强化学习中的 Policy Gradient Theorem 和 Policy Proximal Optimization

强化学习是一个很有意思的领域, 因为在强化学习中, AI 是通过和环境的交互来不断学习, 而不是很依赖预先提供的训练数据. 我们可能都听到过一些术语, 比如 agent, reward function, environment, state 等等, 那么这些名词到底是什么? 接下来我们就用数学语言来定义一下强化学习的背景.

强化学习背景的数学描述

强化学习中有两个主体, 一个是 agent, 可以是电脑, 机器人, 一个程序等等, 还有一个就是 environment, 可以是真实的世界, 也可以是电子游戏世界等等. agent 可以做出动作或者操作, 这称为 action. 同时 agent 还有一个state, 代表了它现在的状态和处境. 可以想见, agent 的操作必然要基于状态, 那么这就形成了一个从 state 到 action 的映射, 这就是 policy. 我们用 s 表示 state, 用 a 表示 action, 用 π 表示策略, agent 在状态 s 选择策略 a 的概率就是 $\pi(a|s)$. 显然, action 会影响下一时刻 agent 所处的状态, 同时每一次 action 之后 agent 会得到环境的一个 reward. 因此, agent 在状态 s 采用操作 a 之后变为状态 s' 并且得到 reward r 的概率就可以写为 $p(s', r|s, a)$, 而 p 就成为这个环境的 dynamics. **大部分情况下, dynamics 我们是不知道的.** 所以为了获取信息, 我们只能通过 sample 的方法不断更新丰富我们对于环境的认识.

> 在接下来的论述中, a, s, r 分别表示 action,state, reward. A, S, R 表示随机变量.

在强化学习中, 一旦有了一个策略, 我们就关心策略的好坏, 用数学语言来刻画就是 reward 之和的大小. 对于一个状态 s , 我们定义一个策略下该状态的平均收益为

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \cdots | S_t = s\right] = \mathbb{E}\left[G_t | S_t = s\right]$$

其中 G_t 是从 stpe t 开始的总收益. 我们会发现, 并不是直接将所有的 reward 加起来, 而是多了一个 γ, 这称为 discounting factor, 第一个好处就是可以防止加出无穷, 第二个就是调整我们对眼前利益的重视程度, γ 越小, 眼前利益就显得越重要. 这里的 v 我们称为 value function.

value function 是从一个状态出发, 根据策略得到的平均收益. 相似的, 我们还可以定义从一个状态 s 出发并且采取操作 a 所得到的平均收益, 定义为

$$q_\pi(s, a) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = a\right] = \mathbb{E}\left[G_t | S_t = s, A_t = a\right]$$

这个我们成为 state-action value function.

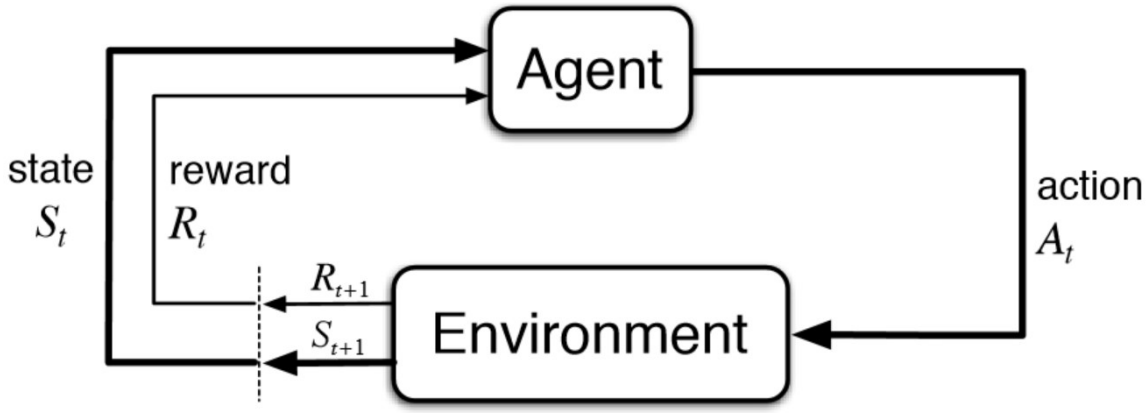
根据概率论的知识, 我们不难推出关于 value function 和 state-action value functon 的递归式子

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)\left(r + \gamma v_\pi(s')\right)$$

$$q_\pi(s, a) = \sum_{s', r} p(r, s'|s, a)\left(r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')\right)$$

这就是著名的 Bellman equations.

在强化学习中, 因为环境的 dynamic 通常是不知道的, 所以 agent 需要和环境不断交互来学习, 更新自己的策略.



Policy Gradient Theorem

有了上面的数学定义, 接下来我们就想想在 RL 中我们到底想做什么. 总结来说, 我们一共有两种目标

- 给定 policy π, 我们想找到 v_π 和 q_π, 这叫做 policy evaluation.
- 我们想找到一个 policy π^*, 使得这个策略是最优的(reward 最多的), 这叫做 policy improvement.

这次我们讲的是 policy gradient method, 简单来说, 就是将 policy 变为一个含参函数 π_θ, 通过调整 θ 的值来使得策略最优. 在深度学习中我们都知道, 这时我们应该找一个关于 θ 的函数, 我们想最大化/最小化这个函数, 在此过程中我们通过梯度/上升的方法来优化并得到参数 θ. 一个自然的函数就是 G_t 的期望, 也就是平均收益. 为了完善这个期望的定义, 我们需要定义 $d(s)$ 为 agent 从 s 状态出发的概率. 然后我们就可以写出我们想要最大化的函数

$$J(\theta) = \mathbb{E}_{\pi_\theta, d}\left[G_t\right]$$

那么我们的首要任务就是计算

$$\nabla_\theta J(\theta) = \nabla \mathbb{E}_{\pi_{\theta,d}}\left[G_t\right]$$

你会发现, 这个东西几乎无法直接计算, 那么我们要对它进行变形.

假设 G_t 经过路径 $\tau = S_0, A_0, R_1, S_1, A_1, R_2, \cdots$, 并且假设 $G(\tau)$ 是这条路径上的总收益, 那么我们可以知道

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta, d}\left[G(\tau)\right] \\
&= \nabla_\theta \sum_\tau p(\tau) G(\tau) \\
&= \sum_\tau \nabla_\theta p(\tau) G(\tau)
\end{aligned}$$

其中 $\tau \sim \pi_\theta$, d 表示 τ 是根据策略 π_θ 起始状态分布 d 得到的. 我们发现, 给定一个路径 τ, 路径上的收益 $G(\tau)$ 是和策略无关的, 所以我们可以得到

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_\tau G(\tau) \nabla_\theta p(\tau) \\
&= \sum_\tau p(\tau) \left(G(\tau) \frac{\nabla_\theta p(\tau)}{p(\tau)} \right) \\
&= \sum_\tau p(\tau) \left(G(\tau) \nabla_\theta \log p(\tau) \right) \\
&= \mathbb{E}_{\tau\sim\pi_\theta, d}\left[G(\tau) \nabla_\theta \log p(\tau) \right]
\end{aligned}$$

乍一看这个变换好像没什么用, 但是**我们把要求的东西变成了一个期望**, 这时我们就可以通过取样的方法获得这个量的估计并且进行梯度上升, 这种方法叫做 **随机梯度上升**(SGA). 但是这个形式还不是很好取样计算, 我们继续进行变形

$$\begin{aligned}\nabla_\theta \log p(\tau) &= \nabla_\theta \log \left[p(S_0)\pi_\theta(A_0|S_0)p(S_1,R_1|S_0,A_0)\pi_\theta(A_1|S_1)\cdots\right] \\ &= \nabla_\theta \left[\log p(S_0) + \log \pi_\theta(A_0|S_0) + \log p(S_1,R_1|S_0,A_0) + \log \pi_\theta(A_1|S_1) + \cdots\right]\end{aligned}$$

注意, p 是环境的 dynamics, 虽然我们不知道 p, 但是它是和 θ 无关的, 所以梯度是零. 因此我们得到

$$\nabla_\theta \log p(\tau) = \nabla_\theta \left[\log \pi_\theta(A_0|S_0) + \log \pi_\theta(A_1|S_1) + \cdots\right]$$

带回去, 我们得到

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta,d}\left[G(\tau)\nabla_\theta \sum_{t=0}^{T} \log \pi_\theta(A_t|S_t)\right]$$

其中我们假设路径 τ 长度为 T, 这时 τ 也可以被称为一个 episode. 这个时候我们已经以进行采样了, 因为对于一个样本 τ, $\log \pi_\theta(A_t|S_t)$ 这个函数我们是完全知道的, 梯度也是可以求出来的. 然后我们就可以不断进行采样, 进行随机梯度上升得到最好的 θ. 看起来问题是解决了. 但是科学家们还不满足, 他们又有了新的发现, 能再次优化这个式子.

Baseline

不知道你有没有想过这个问题, 玩一定局数的石头剪刀布, 赢了获得 3 分, 平局获得 1 分, 输了获得 0 分. 如果我们把 reward 同时平移一下, 比如 赢了获得 2 分, 平局 0 分, 输了 扣一分, 那么你的策略会变吗? 仔细想一想, 策略应该是不变的.

基于这个想法, baseline 方法被创造了出来, 从数学上来说, 是这样子的:

$$\begin{aligned}\mathbb{E}\left[b\nabla_\theta \log \pi_\theta(A_t|S_t)\right] &= \sum_s p(S_t=s)\sum_a \pi_\theta(a|s) b\nabla_\theta \log \pi_\theta(a|s) \\ &= \sum_s p(S_t=s)\sum_a \nabla \pi_\theta(a|s) \\ &= \sum_s p(S_t=s)\nabla_\theta \sum_a \pi_\theta(a|s) \\ &= \sum_s p(S_t=s)\nabla_\theta 1 \\ &= 0\end{aligned}$$

只要 b 和 action 无关这个式子就是成立的(注意, b 可以和 state 相关, 这不影响). 也就是说, 在期望中出现 $\nabla_\theta \log_\pi(A_t|S_t)$ 的地方, 你个可以随意加上 $b(s)$ 倍的 $\nabla_\theta \log_\pi(A_t|S_t)$, 这不会影响期望的大小. 那我们又要问了, 这有什么用呢? 从平均来看, 这确实没什么用, 但是你会发现单次采样的值都变了, 从另一方面来说, 你采样的方差变了, 这就给了我们一个机会, 如果我们精心选择 b, 我们可以使得采样的方差降低.

继续回到 J_θ, 我们发现, 在一次采样中, τ 路径上的后续的步骤仍然用的是总的收益 $G(\tau)$, 这会使得方差很大, 我们可以做如下优化

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta,d}\left[G(\tau)\sum_{t=0}^{T}\log\pi_\theta(A_t|S_t)\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta,d}\left[\sum_{t=0}^{T}G(\tau)\nabla_\theta\log\pi_\theta(A_t|S_t)\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta,d}\left[\sum_{t=0}^{T}\left(\sum_{k=0}^{T}\gamma^k R_{k+1}\right)\nabla_\theta\log\pi_\theta(A_t|S_t)\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta,d}\left[\sum_{t=0}^{T}\left(\sum_{k=t}^{T}\gamma^k R_{k+1}\right)\nabla_\theta\log\pi_\theta(A_t|S_t)\right]
\end{aligned}
$$

仔细看看我们干了什么: 我们把从 0 到 T 的求和变成了从 t 到 T 的求和, 我们能这样做的原因正是基于 baseline, 因为

$$
\sum_{k=0}^{t-1}\gamma^k R_{k+1}
$$

这个随机变量和 S_t 有关联但是和 A_t 无关, 所以可以舍去. 这样, 我们就得到

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta,d}\left[\sum_{t=0}^{T}\left(\sum_{k=t}^{T}\gamma^k R_{k+1}\right)\nabla_\theta\log\pi_\theta(A_t|S_t)\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta,d}\left[\sum_{t=0}^{T}\left(\gamma^t\sum_{k=t}^{T}\gamma^{k-t} R_{k+1}\right)\nabla_\theta\log\pi_\theta(A_t|S_t)\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta,d}\left[\sum_{t=0}^{T}\gamma^t G_t\nabla_\theta\log\pi_\theta(A_t|S_t)\right]
\end{aligned}
$$

这就是 Policy Gradient Theorem. 我们发现, 期望之中的式子不需要我们对环境的 dynamic 任何的了解, 这就是这个定理的强大之处. 我们可以通过采样 τ 然后进行随机梯度上升, 进而优化策略.

再使用 baseline 的想法, 我们可以给每一个状态 s 一个 baseline $b(s)$, 这时我们就可以得到

$$
\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T}\gamma^t(G_t - b(S_t))\nabla_\theta\log\pi_\theta(A_t|S_t)\right]
$$

这里有一个概念: Advantage, 在状态 s, 采用 action a 的 advantage 定义为

$$
A_\pi(s,a) = q_\pi(s,a) - v_\pi(s)
$$

一般来说, 我们希望 $b(s)$ 接近 $v(s)$, 这样 advantage 的期望是零, 能进一步减少方差. 这样, 我们有了一个大致的算法: 不断对 τ 进行采样, 每采样一次进行一次随机梯度上升, 同时不断维护对于 advantage 的估计 $\hat{A}$, 直到收敛(维护 advantage 是 policy evaluation 问题, 受篇幅所限, 假设我们已经有了比较好的办法进行 policy evaluation).

Proximal Policy Optimization

人们不会满足于 policy gradient theorm. 我们发现, 基于此的采样算法确实很好, 但是每次梯度上升的步长却很难确定. 如果步长小了, 那么收敛就要很久, 如果步长大了, 策略就可能出现严重偏差, 这会出现严重问题, 因为我们训练的数据就是基于策略产生的, 糟糕的策略会带来糟糕的数据. 同时, 看我们刚刚的方法, 对一个数据我们仅仅梯度上升一次, 而并没有梯度上升直到达到极值.

回想一下深度学习模型, 我们都是对同一组数据进行梯度下降/上升直到达到极值, 在强化学习之中我们也想这样做, 对数据进行充分的利用.

Surrogate Function

为了解决上述问题, 科学家们进行了一系列精彩操作. 假设 $\rho_\pi(s)$ 表示在一个路径中 s 衰减平均出现的次数, 即

$$\rho_\pi(s) = p(s_0 = s) + \gamma p(s_1 = s) + \gamma^2 p(s_2 = s) + \cdots$$

然后我们会发现, 对于两个策略 π, $\tilde{\pi}$, 有

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a)$$

其中 $\eta(\pi) = \mathbb{E}\left[G_t\right]$ 就是策略 π 之下的平均收益.

然后, 人们将上述式子做了微小改动, 定义了

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a)$$

乍一看好像是一样的, 只是把 $\rho_{\tilde{\pi}}$ 换成了 ρ_π, 但是从后面看, 这个式子是非常方便采样的. 同时, 如果把 π 写成含参函数形式, 定义

$$L_\theta(\tilde{\theta}) = L_{\pi_\theta}(\pi_{\tilde{\theta}})$$

人们证明了, 如果 $\pi_{\tilde{\theta}}$ 距离 π_θ 不太远, 那么 L 近似就等于 η. 所以, 在优化的过程中, 我们就需要**控制策略的变化不能太多**, 一种是使用 KL-divergence 进行控制, 这就是 TRPO 方法, 是我们今天要介绍的 PPO 方法的前身. 我们在把目光放到 L 上, 我们发现

$$\begin{aligned}\sum_a \pi_\theta(a|s_n) A_{\theta_{\text{old}}}(s_n, a) &= \sum_a \pi_{\theta_{\text{old}}}(a|s_n) \left(\frac{\pi_\theta(a|s_n)}{\pi_{\theta_{\text{old}}}(a|s_n)} A_{\theta_{\text{old}}}(s_n, a) \right) \\ &= \mathbb{E}_{a \sim \pi_{\theta_{\text{old}}}(\cdot|s_n)} \left[\frac{\pi_\theta(a|s_n)}{\pi_{\theta_{\text{old}}}(a|s_n)} A_{\theta_{\text{old}}}(s_n, a) \right]\end{aligned}$$

再加上 $\rho_{\theta_{\text{old}}}$ 就是

$$L_{\theta_{old}}(\theta) = \eta(\pi_{\theta_{\text{old}}}) + \mathbb{E}_{s \sim \rho_\pi, a \sim \pi_{\theta_{\text{old}}}(\cdot|s)} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}} A_{\theta_{\text{old}}}(s, a) \right]$$

我们发现, 从采样的角度出发, 我们就需要在保证策略变化不太多的情况下调整 θ 最大化

$$\hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right]$$

这个式子就称为 surrogate function. 其中 $\hat{A}_t$ 表示对 $A_{\theta_{\text{old}}}$ 的估计, $\hat{\mathbb{E}}_t$ 表示这个期望是通过采样得来的. 这个时候我们把 η 写成 L 的好处就体现出来了, 如果不这样做我们是不能写成这种期望形式的. 仔细看这个式子我们会发现, 其中只有 $\pi_\theta(a_t|s_t)$ 是和 θ 有关的, 因此求梯度是十分方便的. 这时, 我们的目标就变成了优化 surrogate function.

TRPO

为了保证策略变化的不是太多, TRPO 方法被提了出来. 我们需要

$$\text{maximize}_\theta \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right]$$

同时

$$\text{subject to } \hat{\mathbb{E}}_t \left[\text{KL} \left[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t) \right] \right] \leq \delta$$

其中 KL 表示 KL 散度, 这个约束条件用来控制策略变化不是太多. 但是有约束的求极值问题我们实际上是很难处理的, 因为这时候直接梯度上升是行不通的. 所以 TRPO 实现起来比较困难.

PPO

终于, 我们要进入 proximal policy method 了. 前面我们说过, 用 L 近似 η 只有在策略变化不太大的时候才是有效的, TRPO 正是基于这一点加上了一个约束条件, 在约束条件之下最大化 surrogate function, 但是这实现起来很难. 在此基础上, PPO 算法通过巧妙地变形, 使得算法实现起来变得快速, 容易. 目前, PPO 及其衍生算法基本上是 RL 中最强大的算法.

我们定义

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

那么 surrogate function 就变为更简洁的形式

$$\hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right]$$

我们引入 clip 函数, 定义为

$$\text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right) = \begin{cases} 1-\epsilon, & \text{if } r_t(\theta) < 1-\epsilon \\ 1+\epsilon, & \text{if } r_t(\theta) > 1+\epsilon \\ r_t(\theta), & \text{else} \end{cases}$$

然后我们把需要最大化的函数定义为

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

我们来直观地理解一下这个函数: $r_t(\theta)$ 代表了新策略和就策略的偏差, 距离 1 越远偏差越大. clip 函数忽略了过大偏差带来的影响. 对两者取最小值我们就会发现, 策略偏差过多带来的优化效应被忽略了, 但策略偏差带来的负面效应被保留了. 所以我们可以说, 在取到最大值的时候, 新策略相比于旧策略应当不会变化太多. 因为如果新策略变化太多, 由此带来的正面影响被忽略, 但负面影响却被保留, 这会导致 L^{CLIP} 不会太大.

这样, 我们有了 PPO 大致的算法. 假设我们可以较好地估计出 $\hat{A}_t$ 的值(这需要用到一些 policy evaluation, temporal difference method 的知识), 在每一轮, 我们使用旧的策略和环境交互, 得到一系列 a_t, s_t, 然后算出 $\hat{A}_t$, 接着 **不断使用梯度上升**(例如可以用 pytorch 自动计算), 得到 $L^{\text{CLIP}(\theta)}$ 的 **极大值**, 获得新的策略. 再更新我们对 $\hat{A}_t$ 的估计(policy evaluation). 不断重复这种操作直到收敛.

我们可以看出, 对于一组数据, 我们不断梯度上升直到达到极值, 这相比于 policy gradient theorem 得到的算法, 对于数据的利用是非常充分的. 同时, 每一次更新策略都保证了不会变化太大, 这使得策略的跟新是非常稳定的. 同时, 相比于 TRPO, PPO 的实现非常简便, 可以直接求梯度. 这都是 proximal policy optimization 的优点.

在 2019 年四月十三日, 由 OpenAI 用 PPO 算法训练的 OpenAI Five 成功击败 dota II 世界冠军 OG 队, 称为历史上第一个在电竞上击败世界冠军队伍的人工智能系统. OpenAI Five 用的就是 PPO 算法进行训练. 在训练中, AI 逐渐学会了许多 moba 的技巧. AI 学会了对残血小兵补刀, 学会了攻击敌方英雄进而吸引小兵仇恨达到控制兵线阵型, 学会了在上线的时候卡兵延缓并线前进从而使得兵线靠近己方防御塔, 同时还学会了在优势时压制对手不让他吃兵, 同时还能理解技能前摇. 这些技能都是 moba 中的重要元素, 而 AI 在强化学习中都学会了. 而且 OpenAI Five 不但能自己组队, 甚至还能和人类玩家组队, 可见 PPO 算法的强大.

(在比赛后, OG 队员难掩失落, 而 OpenAI 工程师则疯狂庆祝)

参考资料:

- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
- *DeepMind x UCL | Deep Learning Lecture Series 2021. (n.d.)*. YouTube. https://www.youtube.com/playlist?list=PLqYmG7hTraZDVH599EItlEWsUOsJbAodm
- *Foundations of Deep RL -- 6-lecture series by Pieter Abbeel. (n.d.)*. YouTube. https://www.youtube.com/playlist?list=PLwRJQ4m4UJjNymuBM9RdmB3Z9N5-0IlY0
- Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015, June). Trust region policy optimization. In *International conference on machine learning* (pp. 1889-1897). PMLR.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., ... & Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.