化。在这一方面的工作不多，其中最重要的工作是 Adversarial Policies: Attacking Deep Reinforcement Learning 一文[2]。其将被攻击目标视为环境的一部分，用单智能体的 PPO 进行攻击，并用 t-SNE 等方法分析结果，得到了能在 3%训练时间之内稳定打败对手、但实际上面对一般对手“毫无竞争力”的模型。不过，其在方法上没有创新。

博弈论和对抗环境的多智能体强化学习结合非常紧密，为多智能体强化学习提供了难得的理论保证；近年来，以 NFSP[25,26]和 CFR[27,28]为代表的算法逐渐走出了只能解决小规模问题的局限，已经可以解决一些 non-trivial 的即时战略游戏[12]和德州扑克[13]。实际上，尽管 NFSP 和 CFR 的如果对手的策略是完全固定的，那么 counterfactual regret minimization 和 fictitious self play 都可以做 adaptation——虽然这两种算法一般用于自我对局寻找纳什均衡。原因有二：其一，在对手策略固定的情况下，我们完全可以把游戏建模成 n*1 的游戏，其中 n 是自己的 policy 的个数，每一个 policy 对应一种 strategy；而对手只有唯一的 strategy，就是维持一个固定的 policy。这样，我们完全可以认为对手也是一个使用与我们相同方法学习的对手（尽管学习没有任何实际意义），从而达到我们对这一固定策略的最优回应——单行矩阵上的纳什均衡，一定是我方 payoff 的最大值。另一方面，这两个算法的计算过程本身就是对当前对手的回应计算最佳策略；而如果对手的回应充分多，也就能还原对手的 policy。

除此之外，[21]提出将 MADDPG 与 minimax-Q learning 结合来改进对抗状态下 agent 的表现，但 MADDPG 本身就不稳定，因此后续发展不多；另外，还有一种叫做“safe exploitation”的目标及其对应算法[34][14]。和其他所有的 adaptation 方式都不同，这种算法主要关心如何在保证探索过程中收益始终不低于某个特殊值的前提下“安全地”探索并利用对手缺乏理性的策略。其主要思路是根据当前的输赢情况和收益矩阵定义一个动作的“安全”集合，每次只尝试使用在安全集合中的动作；或者只在自己当前收益足够高时进行充分探索，否则使用安全集合中的策略，从而达到保证收益始终高于某个值的情况下偏离均衡去最大化收益。[44]提出从最初的某个纳什均衡开始，以达到均衡时对手的策略作为采取动作的先验概率，通过采样计算得到后验概率，并更新对手的模型、重新计算纳什均衡，如此不断迭代。采取类似方法的还有 OLSI[45]，它在每次决策后都会更新对手的模型，然后重新计算针对对手的最优策略并使用 soft update 进行迭代。

另外，如[29] [33]中提到的，还有一些针对非常特殊的游戏设计的算法：比如[15]假定对手基于过去的动作做决策（比如，囚徒博弈中的 tit-for-tat 战略），因此以“现在动作基于过去动作的条件概率”作为对手的模型，用序列预测的方法建模；[32]是将固定的几种策略建模成 multi-armed bandit，用 exp3 这种通常用于解决 online learning 的方法选择最优策略，但不能在没有固有策略的基础上求出最优策略；[19][20]这样更早的工作是专门针对 alpha-beta 剪枝的 minimax 搜索，通过估计对手的 alpha 和 beta 参数来建模，而[35]是 minimax 搜索在大于等于 3 个玩家时的扩展。总的来说，基于博弈论的 adaptation 算法主要注重理论保证，但应用场景经常受到局限。

在 opponent adaptation 中，最常用的方法是对对手建模；由于建模可以将多智能体问题近似转化为单智能体问题，几乎所有对对手建模的方法都可以在建模后针对模型做 adaptation。与对手建模相关的综述见[30]；这里只讨论和适应对手关系较强的工作。总的来说，从建模结果上看，可以分为分类适应（将对手提前分为几个固定的类别）、统计建模（基于贝叶斯公式直接对 policy 建模）和网络预测（预测对手的 policy）三种流派；从使用的方法上看，可以分为基于贝叶斯和基于网络。

与 adaptation 相关性较强的、使用建模方法的论文中，大部分都基于贝叶斯定理。这一方法中较早的尝试包括[17][18]。[17]认为对手的策略是 behavior strategy 而非 mixed strategy，

即将过去双方的动作作为决策条件；对手的策略是从某种概率分布中采样（比如在连续两轮的囚徒博弈中，有二分之一的几率是“第一轮被背叛则第二轮必背叛”，有二分之一的几率是“必然与第一轮对手选择的动作相同”），通过某个特定的先验策略和已有的观测得到对手策略的后验概率分布，再根据后验概率分布计算最优 response——但是，估计最优 response 需要对对手实际采取策略的概率分布进行积分，这里作者选择用先验概率估算，就使得整个算法非常依赖先验概率的选取。Hyper-Q learning[18]将对手的（按一定粒度离散化后的）mixed strategy 视为状态的一部分，用贝叶斯公式建模对手的策略。后续比较成功的工作包括[40]以及[46][4]，在已有一些策略的前提下如何找到合适的使用策略的概率分布来面对之前从未见过的 agent。

除上述文章之外，合作环境下的”ad-hoc teammates”问题[41]，其用贝叶斯建模的解决方案也对 opponent adaptation 有一定的指导意义。Ad-hoc teammate 指 agent 在部署之前不知道自己的队友会如何行动，必须在实践中快速适应队友的行为模式。[24]希望和未知的队友尽快达成高效合作，于是训练了一个由过去与不同队友配合组成的 policy pool，面对新对手时通过比较策略误差选择与过去最接近的队友相对应的策略。[42]提出了一种解决方案 PLASTIC，它假设队友都属于几类策略中的某一类，用收集到的 trajectory 数据做分类，通过推断队友的类型来适应队友。

也有一些算法从深度学习框架入手进行建模适应。DRON[36]和 DPIQN/DRPIQN[37] 都是通过增加辅助的“推断模块”，用预测对手行为这一监督学习问题作为辅助目标以指导 agent 适应对手；LOLA[38]将对手的梯度下降情况也纳入梯度更新考虑范围，但这样做方差较大，需要许多训练技巧。它的改进版本，SOS[39]，在 differentiable games（将游戏视为动力系统，用微分方程组表示其状态；最知名的例子是 GAN 中生成器和分类器的零和游戏）中通过一定的更新规则调整取得了基于稳定不动点的收敛保证。

另外，由于 VAE 可以对一类问题给出低秩编码，也可以通过编码来区分不同的对手决定的不同 MDP。目前已经有一些在 RL 中应用 VAE 的文章[47][48]；但直接运用于 multi-agent 环境中以建模不同对手的研究还不多。

还有一些算法尝试使用更高级、更接近通用人工智能的建模，如 ToMnet 使用神经网络将对手建模成具有“性格”和“精神状态”，分别对应长期行为和短期行为，以预测对手的动作[22]；但这样的算法目前还局限于较为简单的场景（比如规模不太大的 gridworld）中，尚不实用。

除此之外，还有一些零散的、启发式的，更倾向于工程手段的解决方案。DriftER[43]首先将对手视为静态 MDP 的一部分来学习最优策略，然后预测 MDP，如果预测错误率达到一定的程度就判定对手更改了策略，重新进入学习。另外，从过去的迭代版本中采样自己的对手[23]，也是常见的让 agent 避免过拟合、从而在 adaptation 中取得泛化性更好起点的技巧。

二、 重点文章论述

Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments

这篇文章是 meta-learning 应用于多智能体强化学习最重要（几乎也是唯一重要）的工作，它通过将 MAML 应用于多智能体强化学习，在 Robosumo 这样复杂的环境下解决了 Continuous adaptation 问题，即一个经过一定预训练的 agent 如何在有限的交互次数内适应执行时的环境变化。反映到多智能体问题上，就是在环境高度复杂、纳什均衡难以求得时，如何快速适应不同的对手。这篇文章的工作不完全是针对 multi-agent 环境的——也针对单智能体不断变化的环境，比如第一个实验是随机跛脚的机器人；但是它也考察了 multi-agent

环境下的适应，并且能够对从未见过的对手较快地持续学习。

MAML[5]是 meta-learning 近年来的标杆算法。它的基本思想，是假设待完成的一族任务满足某种概率分布，在某个流形上聚集在一起；那么就首先通过梯度下降找到这些任务“聚集”的区域作为 adaptation 的起点，再在执行时更新策略。这篇文章的创新之处，就在于其将这种算法成功应用到复杂环境下的多智能体强化学习中，以应对不同的对手。

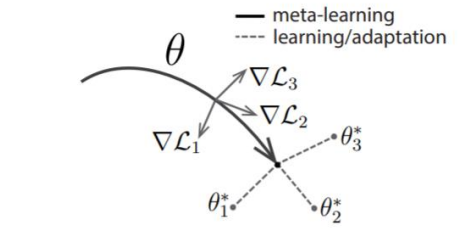


Figure 1. Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

图 1：直观理解 MAML 的基本思路。

Algorithm 3 MAML for Reinforcement Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K trajectories $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using f_θ in $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample trajectories $\mathcal{D}_i' = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using $f_{\theta_i'}$ in $\mathcal{T}_i$
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
11: **end while**

图 2：原论文中提出的 MAML 用于 multi-task RL 的算法流程。

MAML 将每个特定的对手和环境绑定，视为一个有限步长的 task T：

$$T := \left(L_T, P_T\left(\mathbf{x}\right), P_T\left(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{a}_t\right), H\right)$$

其中 L_T 是根据对手确定的 MDP 下的 reward 函数，x 是 observation，中间的两个概率分布（起始状态与状态转移）由对手策略和环境共同决定，H 是持续长度。一族中所有的 task 会在部署时按照顺序出现组成序列 T1, T2, …, Tn，训练时给出 P(Ti, Ti+1)的概率分布，用这个概率分布进行采样；对应到我们的问题中，每个 T 就对应一种对手。

MAML 主要包括两套参数：其一是待适应的初始参数 θ 和学习率 α；其二是通过适应得到的 policy 参数 Φ。记在 θ 所决定的初始 policy 下，从 $T \sim \mathcal{D}(T)$ 这一 task 的概率分布中获得的 trajectory 为 $\boldsymbol{\tau}_\theta^{1:K}$，则部署时，用于适应的 policy 参数 Φ 通过以下式子计算得到：

$$\begin{aligned}
&\phi_i^0 := \theta, \quad \boldsymbol{\tau}_\theta^{1:K} \sim P_{T_i}(\boldsymbol{\tau} \mid \theta), \\
&\phi_i^m := \phi_i^{m-1} - \alpha_m \nabla_{\phi_i^{m-1}} L_{T_i}\left(\boldsymbol{\tau}_{i,\phi_i^{m-1}}^{1:K}\right), \quad m = 1, \ldots, M-1, \\
&\phi_{i+1} := \phi_i^{M-1} - \alpha_M \nabla_{\phi_i^{M-1}} L_{T_i}\left(\boldsymbol{\tau}_{i,\phi_i^{M-1}}^{1:K}\right)
\end{aligned}$$

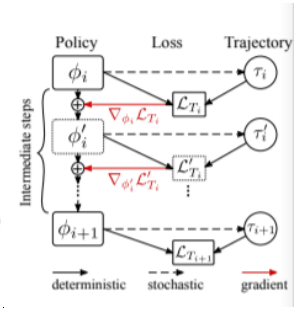


图 3
部署时的计算图

计算图如图 3 所示，Φ_i 和 Φ_(i+1)之间有几次梯度下降。

而 θ 和 α 的梯度（使用 policy gradient 算法）为

$$\nabla_{\theta,\alpha}\mathcal{L}_{T_i,T_{i+1}}(\theta,\alpha) =$$

$$\mathbb{E}_{\substack{\boldsymbol{\tau}_{i,\theta}^{1:K}\sim P_{T_i}(\boldsymbol{\tau}|\theta)\\ \boldsymbol{\tau}_{i+1,\phi}\sim P_{T_{i+1}}(\boldsymbol{\tau}|\phi)}}\left[L_{T_{i+1}}(\boldsymbol{\tau}_{i+1,\phi})\left[\nabla_{\theta,\alpha}\log\pi_\phi(\boldsymbol{\tau}_{i+1,\phi})+\nabla_\theta\sum_{k=1}^{K}\log\pi_\theta(\boldsymbol{\tau}_{i,\theta}^k)\right]\right]$$

直观地说，就是 θ 和 α 要最小化按这个初始参数和学习率 adapt 后的期望损失。由于原问题是需要解决单智能体变化环境的，这些环境可能有某种固定的变化规律，因此在公式中特别提到在训练中，Ti 和 Ti+1 按照某种实现给定的概率分布取样。公式中通过训练得到的目标 policy Φ 按照以下方式计算：（由于实际应用场景中有时不能为同一个 task sample 多次，因此需要复用过去的 trajectory，故引入重要性采样）

$$\phi_i := \theta - \alpha\frac{1}{K}\sum_{k=1}^{K}\left(\frac{\pi_\theta(\boldsymbol{\tau}^k)}{\pi_{\phi_{i-1}}(\boldsymbol{\tau}^k)}\right)\nabla_\theta L_{T_{i-1}}(\boldsymbol{\tau}^k),\quad \boldsymbol{\tau}^{1:K}\sim P_{T_{i-1}}(\boldsymbol{\tau}\mid\phi_{i-1}),$$

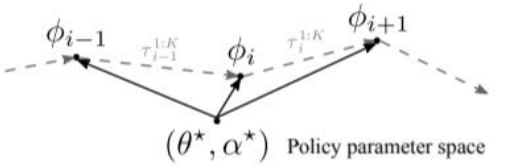
最后的算法如下表所示：

Algorithm 1 Meta-learning at training time.

input Distribution over pairs of tasks, $\mathcal{P}(T_i,T_{i+1})$, learning rate, β.
1: Randomly initialize θ and α.
2: **repeat**
3: Sample a batch of task pairs, $\{(T_i,T_{i+1})\}_{i=1}^n$.
4: **for all** task pairs (T_i,T_{i+1}) in the batch **do**
5: Sample traj. $\boldsymbol{\tau}_\theta^{1:K}$ from T_i using π_θ.
6: Compute $\phi=\phi(\boldsymbol{\tau}_\theta^{1:K},\theta,\alpha)$ as given in (7).
7: Sample traj. $\boldsymbol{\tau}_\phi$ from T_{i+1} using π_ϕ.
8: **end for**
9: Compute $\nabla_\theta\mathcal{L}_{T_i,T_{i+1}}$ and $\nabla_\alpha\mathcal{L}_{T_i,T_{i+1}}$ using $\boldsymbol{\tau}_\theta^{1:K}$ and $\boldsymbol{\tau}_\phi$ as given in (8).
10: Update $\theta\leftarrow\theta+\beta\nabla_\theta\mathcal{L}_T(\theta,\alpha)$.
11: Update $\alpha\leftarrow\alpha+\beta\nabla_\alpha\mathcal{L}_T(\theta,\alpha)$.
12: **until** Convergence
output Optimal θ^* and α^*.

Algorithm 2 Adaptation at execution time.

input A stream of tasks, $T_1,T_2,T_3,\ldots$.
1: Initialize $\phi=\theta$.
2: **while** there are new incoming tasks **do**
3: Get a new task, T_i, from the stream.
4: Solve T_i using π_ϕ policy.
5: While solving T_i, collect trajectories, $\boldsymbol{\tau}_{i,\phi}^{1:K}$.
6: Update $\phi\leftarrow\phi(\boldsymbol{\tau}_{i,\phi}^{1:K},\theta^*,\alpha^*)$ using importance-corrected meta-update as in (9).
7: **end while**



文章最终在 robosumo 环境下取得了良好成绩，并且可以较好地 adapt 从未见过的对手。这个方法的局限性在于不支持适应能在部署时迅速学习的对手；实验表明部署时表现随着每个 episode 对手能做 loss backward 的次数下降较快。

Bayesian Opponent Exploitation in Imperfect-Information Games

这篇文章继承了[17]的研究，在用贝叶斯公式建模对手行为的基础上，研究了在部分可观测环境下，使用 Dirichlet 分布（在一般的博弈中）或均匀随机分布（在博弈的单纯形上）作为先验分布来建模对手的算法，并给出了理论保证。之前所有的文章，或者是只是启发式算法、没有理论保证，或者是需要完全可观测的环境，再要不就是没有时间复杂度的理论保证。

Dirichlet 分布也被称为“分布的分布”，因为其本质是一个各维度和为 1 的向量的概率分布。因为其在用多项分布(multinomial)作为似然函数、计算后验分布后能保持原有的形式，故是最常见的用于建模对手的分布。假设对手在某个状态下有 K 个动作，动作概率（由于是不完全信息博弈，这里用我观测到的不同状态代替）分别为 x_1, x_2, …, x_n，x_1+…+x_n=1。如果每个这样的事件已经被观测到 a_i – 1 次，则认为这个 X 向量决定的概率分布出现的概率为

$$f(x_1,\ldots,x_K;\alpha_1,\ldots,\alpha_K)=\frac{1}{\mathrm{B}(\alpha)}\prod_{i=1}^{K}x_i^{\alpha_i-1}$$

如果只是要对对手建模之后做适应，算法框架其实非常简单，一句话概括就是“求到对手采取各种策略的概率分布，然后取平均值得到对手具体会采取的策略；再将建模后的 agent 视为环境的一部分，用单 agent 的方法求解最优 response”：

Algorithm 1 Meta-algorithm for Bayesian opponent exploitation

Inputs: Prior distribution p_0, response functions r_t for $0 \le t \le T$

$M_0 \leftarrow \overline{p_0(\sigma_{-i})}$
$R_0 \leftarrow r_0(M_0)$
Play according to R_0
for $t = 1$ to T **do**
  $x_t \leftarrow$ observations of opponent's play at time step t
  $p_t \leftarrow$ posterior distribution of opponent's strategy given prior p_{t-1} and observations x_t
  $M_t \leftarrow$ mean of p_t
  $R_t \leftarrow r_t(M_t)$
  Play according to R_t

其中 t 代表迭代轮数，r 是从对手策略到应对策略的函数（可以是 best response，也可以遵循有限理性使决策更鲁棒）， R 是最终我方的策略。考虑到一轮的后验概率要作为下一轮的先验概率则必须有封闭形式，实践中更合适的做法是将第 t 轮的 p_t 改成直接从 p_0 的先验经过所有 trajectory 的采样得到的结果。

需要特别注意的是：在这里，先验分布和后验分布都是“分布的分布”，因为单纯的“分布”是对手的 policy，而这里所说的先验分布和后验分布是 policy 的概率分布；所以才有 Mt 取平均来确定对手策略这一说。需要把“分布的分布”和简单的基于动作统计概率的方法相区分。

文章假定对手在某个状态下有若干可能的“私人状态”，这些状态在 episode 结束时会被展示。（例如，规定德州扑克每场游戏后要将所有人的牌翻开）。经过推导（过程略），后验分布的概率可以写成

$$\begin{aligned}P(q|O) &= \frac{\sum_i\left[\pi_i\sum_{\{\rho\}}\prod_h B(\gamma_{1j},\ldots,\gamma_{nj})\right]}{Z}\\ &= \frac{\sum_i\left[\pi_i\sum_{\{\rho\}}\exp\left(\sum_{j=1}^{m}\left(-\gamma_j E(\hat{P}_j)-\frac{1}{2}(n-1)\ln(\gamma_j)+\sum_{i=1}^{n}\ln(P_j(i))+d\right)\right)\right]}{Z},\end{aligned}$$

其中，最外层的求和是对对手所有在游戏中不为我们所知的私人状态求和，中间对 ρ 的求和是在枚举每个私人状态对应多少次动作（比如在德州扑克中，3 次选加注可能是手上 2 次真正有大牌、1 次耍诈，或者 1 次真正有大牌、2 次耍诈，又或者有大牌的次数是 0 或 3 次）。π i 是对手的私人信息为 x_i 的概率；j 是对所有实际做出的动作求和。Pj(i)是在 j 动作下私人状态为 i 的概率。d 是一个常数，满足 $\frac{1}{2}\ln(2\pi)n-1\le d\le n-\frac{1}{2}\ln(2\pi)$ 。n 和 m 分别是对手私人状态和动作的数量。γ j 是动作出现的次数+1。Z 是归一项。这个计算的时间复杂度是关于私人状态的数量（即整个游戏的状态数量）及动作数量的多项式级别的。

这篇文章尽管是贝叶斯建模的文章，但仍能看出很重的博弈论痕迹。其优势是理论严谨，缺点是实验环境只解决了很简单的问题。

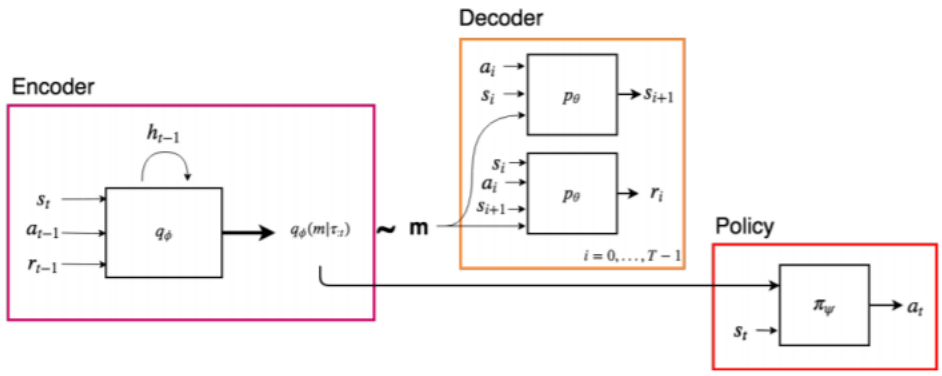
Variational Task Embeddings for Fast Adaptation in Deep Reinforcement Learning

这篇文章虽然是针对单 agent 的文章，但其采用 VAE 进行建模的方法可以应用在 multi-agent setting 中。实际上，的确有一篇文章将这种方法应用在 multi-agent setting 上[48]，但由于缺乏 novelty 和 contribution 而被 ICLR 2020 拒稿；不过这也说明 VAE 适应变化环境的特性可以很自然地移植到 opponent adaptation 这一任务中。该文章提出了一种新的算法架构 variational task embedding(VATE)，通过学习一个低维隐变量作为任务的嵌入表达，以适应不断变化的环境，解决了 BAMDP 问题（即 MDP 本身服从未知的概率分布而非定值）。它是一种在线算法，因此不需要像 MAML 一样明确区分数据收集、训练和部署的过程。

VATE 显式地用一个低维隐变量来表达任务，并认为 reward 和 transition 函数都是由隐变量决定的：

$$R_i(r_{t+1}|s_t, a_t, s_{t+1}) \equiv R'(r_{t+1}|s_t, a_t, s_{t+1}; m_i),$$
$$T_i(s_{t+1}|s_t, a_t) \equiv T'(s_{t+1}|s_t, a_t; m_i),$$

整个网络的 architecture 如下图所示：



其中，decoder 希望能通过 embedding 的向量预测 MDP，也就是希望提高是实际结果在预测中的对数概率，预测的越准确 loss 越低。预测 MDP 主要分为两部分：预测当前的 reward 和接下来的 transition 结果。

$$\begin{aligned}\log p(\tau_{:H}|m, a_{:H-1}) &= \log p((s_0, r_0, \ldots, s_{t-1}, r_{t-1}, s_t)|m, a_{:H-1}) \\ &= \log p(s_0|m) + \sum_{i=0}^{H-1} [\log p(s_{i+1}|s_i, a_i, m) + \log p(r_{i+1}|s_i, a_i, s_{i+1}, m)]\end{aligned}$$

最终的目标是

$$\mathcal{L}(\phi, \theta, \psi) = \mathbb{E}_{p(M)} \left[\mathcal{J}(\psi, \phi) + \lambda \mathbb{E}_\rho \sum_{t=0}^{H} ELBO_t(\phi, \theta) \right]$$

其中 Φ 是 encoder 的参数，ψ 是 policy 的参数，θ 是最终估计的 MDP 结果的参数，J 代表 reward 之和。而 ELBO(evidence lower bounds)是 VAE 的 loss function，它满足

$$\mathbb{E}_\rho \left[\frac{1}{H} \sum_{t=0}^{H} \mathbb{E}_{q_t} [\log p_\theta(\tau_{:H}|m)] - KL(q_\phi(m|\tau_{:t}) \| p_\theta(m)) \right] = \mathbb{E}_\rho \left[\frac{1}{H} \sum_{t=0}^{H} ELBO_t \right]$$

总而言之，这篇文章构建了一个简洁明了的利用 VAE 做 adaptation 的模型。潜在的问

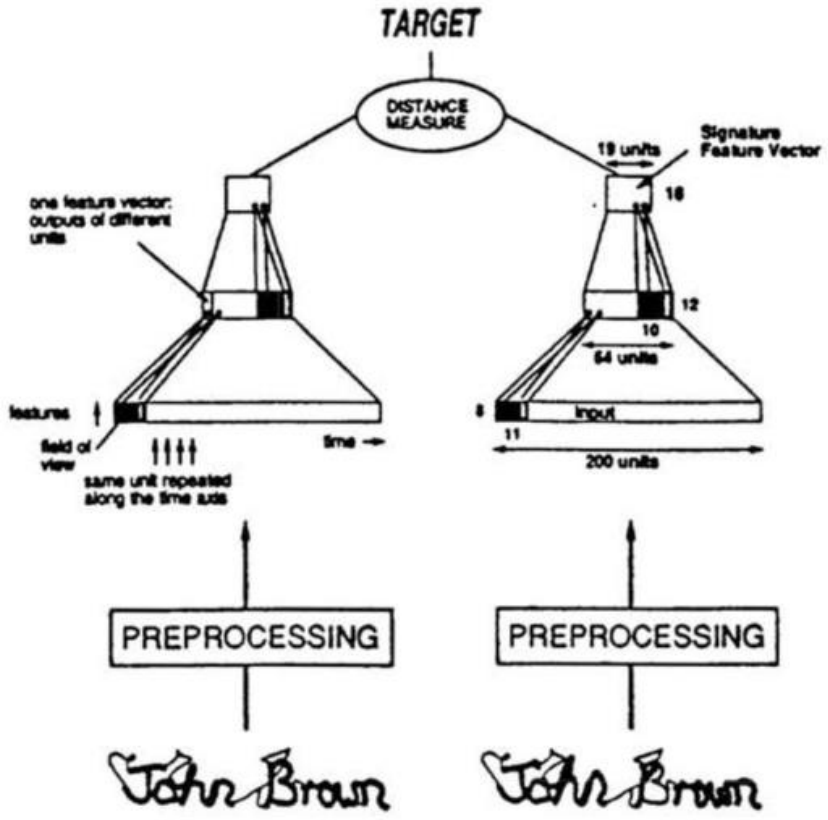
题是：文章中提到“隐变量的目的是学习 MDP 的概率分布”，这一点似乎和 multi-agent 环境的训练不甚兼容；如果真如文章所说，可以想象训练时对手采样情况不同会对 agent 的表现造成影响。

三、 可能的改进方向

以下是一些对 opponent adaptation 方向的改进思路，其中第一个和第二个都与我自己的研究更相关，而和 opponent adaptation 的关系小一些。opponent adaptation 本身的改进从目前来看，实际上更好的 modeling 算法才是发展的主流；实际上，这个问题本身也和 modeling 高度相关。

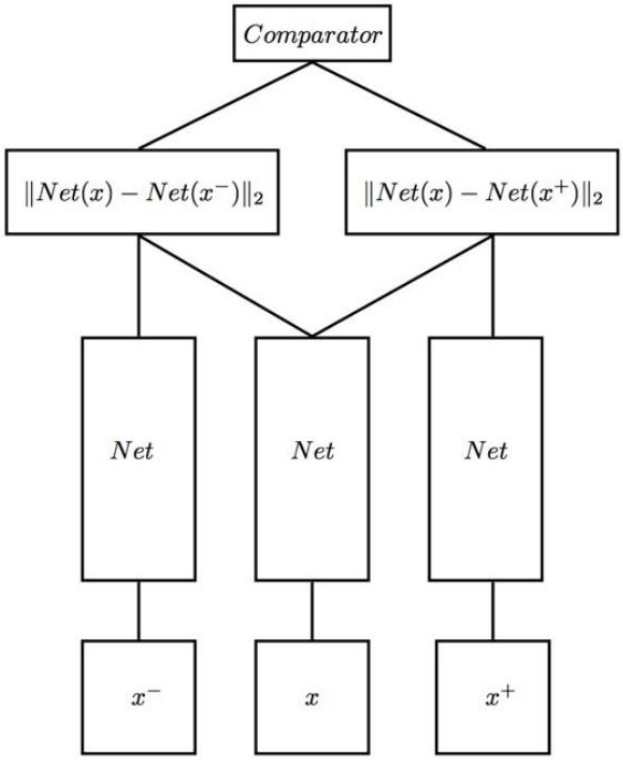
训练 Siamese Network 以区分不同的 agent：

Siamese network 是 meta-learning 的三大分支中基于 metric 的代表算法之一[7]（另两个分支分别是基于新结构，代表算法是神经图灵机；以及基于学习规则，代表算法是 MAML）。Siamese network 的结构如下图，其中两侧网络的权值可以共享也可以独立；它主要是用来计算两个输入的相似程度。



如果在 multi-agent 环境中，要对来自不同种类的 agent 的 trajectory 做二分类，则每次从网络的两个入口分别输入一个 trajectory 的 batch，这些 batch 采样自两类 agent，然后输出它们是同一类的概率（最后一层汇总为 softmax，然后用交叉熵为 loss function，以同类为 0 异类为 1 作为 target 做梯度下降）；softmax 的输出结果，或者最后一个全连接层，可以在 detach 后作为提供给 actor 模块的 embedding 辅助 actor 做出决策。

结合下面“数据增强”的想法，我们可以用 Siamese network 的变种——triplet network[49]的结构来训练面对对方 agent 的分类器：



其中，左边和右边每次分别从两类 agent 产生的 trajectory 中采样，而中间输入的 trajectory 其 policy 是左边对应 agent 的 policy 和右边对应 agent 的 policy 的线性组合。上面的两个距离，就是待训练的输出系数；希望分类器能学习到线性组合的系数。例如，中间的 trajectory 对应的 policy 是左 policy 和右 policy 以 0.3 和 0.7 的系数组合的，则最后的 loss 就是(x-0.3)^2 + (0.7-(1-x))^2。

数据增强：

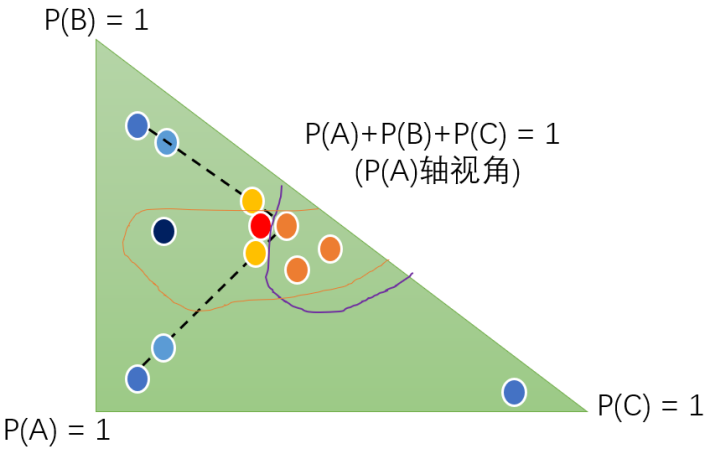
借鉴 adversarial learning 中的数据增强的算法。这主要是针对我自己的研究（一个需要给对手分类的 agent）而考虑的，同时也受到之前曾在 2019 年图灵大会上听到的某篇文章的启发——这篇文章把传统的图片分类改成训练分类器输出每一类的比例，新数据集里的每张图片都是由原数据集中两类样本以一定比例混合而成的。由此想到，对于 stochastic 的 policy 来说，policy 也是可以取（加权）平均的；那么我们可以在训练中加入一些把两类策略 mixed 起来的 agent 作为数据增强的新数据点。混合的公式与 NFSP 所用的公式相同：

$$\sigma(s,a) \propto \lambda_1 x_{\pi_1}(s)\pi_1(s,a) + \lambda_2 x_{\pi_2}(s)\pi_2(s,a) \forall s,a,$$

举例来说，如果原本在持续 10 轮的石头剪刀布游戏中有两类 agent，其中一类总是出石头，另一类总是出剪刀；那么两种策略按照 0.3:0.7 mixed 起来的结果就是以 30%的概率出石头，以 70%的概率出剪刀。注意这里一定是把策略 mix 起来，单纯地把 Q 值 mix 起来是不可取的；因为对于 deterministic policy 来说，policy 是随 Q 值的变化而突变的，无法达到产生这些样本的目的。另外，我曾经实践过去训练 policy 的 KL 散度差距之比满足某一特殊比例的 agent（例如，若与第一个 agent 的 KL 散度为 x，则与第二个 agent 的 KL 散度为 9x），结果不能收敛。

下面是算法的参考图示，将采取三种动作的概率视为坐标轴，以 P(A)为 Z 轴俯视。其中橙色和深蓝色点为两类 agent，红色点为应该被分为和橙色一类、但没有被观测到的 agent（蓝黑色点应该和深蓝色一类，同理未观测到），黄色点和浅蓝色点为增强的 agent；橙线和

紫线为两种不加数据增强时可能出现的错误分界线。



博弈论/online learning 启发下的 reward shaping：

正如之前部分所言，基于博弈论的适应算法虽然理论保证强大，但是首先能解决的问题规模一般比较有限，第二对场景的要求比较苛刻。但是，博弈论/online learning 中的概率计算公式，可以作为 reward shaping 引导 agent 尝试 exploitation。例如，CFR 在每一轮迭代中选择动作的公式如下：

$$\delta_i^{T+1}(I,a) = \begin{cases} \dfrac{R_i^{T,+}(I,a)}{\sum_{a\in A(I)} R_i^{T,+}(I,a)} & \text{if denominator} > 0 \\ \dfrac{1}{|A(I)|} & \text{otherwise.} \end{cases}$$

那么在部署在 adaptation 的环境中时，也可以基于上述公式添加 reward shaping。Regret 的计算与 CFR 算法相同，都是“在信息集 I 时选择状态 a 相对于实际选择获得的累计收益，与 0 取 max 得到的结果”。如果当前所有动作 regret 均不大于 0，则不做 reward shaping；否则做与上式成比例的 shaping。

又如在 multi-armed bandit 问题中，经典算法 Thompson Sampling 的流程分别为（假设 reward 在 0 到 1 之间）：

1. **for** $t = 1, 2, \ldots$ **do**
2. *//sample model*
3. **for** $k = 1, \ldots, K$ **do**
4. Sample $\hat{\theta}_k \sim \mathrm{Beta}(\alpha_k, \beta_k)$
5. **end for**
6. *//select and apply action:*
7. $a_t \leftarrow \arg\max_k \hat{\theta}_k$
8. Apply a_t and observe r_t
9. *//update distribution:*
10. $(\alpha_{a_t}, \beta_{a_t}) \leftarrow (\alpha_{a_t} + r_t, \beta_{a_t} + 1 - r_t)$
11. **end for**

其中 θ k 是第 k 个摇杆的收益，假设其是从 Beta 分布中采样得到。对于 stochastic off-

policy learner 来说，如果我们把游戏的每一步独立出来近似看成一个 multi-armed bandit 问题，其拉动拉杆获得的 reward 用 gamma * Q + reward 来近似，则可以把当前状态下不同的动作看成不同的拉杆，用 Thompson Sampling 的方式来选择动作，以期尽快探索到 exploit 程度较高的 trajectory。

参考文献

[1] Marc Lanctot, Vinícius Flores Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, Thore Graepel. A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning, NIPS 2017.

[2] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, Stuart Russell. Adversarial Policies: Attacking Deep Reinforcement Learning, ICLR 2020.

[3] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, Pieter Abbeel. Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments, ICLR 2018.

[4] Benjamin Rosman, Majd Hawasly, Subramanian Ramamoorthy .Bayesian Policy reuse, *Machine Learning*, 104, 99-127(2016)

[5] Chelsea Finn, Pieter Abbeel, Sergey Levine. MAML:Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks, ICML 2017.

[6] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, Enrique Munoz de Cote. A Survey of Learning in Multiagent Environments: Dealing with Non-Stationarity, arXiv:1707.09183

[7] Lilian Weng.https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html

[8] Georgios Papoudakis, Filippos Christianos, Arrasy Rahman, Stefano V. Albrecht.Dealing with Non-Stationarity in Multi-Agent Deep Reinforcement Learning.arXiv:1906.04737

[9] Yoav Shoham, Rob Powers, and Trond Grenager. If multi-agent learning is the answer, what is the question? Article in *Artificial Intelligence* 171(7):365-377, 2007.

[10] Lucian Busoniu, Robert Babuska, Bart De Schutter. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 38(2), 2008*

[11] Jialian Li, Hang Su, Jun Zhu .Sample-efficient policy learning in multi-agent Reinforcement Learning via meta-learning. 2018

[12] Keigo Kawamura, Yoshimasa Tsuruoka. Neural Fictitious Self-Play on ELF Mini-RTS. AAAI 2020

[13] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, Michael Bowling. DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker. arXiv:1701.01724

[14] Sam Ganzfried, Tuomas Sandholm. Safe Opponent Exploitation. EC 2012.

[15] Richard Mealing, Jonathan L. Shapiro. Opponent Modelling by Sequence Prediction and Lookahead in Two-Player Games. ICAISC 2013.

[16] Gregory Palmer, Karl Tuyls, Daan Bloembergen, Rahul Savani.Lenient Multi-Agent Deep Reinforcement Learning. AAMAS 2018.

[17] Finnegan Southey , Michael Bowling , Bryce Larson , Carmelo Piccione , Neil Burch , Darse Billings , Chris Rayner.Bayes' Bluff: Opponent Modelling in Poker. UAI 2005.

[18] Gerald Tesauro. Extending Q-Learning to General Adaptive Multi-Agent Systems. NIPS 2003.
[19] David Carmel, Sahul Markovich. Learning models of opponent's strategy in game playing. 1993.
[20] Andrew L. Reibman, Bruce W. Ballard. Non-minimax search strategies for use against fallible opponents. 1983.
[21] Shihui Li, Yi Wu, Xinyue Cui, Honghua Dong, Fei Fang, Stuart J. Russell . Robust Multi-Agent Reinforcement Learning via Minimax Deep Deterministic Policy Gradient. AAAI 2019.
[22] Neil C. Rabinowitz, Frank Perbet, H. Francis Song, Chiyuan Zhang, S.M. Ali Eslami, Matthew Botvinick . Machine Theory of Mind. arXiv: 1802.07740.
[23] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, Igor Mordatch . Emergent Complexity via Multi-Agent Competition. ICLR 2018.
[24] Samuel Barrett, Peter Stone. Cooperating with Unknown Teammates in Complex Domains: A Robot Soccer Case Study of Ad Hoc Teamwork. AAAI 2015.
[25] Johannes Heinrich, David Silver.Deep Reinforcement Learning from Self-Play in Imperfect-Information Games. arXiv: 1603.01121.
[26] Keigo Kawamura, Naoki Mizukami, Yoshimasa Tsuruoka. Neural Fictitious Self-Play in Imperfect Information Games with Many Players. CGW 2017
[27] Noam Brown, Adam Lerer, Sam Gross, Tuomas Sandholm. Deep Counterfactual Regret Minimization. ICML 2019.
[28] Eric Steinberger. Single Deep Counterfactual Regret Minimization. arXiv:1901.07621.
[29] Tuomas Sandholm. Steering Evolution Strategically: Computational Game Theory and Opponent Exploitation for Treatment Planning, Drug Design, and Synthetic Biology. AAAI 2015.
[30] Stefano V. Albrecht, Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. arXiv:1709.08071
[31] Andrea Tacchetti, H. Francis Song, Pedro A. M. Mediano, Vinicius Zambaldi, János Kramár, Neil C. Rabinowitz, Thore Graepel, Matthew Botvinick, Peter W. Battaglia. Relational Forward Models for Multi-Agent Learning. ICLR 2019.
[32] Bret Hoehn, Finnegan Southey, Robert C. Holte, Valeriy Bulitko. Effective short-term opponent exploitation in simplified poker. AAAI 2005
[33] Tuomas Sandholm. The state of solving large incomplete-information games, and application to poker. *AI Magazine*, 31(4), 13-32, 2010.
[34] Peter McCracken, Michael Bowling. Safe strategies for agent modelling in games. AAAI 2004.
[35] Nathan Sturtevant, Martin Zinkevich, Michael Bowling. Prob-maxn: Opponent modeling in n-player games. AAAI 2006.
[36] Zhang-Wei Hong, Shih-Yang Su, Tzu-Yun Shann, Yi-Hsiang Chang, Chun-Yi Lee. A Deep Policy Inference Q-Network for Multi-Agent Systems. AAMAS 2017.
[37] He He, Jordan Boyd-Graber, Kevin Kwok, Hal Daumé III. Opponent Modeling in Deep Reinforcement Learning. arXiv:1609.05559.
[38] Jakob N. Foerster, Richard Y. Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, Igor Mordatch. Learning with Opponent-Learning Awareness. AAMAS 2018.
[39] Alistair Letcher, Jakob Foerster, David Balduzzi, Tim Rocktäschel, Shimon Whiteson. Stable Opponent Shaping in Differentiable Games. ICLR 2019.
[40] Sam Ganzfried, Qingyun Sun. Bayesian Opponent Exploitation in Imperfect-Information Games. arXiv: 1603.03491.

[41] Peter Stone, Gal A. Kaminka, Sarit Kraus, and Jeffrey S. Rosenschei. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. AAAI 2010.
[42] Samuel Barrett, Peter Stone. Cooperating with Unknown Teammates in Complex Domains: A Robot Soccer Case Study of Ad Hoc Teamwork. AAAI 2015.
[43] Pablo Hernandez-Leal, Yusen Zhan, Matthew E. Taylor, L. Enrique Sucar, Enrique Munoz de Cote. Efficiently Detecting Switches Against Non-Stationary Opponents. AAMAS 2017.
[44] Sam Gazfried, Tuomas Sandholm. Game Theory-Based Opponent Modeling in Large Imperfect-Information Games. AAMAS 2011.
[45] Pablo Hernandez-Leal, Micheal Kaisers. Learning against sequential opponents in repeated stochastic games. 2017.
[46] Yan Zheng, haopeng Meng, Jianye Hao, Zongzhang Zhang, Tianpei Yang, Changjie Fan. A deep Bayesian Policy Reuse Approach against Non-stationary agents. NIPS 2018.
[47] Luisa Zintgra, Maximilian Igl, Kyriacos Shiarlis, Anuj Mahajan, Katja Hofmann, Shimon Whiteson University of Oxford. Variational Task Embeddings for Fast Adaptation in Deep Reinforcement Learning, ICLR workshop 2019.
[48] Georgios Papoudakis, Stefano V. Albrecht. Variational Autoencoders for Opponent Modeling in Multi-Agent Systems.2019.
[49] Elad Hoffer, Nir Ailon. Deep Metric Learning Using Triplet Network.arXiv:1412.6622.