Санкт-Петербургский государственный университет
Кафедра математического моделирования экономических систем

Осипов Анатолий Александрович

Магистерская диссертация

Применение контрфактуального градиентного алгоритма группового обучения для моделирования конфликтно-управляемых мультиагентных систем

Направление 01.04.02 - «Прикладная математика и информатика»
Магистерская программа
«Прикладная математика и информатика в задачах медицинской диагностики»

Научный руководитель:
кандидат физ.-мат. наук,
ассистент
Петросян О. Л.

Санкт-Петербург
2019 г.

Содержание

Определения и сокращения

В данной работе используются следующие сокращения:

- RL - reinforcement learning - обучение с подкреплением
- ML - machine learning - машинное обучение
- DQN - deep q-network - глубокая q-сеть
- RNN - recurrent neural network - рекуррентная нейронная сеть
- LSTM - long short-term memory - нейронная сеть с долгой кратковременной памятью
- COMA gradient policy - counterfactual multi-agent gradient policy - контрфактуальный градиентный метод группового обучения

Введение

В настоящее время всё большее распространение получает искусственный интеллект, который может применяться для решения огромного количества задач. Одним из перспективных его видов является обучение с подкреплением. Это вид машинного обучения, при котором агент обучается, взаимодействуя с некоторой средой. В нём используются несколько иные подходы к обучению, нежели в классическом обучении. Обучение с подкреплением стало активно развиваться благодаря таким событиям как:

- В 1997 году в Нью-Йорке чемпион мира по шахматам Гарри Каспаров проиграл компьютеру «Deep Blue», фирмы IBM;
- В 2016 году программа AlphaGo компании Deepmind одолела чемпиона Кореи Ли Седоля по игре «Го»;
- На киберспортивном турнире International по Dota2 в 2018 году были проведены матчи между людьми и машинами, а именно Dota2 OpenAI Five (от компании OpenAI), в которых последним удалось выиграть несколько раз лучших игроков мира.

Как мы видим, искусственный интеллект развивается достаточно быстро и уже имеется ряд предметных областей[44], где может быть задействовано обучение с подкреплением, помимо игровой индустрии:

- Беспилотные автомобили;
- Рекомендательные системы;
- Робототехника;
- Торговля на бирже.

Задача, которая рассматривается в настоящей работе, называется обучение с покреплением(reinforcement learning) и очень похожа на изучение человеком окружаещего мира. В детстве, когда мы не знали многие вещи, мы узнавали, например, что огонь излучает тепло, но если слишком сильно приблизить руку к нему, то будет больно и возможно останется

ожог. Тепло и ожог - это своеобразные отклики окружающей среды на наши действия. В этой сравнительной модели человек является агентом, мир вокруг человека - окружающая среда, а тепло и боль - это так называемые награды за то, что мы сделали. Когда нам было больно, мы понимаем, что так делать не нужно и стараемся избегать таких ситуаций. Также и в модели обучения с подкреплением агент старается принимать решения, основываясь на награде за свои действия, при этом ничего не зная об окружающей среде.

Таким образом, цель обучения с подкреплением - сделать поведение некоторых объектов, называемых агентами, которые умеют передвигаться в своей системе, реалистичным или проще говоря, похожим на человеческое. Есть те, кто считают, что обучение с подкреплением - это область обучения с учителем. Но есть существенная разница между двумя этими направлениями обучения. В классическом обучении с учителем, чтобы научить некоторый аппроксиматор, в роли которого используется нейронная сеть, находить правильные связи, требуются данные, в которых есть так называемые метки учителя. То есть каждая отдельная единица данных должна быть оценена каким-то независимым учителем, чаще всего в роли которого выступает человек или группа людей, чтобы гарантировать точность оценки. Это нужно для того, чтобы нейронная сеть, используемая в задаче, сравнивала свой ответ с правильным ответом и за счёт этого могла обучаться. Таким своеобразным учителем в RL выступает окружающая среда, которая даёт свою оценку действиям агента в виде сигналов скалярных наград. Этот факт связывает два вышеупомянутых класса обучения, но различает их то, что для обучения с подкреплением не требуется никаких данных, чтобы начать обучение. Агент сам в процессе обучения, методом проб и ошибок поймёт какое поведение от него хотят.

Но есть также и те, кто относят обучение с подкреплением к классу задач обучения без учителя. В этих задачах испытуемая система спонтанно обучается выполнять поставленную задачу без вмешательства со стороны экспериментатора. Но, как правило, это пригодно только для задач, в которых известны описания множества объектов (обучающей выборки), и требуется обнаружить внутренние взаимосвязи, зависимости, закономерности,

существующие между объектами. Таким образом здесь также требуются данные для обучения.

Учитывая всё вышесказанное, можно сказать, что обучение с подкреплением это третий, независимый класс машинного обучения.

Одной из важнейших областей искуственного интеллекта являются нейронные сети. Они используются везде - в медицине, в экономике, в бизнесе, робототехнике, охранных системах, в автоматизации производства и многих других.

В обучении с подкреплением также используются нейронные сети. Обычно её задают в архитектуре агента[1], потому что именно он учится совершать действия, которые будут давать наибольшую награду, ведь мы хотим, чтобы было именно так. Поэтому можно сказать, что после каждого сделанного действия агента, получив отклик системы на это действие, агент делает соответствующие выводы, а иначе говоря - корректирует веса нейронной сети таким образом, чтобы увеличить награду. Это не говорит о том, что агент теперь будет постоянно делать одно и то же действие, если отклик был положительным, или наоборот не будет его делать, если отклик был отрицательным. Это говорит о том, что если агент снова будет в той или схожей ситуации, что и раньше, когда совершил полезное (или неполезное) действие, то он сделает его ещё раз (или не сделает) с большей вероятностью. Данные выводы возможны благодаря нейронным сетям, а именно - их возможности изучать различные, порой очень сложные связи, которые порой невозможно заметить человеку, в определённых данных.

Целью данной работы является применение алгоритма мультиагентного обучения с подкреплением к моделированию реалистичного поведения агентов в игре в футбол. При этом необходимо не просто реализовать искусственный интеллект, а использовать при этом самые современные технологии в данной области. Для достижения данной цели необходимо решить следующие задачи:

1. Осуществить постановку одноагентной, а также мультиагентной задач;

[1] Существуют различные подходы к решению задач обучения с подкреплением. В данном случае мы говорим о простом естественном случае. О других, более сложных подходах будет говориться далее.

2. Осуществить обзор существующих подходов к решению этих задач;
3. Описать алгоритм, выбранный для решения основной задачи;
4. Осуществить разбор сложных моментов в настройке данного алгоритма;
5. Спроектировать и разработать программу;
6. Провести вычислительные эксперименты, показывающие эффективность применения алгоритма обучения с подкреплением к модели игры в футбол.

Постановка задачи

0.1 Одноагентная задача

Обучение с подкреплением является областью машинного обучения, которое изучает, как агент должен действовать в окружении, чтобы максимизировать долговременный выигрыш[1].

Пусть у нас есть некоторая окружающая среда(environment) и в ней находится один агент. Представим время дискретной величиной $t = \{1, ..., T_e\}$, где T_e - конечный момент времени, при котором заканчивается одна эмуляция процесса, индекс e показывает какая именно эмуляция. Величина T - не обязана быть одинаковой в каждой эмуляции. В каждый момент времени система находится в определённом состоянии $s_t \in S$, при $t \in \{1, ..., T_e\}$, а агент выбирает действие $u_t \in U$, при $t \in \{1, ..., T_e\}$. В ответ на это действие система даёт сигнал агенту в виде скалярной величины - награду R_t(reward) и переходит в другое состояние s_{t+1}, если $t \neq T_e$. Также мы предполагаем, что множества S и U - конечны.

Таким образом вышеописанная задача представляет собой кортеж:

$$G = \langle S, U, P_u, R_u \rangle, \tag{1}$$

где:

- S - конечное множество состояний среды, s_t - это состояние среды в момент времени t;
- $U = \{u_1, ..., u_n\}$ - конечное множество действий агента, n - количество возможных действий агента, u_t - действие агента в момент времени t;
- $P_u(s, s') = P(s_{t+1} = s' | s_t = s, u_t = u)$ - это вероятность того, что действие u в состоянии s во время t приведёт к состоянию s' в момент $t + 1$;

[1] Об этом будет сказано далее.

- $R_u(s, s')$ - это немедленное вознаграждение за переход из состояния s, в состояние s', вследствие действия u;

Основная задача такой модели заключается в том, чтобы найти «политику» для лица, принимающего решения, то есть агента: функцию π, которая определяет действие $\pi(s)$, которое агент выберет, находясь в состоянии s [1].

Ожидаемое вознаграждение на каждом временном шаге t можно записать так:

$$H_t = R_{u_{t+1}}(s_{t+1}, s_{t+2}) + R_{u_{t+2}}(s_{t+2}, s_{t+3}) + ... + R_{u_{T_e-1}}(s_{T_e-1}, s_{T_e}) = \sum_{t}^{T_e} R_{u_{t+1}}(s_{t+1}, s_{t+2})$$

Однако на самом деле мы не можем просто так добавлять такие награды. Награды, которые приходят раньше(в начале эмуляции), более вероятны, так как они более предсказуемы, чем будущие вознаграждения. Допустим, агентом является маленькая мышь. Цель игры состоит в том, чтобы съесть максимальное количество сыра, прежде чем быть съеденным кошкой(рис. 1).

Как мы можем видеть на изображении ниже, скорее всего мышь будет есть сыр рядом с собой, вместо того, чтобы есть сыр, расположенный рядом с кошкой. Поскольку за то, что нас поймает кошка, будет наказание в виде отрицательного вознаграждения. А чем ближе мы к кошке, тем она опаснее. Поэтому ценность награды рядом с кошкой, даже если она больше чем та, которая расположена рядом с агентом, будет снижена, поскольку у агента нет уверенности, что он сможет её достигнуть.

Перерасчёт ожидаемого вознаграждения делается следующим образом: определяем коэффициент дисконтирования - $\gamma \in [0, 1]$, который показывает насколько сильно агент должен обращать внимание на будущие награды. То есть чем меньше γ, тем больше агент акцентирует своё внимание на награды в недалёком будущем. И наоборот, чем больше γ, тем

[1] Когда речь идёт о состоянии системы, мы подразумеваем собой также состоянии агента. Потому что агент и система неразрывно связаны. В качестве примера состояния s могут быть координаты агента в системе. Таким образом, говоря о состоянии системы имеют ввиду положение(как пример) агента в этой системе.

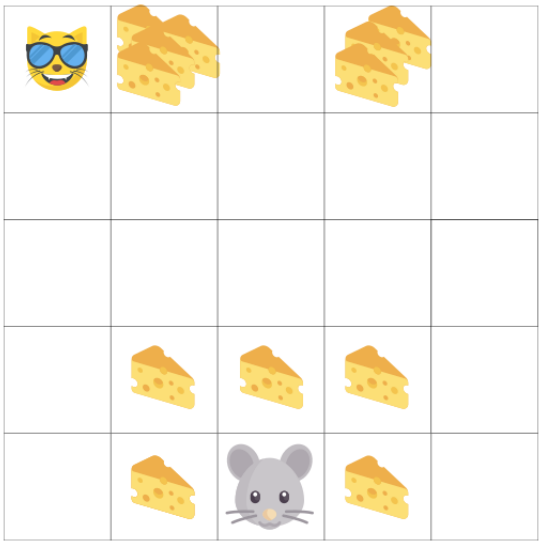

Рис. 1: Возможная среда для обучения с подкреплением.

сильнее агент задумывается о выгоде от своих будущих действий.

Таким образом к кортежу 1 можно добавить ещё и γ. Тогда получается кортеж:

$$G = \langle S, U, P_u, R_u, \gamma \rangle. \tag{2}$$

Переобозначим $r_t = R_{u_t}(s_t, s_{t+1})$, $R_t = H_t$. Таким образом ожидаемое вознаграждение можно рассчитать так:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^t r_{t+k+1}$$

Опишем алгоритм одной эмуляции игры с такой математической моделью:

1. Инициализация стратегии $\pi_1(u|s_1)$ и состояния среды s_1;
2. Агент выбирает действие $u_1 \sim \pi_1(u|s_1)$;
3. Среда даёт награду агенту r_1 за действие u_1 и переходит в новое состояние s_2 с вероятностью перехода $P_{u_1}(s_1, s_2)$;
4. Агент корректирует стратегию $\pi_2(u|s_2)$;
5. Агент выбирает действие $u_2 \sim \pi_2(u|s_2)$;

6. Среда даёт награду агенту r_2 за действие u_2 и переходит в новое состояние s_3 с вероятностью перехода $P_{u_2}(s_2, s_3)$;

7. **Для всех $\mathbf{t = 3, ..., T_e}$.**

8. Агент корректирует стратегию $\pi_t(u|s_t)$;

9. Агент выбирает действие $u_t \sim \pi_t(u|s_t)$;

10. Среда даёт награду агенту r_t за действие u_t и переходит в новое состояние s_{t+1} с вероятностью перехода $P_{u_t}(s_t, s_{t+1})$.

Замечания.

1. Формально политика представляет собой отображение состояний на вероятности выбора каждого возможного действия. Если агент следует политике π в момент времени t, то $\pi(u|s)$ это вероятность того, что $u_t = u$, если $s_t = s$. То есть π это просто функция. «|»в середине $\pi(u|s)$ просто напоминает, что она определяет распределение вероятностей по $u(s) \in U$ для каждого $s \in S$.

2. Знак $\sim$ означает «соответствие »;

3. Если мы установим, что нейронная сеть будет определять политику агента π, то корректировака политики означает корректировку весов в данной нейросети.

   Если

$$P(s_{t+1} = s', r_{t+1} = r|s_t, u_t, r_t, s_{t-1}, u_{t-1}, r_{t-1}, ..., s_1, u_1, r_1) =$$

$$= P(s_{t+1} = s', r_{t+1} = r|s_t, u_t),$$

то есть, выбор действия на текущем шаге зависит от предыдущего состояния, то такую модель можно охарактеризовать как Марковский процесс принятия решений(МППР)[50], поскольку выбор действия агентом в каждый момент времени t зависит от предыдущего момента $t - 1$.

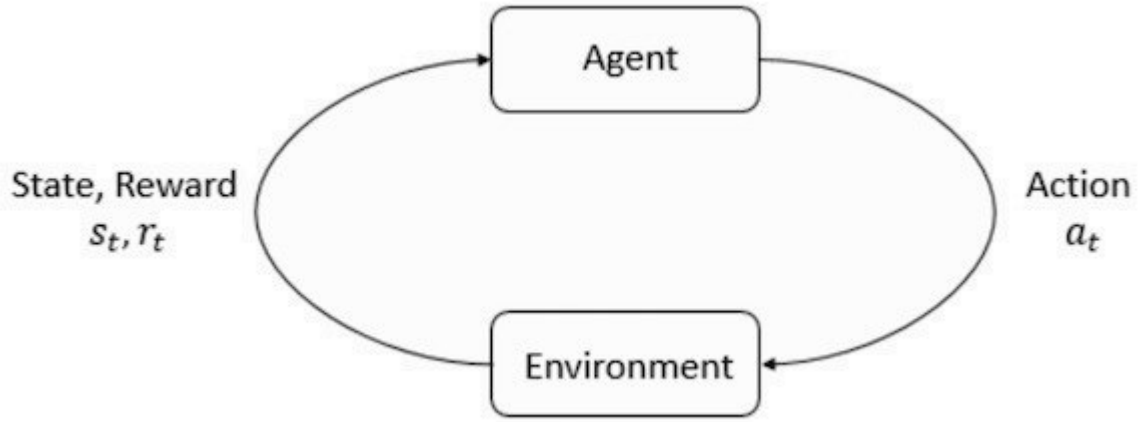

Рис. 2: Модель обучения с подкреплением с одним агентом.

Данная модель изображена на рисунке 2.

Задача агента заключается в нахождении политики π, которая описывает наилучшее действие для каждого состояния системы, то есть неохо-димо найти оптимальную политику путём проб и ошибок. Наилучшие действия влекут наибольшие награды для каждого состояния. То есть можно сказать, что цель обучения с подкреплением состоит в том, чтобы найти такую политику агента, которая максимизирует ожидаемое вознаграждение.

В данной задаче есть следующие проблемы[16]. Награда за совершённое агентом действие может даваться ему не сразу. Простой пример: змейка. Вы получаете награду только при съедании определённого предмета, который может быть расположен в любом месте игровой области. Поэтому процесс обучения может вестись достаточно долго, потому что для получения какой-либо обратной связи от окружающей среды агент должен сделать определённую последовательность действий. Также представим себе ситуацию, когда в той же упомянутой выше игре у нас возникла ситуация, где есть 2 предмета. При достижении одного агент получает награду +1, а при достижении другой +100. Но второй предмет находится в рискованном месте, где больше вероятность проигрыша. При неправильной настройке параметров обучения, агент никогда не достигнет большой награды, а будет сосредоточен только на лёгком достижении небольшой

награды, потому что он даже и не узнает о том, что есть награда больше той, которая рядом. Эта трудность называется балансом разведки и эксплуатации. О существующих подходах к решению данных проблем будет рассказано в следующей главе.

0.2 Мультиагентная задача

Также существуют модели, в которых фигурируют несколько агентов. Мультиагентную задачу RL принято рассматривать[13] как кооперативную, стохастическую игру G, определяемую набором:

$$G =< S, U, P, r, Z, O, n, \gamma >, \tag{3}$$

в котором:

- n агентов обозначаемых $a \in A \equiv \{1, ..., n\}$, выбирают последовательность действий;
- Среда имеет состояние $s \in S$;
- В каждый временной шаг каждый агент одновременно с другими агентами выбирают действие $u^a \in U$, образуя совместное действие $\mathbf{u} \in \mathbf{U} \equiv U^n$;
- Совместное действие $\mathbf{u}$ индуцирует переход в среде, согласно функции перехода состояния: $P(s'|s, \mathbf{u}) : S \times \mathbf{U} \times S \to [0, 1]$;
- Все агенты имеют одинаковую функцию вознаграждения $r(s, \mathbf{u}) : S \times \mathbf{U} \to R$;
- $\gamma \in [0, 1]$ - коэффициент дисконтирования, как и в задаче, описанной в предыдущем параграфе.
- Агенты проводят наблюдения $z \in Z$ в соответствии с функцией наблюдения $O(s, a) : S \times A \to Z$;
- Каждый агент имеет историю наблюдений действий: $\tau^a \in T \equiv (Z \times U)'$;

- У каждого агента на его локальной истории наблюдения действий задаётся стохастическая политика(или стратегия): $\pi^a(u^a|\tau^a) : T \times U \to [0, 1]$;

- Дисконтированный доход равен: $R_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l}$;

- Совместная политика агентов индуцирует функцию значения(value-function), то есть математическое ожидание от R_t :
  $V^{\pi}(s_t) = E_{s_{t+1:\infty}, u_{t:\infty}}[R_t|s_t]$. Данная функция может потребоваться в методе обучения для того, чтобы агент мог понять, насколько хорошо для него находиться в данном состоянии s_t;

- Также совместная политика агентов индуцирует функцию значения-действия(action-value function), которую чаще всего называют просто Q-функцией (Q-function): $Q^{\pi}(s_t, u_t) = E_{s_{t+1:\infty}, u_{t+1:\infty}}[R_t|s_t, u_t]$. Она, в свою очередь даёт агенту понимание, насколько хорошо выполнять данное действие u_t в данном состоянии s_t;

- А также есть функция преимущества, которая использует разность двух вышеупомянутых функций, а именно: $A^{\pi}(s_t, u_t) = Q^{\pi}(s_t, u_t) - V^{\pi}(s_t)$. Она же показывает насколько лучше в данном состоянии s_t выбрать именно u_t, действуя в соответствии с политикой π^a.

Функции значения, значения-действия и преимущества используются в различных алгоритмах обучения с подкреплением для улучшения политик агентов или для обучения критика, основная задача которого - оценивание действий агентов, действующих по своим политикам.

В данном случае целью обучения с подкреплением является получение такой политики, которая максимизировала бы суммарное значение вознаграждений всех агентов.

Основными проблемами в мультиагентных задачах, помимо упомянутых проблем одноагентных задач, являются следующие проблемы:

- Нестационарность среды. В одноагентной модели обучения с подкреплением агент стремится как можно шире узнать окружающую среду и приспособиться к ней. А в мультиагентной задаче это невозможно

ввиду того, что агенты действуют в среде все вместе, меняя свои действия по мере обучения, а не действуют всегда одинаково. Поэтому к такой среде невозможно приспособиться;

- Экспоненциальный рост вычислений. При наличии большого числа агентов, количество необходимых вычислений, проводимых на каждой итерации обучения может быть очень велико. Во многом это зависит от эффективности применяемого для обучения метода, а также от конкретного числа агентов. Так или иначе для обхода данной проблемы необходимо улучшать существующие алгоритмы, либо улучшать системы, на которых производятся вычисления.

Обзор литературы

Машинное обучение (machine learning) - это научное исследование алгоритмов и статистических моделей, которые используются компьютерными системами для эффективного выполнения конкретной задачи без использования явных инструкций, вместо этого опираясь на шаблоны и умозаключения. Машинное обучения является областью искуственного интеллекта. Алгоритмы машинного обучения строят математическую модель на основе выборочных данных, известных как «обучающие данные», для того, чтобы делать прогнозы или решения без явного программирования для выполнения задачи. Алгоритмы машинного обучения используются в широком спектре приложений, таких как фильтрация электронной почты или компьютерное зрение, где невозможно разработать алгоритм конкретных инструкций для выполнения задачи. Машинное обучение тесно связано с вычислительной статистикой, которая фокусируется на прогнозировании с использованием компьютеров. Изучение математической оптимизации позволяет появление методов, теорий и областей применения машинного обучения.

Название машинного обучения было придумано в 1959 году Артуром Самуэлем [34].

Задачи машинного обучения подразделяются на несколько широких категорий. При обучении с учителем алгоритм строит математическую модель из набора данных, который содержит как входные данные, так и требуемые выходные данные. Например, если задача заключалась в том, чтобы определить, содержит ли изображение определённый объект, обучающие данные для алгоритма обучения с учителем будут включать изображения с этим объектом и без него(входные данные), а также каждое изображение будет иметь метку(выходные данные), указывающую, содержало ли изображение данный объект. Таким образом проводится обучение на большом количестве данных, чаще всего разделяемых на обучающую выборку и контрольную, для того, чтобы система не могла бы запомнить все существующие варианты. Когда система достигает достаточной точности на контрольных данных, она готова для использования на других данных.

Алгоритмы классификации и алгоритмы регрессии являются алгоритмами обучения с учителем. Алгоритмы классификации используются, когда количество выходов ограничены конечным набором значений. Для алгоритма классификации, который фильтрует электронные письма, входными данными будут входящие электронные письма, а выходными данными будет имя папки, в которую следует отправлять электронную почту. Для алгоритма, который идентифицирует электронные письма со спамом, результатом будет предсказание «спам» или «не спам», представленное логическими значениями «истина» и «ложь». Алгоритмы регрессии названы из-за их непрерывных выходов, что означает, что они могут иметь любое значение в пределах диапазона. Примерами непрерывного значения являются температура, длина или цена объекта.

При обучении без учителя алгоритм строит математическую модель из набора данных, который содержит только входные данные и не имеет желаемых выходных меток. Алгоритмы обучения без учителя используются для нахождения структуры данных, такой как группировка или кластеризация точек данных. Обучение без учителя может обнаруживать закономерности в данных и может группировать входные данные по категориям.

Ещё одной областью машинного обучения является обучение с подкреплением. Оно построено на обратной связи среды и агента, действующего в ней. При этом агент ничего не знает о среде. Он учится делать действия на основе сигнала, получаемого им от среды, который говорит ему о том, хорошо или плохо то, что он сделал. Таким образом, обучение с подкреплением очень похоже на реальную модель обучения человека в природе, где человек методом проб и ошибок учится познавать мир. У него нет инструкций правильных действий, как нет и инструкций неправильных действий.

Многие сложные задачи обучения с подкреплением, такие как координация автономных транспортных средств, сетевая доставка пакетов и распределённая логистика, естественно, моделируются как кооперативные мультиагентные системы. Однако методы RL, разработанные для моделей с одним агентом, обычно плохо справляются с такими задачами, посколь-

ку пространство совместных действий агентов и пространство наблюдений растут экспоненциально с числом агентов[13].

Чтобы справиться с такой сложностью, часто приходится прибегать к децентрализованной политике, в которой каждый агент выбирает своё собственное действие, обусловленное только его локальной историей наблюдения за действиями. Кроме того, частичная наблюдаемость и коммуникационные ограничения во время исполнения могут потребовать использования децентрализованных политик, даже если пространство для совместных действий не слишком велико.

Следовательно, существует большая потребность в новых методах RL, которые могут эффективно изучать децентрализованные политики. В некоторых случаях само обучение может также потребовать децентрализации. Однако во многих случаях обучение может проходить на тренажере или в лаборатории, в которой доступна дополнительная информация о состоянии, и агенты могут свободно общаться. Эта централизованная подготовка децентрализованных политик является стандартной парадигмой для мультиагентного планирования [31],[23] и недавно была замечена сообществами глубокого RL [11],[19]. Однако вопрос о том, как наилучшим образом использовать возможности для централизованного обучения, остается открытым.

Еще одна важная задача - назначение кредитов нескольким агентам [6]: в кооперативных условиях совместные действия обычно генерируют только глобальные вознаграждения, что затрудняет каждому агенту осознавать свой вклад в успех команды. Иногда можно разработать индивидуальные функции вознаграждения для каждого агента. Тем не менее, эти вознаграждения обычно недоступны в кооперативных условиях и часто не побуждают отдельных агентов жертвовать ради общего блага. Это нередко существенно препятствует многоагентному обучению в сложных задачах, даже при относительно небольшом количестве агентов.

Хотя RL с несколькими агентами применялось в различных условиях [5],[47], он часто ограничивался табличными методами и простыми средами. Единственным исключением является недавняя работа в глубоком многоагентном RL, которая может масштабироваться до многомерных

пространств ввода и действия. Так в [40] используют комбинацию DQN и независимого Q-обучения [41],[35], чтобы научиться играть в понг между двумя игроками. Совсем недавно этот же метод был использован в [25], чтобы изучить возникновение сотрудничества и предательства в последовательных социальных дилеммах.

Кроме того, недавние работы на возникновение связи между агентами, обучаемых посредством градиентного спуска – [7],[30],[24],[11],[37]. В этом направлении статей прохождение градиентов между агентами во время обучения и совместное использование параметров являются двумя распространенными способами использования преимуществ централизованного обучения. Однако эти методы не позволяют использовать дополнительную информацию о состоянии во время обучения и не затрагивают проблему мультиагентного распределения наград.

В [15] исследовали методы актера-критика с децентрализованным исполнением с централизованным обучением. Тем не менее, в их методах условия как актеров так и критиков о частных, для каждого агента своих наблюдениях и действиях, а также о назначении мультиагентных наград рассматриваются только с помощью собственноручно присваиваемых частных наград.

В большинстве приложений RL к микроуправлению StarCraft используется централизованный контроллер с доступом к полному состоянию и управлением всеми устройствами, хотя архитектура контроллеров использует многоагентный характер проблемы. В [43] используют жадный MDP, который на каждом временном шаге последовательно выбирает действия для агентов с учетом всех предыдущих действий в сочетании с оптимизацией нулевого порядка, в то время как в [32] используют метод актер-критик, который использует RNN для обмена информацией между агентами.

Наиболее современным является решение статьи [12], авторы которой также используют мультиагентное представление и децентрализованную политику. Однако они фокусируются на стабилизации воспроизведения опыта при использовании DQN и не в полной мере используют режим централизованного обучения.

В [26] также предлагают мультиагентный алгоритм градиента поли-

тики с использованием централизованных критиков. Их подход не касается мультигентного присвоения наград. Он обучает отдельного централизованного критика для каждого агента и применяется к конкурентной среде с непрерывным пространством действий.

Помимо одного из самых современных алгоритмов мультиагентного обучения с подкреплением COMA[13], особенностью данной работы является модель, к которой необходимо применить этот метод, а именно модель игры в футбол. Задача обучения агентов игры в футбол с применением обучения с подкреплением описывается в работах [27] [36]. Ни в одной из них не был реализован метод COMA или его модификации.

В работе [13] метод COMA был применён к задаче микроуправления в игре Starcraft. В самой статье приведён псевдокод алгоритма, но настройки, которые применяются в реализации обучения с подкреплением, необходимо выбрать не опираясь на данную работу, поскольку мультиагентные модели существенно отличаются. Например, в модели микроуправления Starcraft основной задачей агентов является правильное распределение сил и позиционирование для того, чтобы уничтожить команду противника. В модели же футбола целью агентов является рациональное видение поля и перемещение для забивания голов в ворота противника.

Глава 1. Методы обучения в RL задачах

В данной главе приводится описание классического метода обучения с подкреплением - глубокое Q-обучение(DQN), основных видов нейросетей, а также мультиагентный метод обучения - COMA.

1.1 DQN

Одним из стандартных подходов к решению задачи обучения с подкреплением считается алгоритм DQN. Он был эффективно применён командой исследователей компании DeepMind Technologies к играм Atari, в который данный алгоритм показал результаты, в некоторых играх превышающие лучшие результаты игры человека.

Рассмотрим кортеж 2:

$$G = \langle S, U, P_u, R_u, \gamma \rangle.$$

с задачей выбора агентом таких действий, которые бы максимизировали величину ожидаемого дисконтированного дохода: $R_t = \sum_{k=0}^{\infty} \gamma^t r_{t+k+1}$. Если мы добавим в этот кортеж политику π, то сможем переформулировать задачу обучения с подкреплением, подходящую к Q-обучению.

Итак, имеем кортеж

$$G = \langle S, U, P_u, R_u, \pi, \gamma \rangle.$$

где:

- S - конечное множество состояний среды, s_t - это состояние среды в момент времени t;
- $U = \{u_1, ..., u_n\}$ - конечное множество действий агента, n - количество возможных действий агента, u_t - действие агента в момент времени t;
- $P_u(s, s') = P(s_{t+1} = s' | s_t = s, u_t = u)$ - это вероятность того, что действие u в состоянии s во время t приведёт к состоянию s' в момент

$t+1$;

- $R_u(s, s')$ - это немедленное вознаграждение за переход из состояния s, в состояние s', вследствие действия u;
- Политика(стратегия) π - функция которая определяет распределение вероятностей по $u(s) \in U$ для каждого $s \in S$;
- $\gamma \in [0, 1]$ - коэффициент дисконтирования, который влияет на важность бущущих наград для агента;

И тогда задачей обучения с подкреплением в данном виде является максимизация ожидаемого дисконтированного дохода путём выбора политики π:

$$R_t = \sum_{k=0}^{\infty} \gamma^t r_{t+k+1} \to \max_{\pi}$$

В Q-обучении для решения данной задачи вводится Q-функция - называемая также функцией значения-действия:

$$Q^{\pi}(s, u) = E_{\pi}[R_t | s_t = s, u_t = u] = E_{\pi}[\sum_{k=0}^{\infty} \gamma^t r_{t+k+1} | s_t = s, u_t = u].$$

Q-функция ставит в соответствие паре состояние - действие математическое ожидание вознаграждения при принятии этого действия в данном состоянии, а также она показывает полезность действия u_t в состоянии s_t действуя согласно политике π.

Предполагая, что дальше агент будет следовать той же политике π, мы можем переписать функцию в рекуррентном виде:

$$Q^{\pi}(s, u) = E_{s'}[r(s, u) + \gamma E_{u' \sim \pi}[Q^{\pi}(s', u')]].$$

Политика строится так, чтобы действия, предпринимаемые согласно ей, максимизировали значение Q в каждом состоянии:

$$u(s) = argmax(Q(u, s)), s \in S$$

Но такой подход не применим, если число состояний и в особенности, возможных действий агента велико, поскольку вычисления в этом случае будут очень затратными.

В книге [39] также рассматривается возможность применения глубокой нейронной сети для получения параметрической аппроксимации функции полезности Q^* путём минимизации функции потери $L(\theta)$:

$$L(\theta) = E_{s,u,r,s'}[(Q^*(s,u,\theta) - r - \gamma \max_{u'} Q^{*'}(s',u'))^2].$$

Этот подход с использованием глубокой нейронной сети называется глубокое Q-обучение или DQN. Но всё же метод малоэффективен, если пространство состояний и действий велико.

1.2 Некоторые подходы к решению одноагентных RL задач

Задача 2 также может решаться с помощью методов градиента политики. Они оптимизируют политику агента, параметризованную θ^π, выполняя градиентное восхождение по оценке ожидаемого дисконтированного общего вознаграждения $J = E_\pi[R]$. Одной из простых форм градиента политики является [45], в которой градиент:

$$g = E_{s_{0:\infty},u_{0:\infty}}[R^* \nabla_{\theta^\pi} \log \pi(u_t|s_t)].$$

Однако здесь есть разные подходы к вычислению оценки функции R^*. Это может быть просто сумма всех вознаграждений после одной эмуляции игры(метод REINFORCE[45]). Но тогда мы не сможем понять, какое именно действие возможно ухудшило наше конечное значение суммы $R = \sum_{t=0}^{T} \gamma^t R_t$. Поскольку если после эмуляции значение R положительное, то и для всех действий, предпринимаемых в этой симуляции мы даём положительную оценку вклада в общий доход.

Для обхода этой проблемы используются подходы actor-critic, где актёр, то есть политика, обучается, следуя градиенту, который зависит от критика, оценивающего функцию значений. В частности R_t заменяется

любым выражением, эквивалентным $Q(s_t, u_t) - b(s_t)$, где $b(s_t)$ - это базисная линия, предназначенная для уменьшения дисперсии[46]. Общий выбор $b(s_t) = V(s_t)$ и в этом случае R_t заменяется на $A(s_t, u_t)$. Другой вариант заключается в замене R_t на так называемую временную разность(temporal difference - TD)[2]: $r_t + \gamma V(s_{t+1}) - V(s_t)$, которая является несмещённой оценкой $A(s_t, u_t)$. На практике градиент должен оцениваться по траекториям, отобранным из окружающей среды, а функции (действия-)значения должны оцениваться с помощью аппроксиматоров функций. Следовательно, смещение и дисперсия оценки градиента сильно зависят от точного выбора оценочной функции[22]. В алгоритмах actor-critic используются две нейросети - одна для исполнителя, то есть для выбора действия, следуя политике π, а вторая для критика, для оценки полезности совершаемых действий политикой π. Параметры нейросети исполнителя обновляются после выбора первого действия(u_t), получения награды(r_t) за него и нового состояния(s_{t+1}), с использованием оценки критиком полезности этого действия в прошлом состоянии(s_t). Благодаря обновлению своих параметров, он производит следующее действие u_{t+1}, учитывая новое состояние s_{t+1}. Только после этого критик обновляет свои параметры. Опишем эти нейросети:

1. Actor: $\pi(s, u, \theta)$, s - состояние, u - действие, θ - параметры нейросети исполнителя;

2. Critic: $Q^*(s, u, \omega)$, s - состояние, u - действие, ω - параметры нейросети критика.

Оптимизация параметров θ и ω производится отдельно:

- $\Delta\theta = \alpha \nabla_\theta (\log \pi_\theta(s, u)) Q^*(s, u)$, где $Q^*_\omega(s, u)$ - оценивается критиком;
- $\Delta\omega = \beta(r(s, u) + \gamma Q^*_\omega(s_{t+1}, u_{t+1}) - Q^*_\omega(s_t, u_t)) \nabla_\omega Q^*(s_t, u_t)$, где $r(s, u)$ - награда за совершенное критиком действие, $\nabla_\omega Q^*(s_t, u_t)$ - градиент функции значения-действия.

Замечание. Обновления параметров сети исполнителя и критика идёт с разными весовыми коэффициентами α и β, потому что для каждого

из них важна собственная скорость обучения. Это из-за того, что градиенты, оцениваемые ими имеют разные формы.

1.3 Основные виды нейросетей

В данном параграфе идёт описание только двух видов нейросетей, потому что именно они представляют особую сложность в понимании одновременно с высокой эффективностью использования в обучении с подкреплением.

RNN

Человек не начинает каждый момент свое мышление с нуля. В то время, как вы читаете какое-нибудь предложение, вы воспринимаете каждое слово, основываясь на понимании значения предыдущих слов. Вы не забываете все и не начинаете анализировать каждое слово в отдельности. В целом, все ваши мысли откладываются в памяти.

Традиционные нейронные сети не могут запоминать информацию, и это является их главным недостатком. Например, представьте, что вы хотите классифицировать события происходящее в каждом кадре фильма. Непонятно, как классическая нейронная сеть может использовать предыдущие свои выводы для дальнейших решений.

Рекуррентные сети направлены на исправление этого недостатка: они содержат циклы, которые позволяют сохранять информацию.

В рекуррентных нейронных сетях, разница между выходами для разных элементов последовательности исходит из разных скрытых состояний, которые зависят от текущего элемента во входной последовательности и значения скрытых состояний на последнем шаге времени:

$$o^t = f(h^t; \theta) \tag{4}$$

$$h^t = g(h^{t-1}, x^t; \theta) \tag{5}$$

где o^t - вектор выходного стоя RNN в момент времени t, x^t - вектор входного слоя в момент времени t и h^t вектор скрытого слоя в момент

времени t.

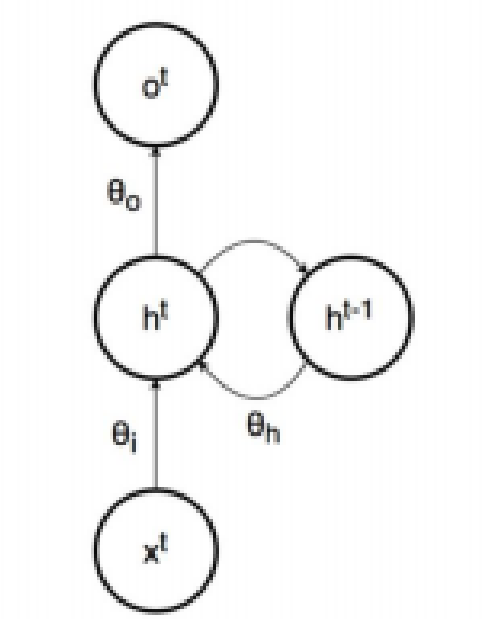

Рис. 3: Модель RNN. Значения $\theta_i, \theta_h, \theta_o$ представляют собой параметры, связанные с входом, предыдущим скрытым слоем и выходом, соответственно.

На рисунке 3 представлена схема, иллюстрирующая связь между входом, выходом и скрытым слоем в RNN.

Уравнение 4 показывает, что при заданных параметрах θ вектор выхода в момент времени t зависит только от состояния скрытого слоя в момент t, подобно нейронной сети прямого распространения. В свою очередь уравнение 5 говорит, что при одинаковых параметрах θ скрытый слой в момент времени t зависит от скрытого слоя в момент времени $t-1$ и от входа в момент времени t. Именно второе уравнение указывает, что RNN может помнить свое прошлое состояние h^{t-1} и влиять на текущее определение скрытого слоя h^t.

Из-за циклов рекуррентные нейронные сети становятся трудными в понимании. Однако, они имеют много общего с обыкновенными сетями. Рекуррентную сеть можно развернуть в последовательность одинаковых обыкновенных нейронных сетей, передающих информацию к последующим, например, как изображено на рисунке 4.

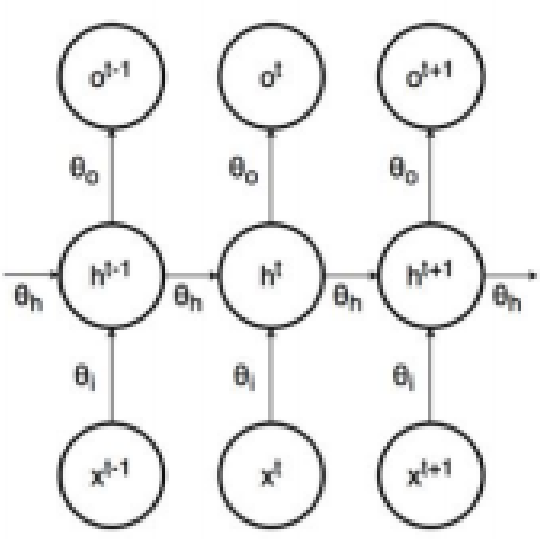

Рис. 4: Развёрнутая RNN в моменты времени $t-1, t, t+1$.

Эта цепочка показывает, что природа рекуррентных нейронных сетей тесно связана с последовательностями и списками. Они являются естественными архитектурами для использования таких данных.

И, естественно, они используются. За последние несколько лет был достигнут значительный успех в применении рекуррентных нейронных сетей для распознавания речи, моделирования языка, перевода, распознавания изображений и других интересных вещей[4].

Существенное влияние на успех оказало появление LSTM сетей — очень специфичного типа рекуррентных нейронных сетей, которые работают для большого количества задач значительно лучше, чем обыкновенные сети. Практически все выдающиеся результаты, достигнутые с помощью рекуррентных нейронных сетей основаны на них.

LSTM

Долгая краткосрочная память (Long short-term memory), обычно называемая LSTM-сетями — это особый вид рекуррентных нейронных сетей, способных к запоминанию долговременных зависимостей. Они были введены Сеппом Хохрайтером и Юргеном Шмидхубером в 1997 году и были использованы и развиты многими исследователями в своих работах. Эти сети работают в широком спектре задач и довольно часто используются.

LSTM сети были разработаны для решения проблемы долговременных зависимостей. Запоминание информации на продолжительный срок — это одна из основных особенностей этих сетей, не требующая продолжительного обучения.

Рассмотрим реализацию Грейвзом[3]. В отличие от рекуррентного блока, который просто вычисляет взвешенную сумму входного сигнала и применяет нелинейную функцию, каждый блок LSTM содержит три шлюза, а именно входной вентиль, забывающий вентиль и выходной вентиль для защиты и контроля состояния блока. Блок LSTM показан на рисунке 5, на котором представлены входные и выходные данные предыдущего блока.

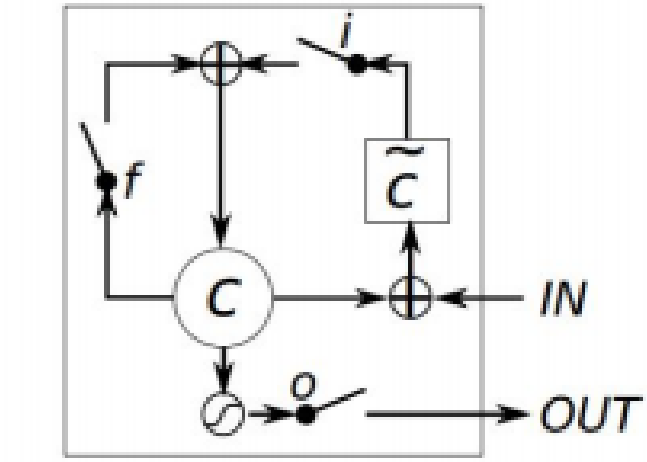

Рис. 5: Блок LSTM.

Каждый j-ый блок LSTM содержит память $\frac{j}{t}$ в момент времени t. Выход h_t^j или функция активации блока LSTM имеет вид:

$$h_t^j = o_t^j \tanh(c_t^j)$$

где o_t^j выходной вентиль, который регулирует количество содержимого в памяти. Выходной вентиль вычисляется:

$$o_t^j = \sigma(W_o x_t + U_o h_{t-1} + V_o c_t)^j$$

где σ - функция сигмоиды. V_o - диагональная матрица.

Ячейка памяти c_t^j обновляется, частично забывая текущее состояние памяти и добавляя новое содержимое $\tilde{c}_t^j$:

$$c_t^j = f_t^j c_{t-1}^j + i_t^j \tilde{c}_t^j$$

где новый контент:

$$\tilde{c}_t^j = \tanh(W_c x_t + U_c h_{t-1})^j$$

Степень забывания текущего состояния памяти моделируется с помощью вентиля забывания f_t^j , а степень добавления нового контента в ячейку памяти определяется с помощью элемента ввода i_t^j, соответственно имеют вид:

$$f_t^j = \sigma(W_f x_t + U_f h_{t-1} + V_f c_{t-1})^j$$

$$i_t^j = \sigma(W_i x_t + U_i h_{t-1} + V_i c_{t-1})^j$$

где V_f, V_i - диагональные матрицы.

В отличие от традиционного рекуррентного модуля, который перезаписывает свое содержимое на каждом временном шаге, модуль LSTM может решить, сохранить ли текущее состояние памяти через введенные вентили. Таким образом, если модуль LSTM обнаруживает важную зависимость из входной последовательности на начальной стадии, он может зафиксировать ее вне зависимости от длины последовательности.

Это лишь одна вариация LSTM-модели. Существует множество других, таких как глубокие вентильные LSTM-сети (Depth Gated LSTM) [20]. Кроме того предлагаются принципиально иные подходы, такие как часовые нейронные сети (Clockwork RNNs) предложенные в статье [17].

Какой из этих вариантов лучше? Имеются ли различия? В [21] делается большое сравнение популярных вариантов, и приходят к выводу о том, что они примерно одинаковы. В [33], протестировав более 10 тысяч архитектур рекуррентных нейронных сетей, говорят о том, что некоторые из них работали лучше, чем LSTM в специализированных задачах.

1.4 Метод COMAGP

Алгоритм COMAGP, описанный в статье [13], является главным объектом данной работы. Он позволяет эффективно решать задачу 3. Рассмотрим его.

В основе алгоритма COMA лежат 3 принципа:

1. Использование централизованного критика. Он используется только во время обучения. Во время исполнения действует только актёр.

2. Используется контрфактуальная базисная линия.

3. Используется представление критика, которое позволяет эффективно рассчитывать контрфактуальную базисную линию. Q-значения могут быть вычислены за 1 пакетный прямой проход.

Чуть позже мы поговорим про каждый из пунктов подробнее.

Стандартным решением было бы следование каждого участника градиенту, основанному на TD, оценённой централизованным критиком:

$$g = \nabla_{\theta^\pi} log\ \pi(u|\tau_t^a)(r + \gamma V(s_{t+1}) - V(s_t)).$$

Однако такой подход не решает ключевую проблему присвоения наград. Поскольку ошибка TD учитывает только глобальные вознаграждения, градиент, рассчитанный для каждого участника, не рассматривает то, как действия конкретно этого агента способствуют достижению этой глобальной награды. Градиент может становиться очень шумным, особенно, когда есть много агентов. Поэтому COMA использует контрфактуальную базисную линию. Идея основана на разностной награде, в которой каждый агент учится на основе заданной награды $D^a = r(s,u) - r(s,(u^{-a},c^a))$, которая сравнивает общую награду с вознаграждением, которое получается, когда действие агента a заменяется действием по умолчанию c^a. Любое действие агента a, которое улучшает D^a, также улучшает истинную глобальную награду $r(s,u)$, потому что $r(s,(u^{-a},c^a))$ не зависит от действий агента a.

Разностные награды являются мощной системой присвоения наград, но требуют дополнительных симуляций, которые, если симулятор занят обучением, предложено оценивать, используя приближение функций, а не симулятор. Но для этого всё равно требуется значение действия по умолчанию, задаваемое пользователем, выбрать которое бывает трудно. Основ-

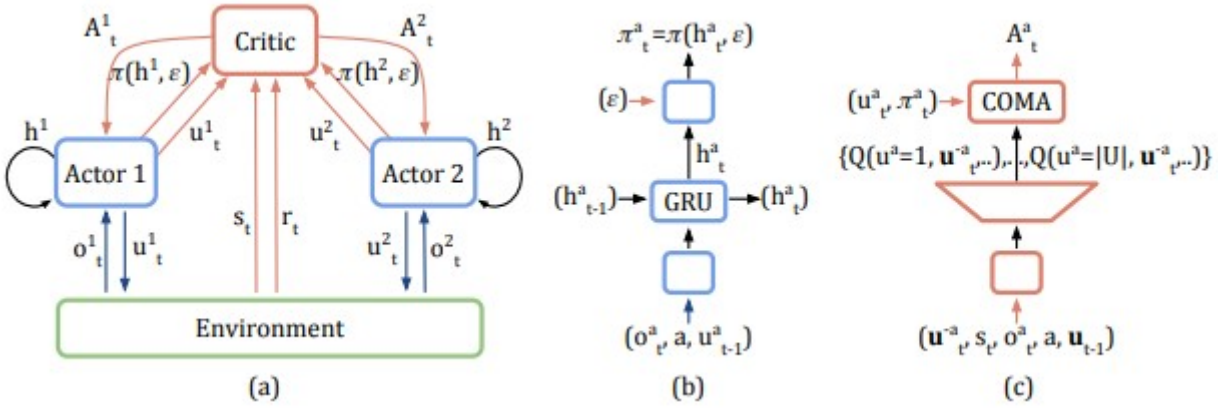

Рис. 6: В (a) поток информации между децентрализованными субъектами, окружающей средой и централизованным критиком в COMA; компонент критика и стрелки, идущие к нему и от него требуются только во время централизованного обучения. В (b) и (c) архитектуры актёра и критика.

ной конёк COMA заключается в том, что централизованный критик может быть использован для реализации разностных наград таким образом, чтобы избежать этих проблем. COMA обучает централизованного критика $Q(s, \mathbf{u})$, оценивающего Q-значения для совместного действия $\mathbf{u}$, обусловленного состоянием s. Для каждого агента a мы можем вычислить функцию преимущества, которая сравнивает Q-значение для текущего действия u^a с контрфактуальной базисной линией, которая изолирует u^a, оставляя действия других агентов u^{-a} фиксированными:

$$A^a(s, \mathbf{u}) = Q(s, \mathbf{u}) - \sum_{u'^a} \pi^a(u'^a|\tau^a) Q(s, (\mathbf{u}^{-a}, u'^a)).$$

Следовательно $A^a(s, \mathbf{u})$ вычисляет отдельную контрфактуальную базовую линию для каждого агента, которая использует централизованного критика для рассуждения о нереальных фактах, в которых изменяется только действие u, извлечённое непосредственно из опыта агентов, вместо того,

чтобы проводить дополнительные симуляции, или использовать стандартную модель вознаграждений, или использовать разработанное пользователем действие по умолчанию. В COMA ожидаемый вклад контрфактуальной базисной линии в градиент, как и в случае других базисных линий градиента политики, равен нулю. Таким образом, хотя базисная линия зависит от политики, математического ожидания от него нет. Поэтому здесь нет проблемы самосогласованности[13].

Обучение критика $f^c(\cdot, \theta^c)$ происходит на основе политики для оценки Q, используя вариант TD(λ)[38]. Он использует так называемую смесь n-шаговых наград $G_t^{(n)} = \sum_{l=1}^{n} \gamma^{l-1} r_{t+l} + \gamma^n f^c(\cdot_{t+n}, \theta^c)$. И параметры критика θ^c обновляются путём градиентного спуска по минибатчу, чтобы минимизировать следующие потери:

$$L_t(\theta^c) = (y^{(\lambda)} - f^c(\cdot_t, \theta^c))^2,$$

где $y^{(\lambda)} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$, а n-шаговые награды рассчитываются с загрузочными значениями, оценёнными целевой сетью(target network)[29] с параметром, периодически копируемым из θ^c.

Архитектура модели представлена на рисунке 6.

Псевдокод алгоритма представлен на рисунке 7.

Algorithm 1 Counterfactual Multi-Agent (COMA) Policy Gradients

Initialise θ_1^c, $\hat{\theta}_1^c$, θ^π
for each training episode e **do**
  Empty buffer
  for $e_c = 1$ **to** $\frac{\text{BatchSize}}{n}$ **do**
    $s_1 =$ initial state, $t = 0$, $h_0^a = \mathbf{0}$ for each agent a
    while $s_t \neq$ terminal **and** $t < T$ **do**
      $t = t + 1$
      for each agent a **do**
        $h_t^a = \text{Actor}\left(o_t^a, h_{t-1}^a, u_{t-1}^a, a, u; \theta_i\right)$
        Sample u_t^a from $\pi(h_t^a, \epsilon(e))$
      Get reward r_t and next state s_{t+1}
    Add episode to buffer
  Collate episodes in buffer into single batch
  for $t = 1$ **to** T **do** // from now processing all agents in parallel via single batch
    Batch unroll RNN using states, actions and rewards
    Calculate TD(λ) targets y_t^a using $\hat{\theta}_i^c$
  for $t = T$ **down to** 1 **do**
    $\Delta Q_t^a = y_t^a - Q\left(s_j^a, \mathbf{u}\right)$
    $\Delta\theta^c = \nabla_{\theta^c}(\Delta Q_t^a)^2$ // calculate critic gradient
    $\theta_{i+1}^c = \theta_i^c - \alpha\Delta\theta^c$ // update critic weights
    Every C steps reset $\hat{\theta}_i^c = \theta_i^c$
  for $t = T$ **down to** 1 **do**
    $A^a(s_t^a, \mathbf{u}) = Q(s_t^a, \mathbf{u}) - \sum_u Q(s_t^a, u, \mathbf{u}^{-a})\pi(u|h_t^a)$ // calculate COMA
    $\Delta\theta^\pi = \Delta\theta^\pi + \nabla_{\theta^\pi} \log \pi(u|h_t^a) A^a(s_t^a, \mathbf{u})$ // accumulate actor gradients
  $\theta_{i+1}^\pi = \theta_i^\pi + \alpha\Delta\theta^\pi$ // update actor weights

Рис. 7: Псевдокод алгоритма COMA.

Глава 2. Анализ экспериментов

2.1 Описание среды экспериментов

Для проведения мультиагентного обучения с подкреплением была выбрана модель игры в футбол. Эта модель обладает кооперативными свойствами, которые как раз подходят для алгоритма COMA. А в качестве среды для экспериментов использовалось окружение Unity3D[48] с использованием библиотеки для RL - ML-Agents[49]. В данном инструментарии присутствует модель футбольного поля для тестирования методов обучения с подкреплением. ML-Agents обеспечивает связь Unity3D со средой разработки языка Python, что является незаменимой вещью, если речь идёт о машинном обучении. Также пекрасным дополнением к вышеуказанным достоинствам данной связки Python + ML-Agents + Unity3D является визуализация процесса обучения и запуск обученных моделей, чтобы можно было посмотреть как обученные агенты играют в футбол.

В данной модели 6 агентов: 3 в одной команде и 3 в другой. Игроки делятся на голкипера - 1 и 2 полевых игроков, что видно на рисунке 8. Цель нападающих - забить как можно больше голов в ворота соперников, а цель голкипера - не дать залететь мячу в свои ворота. Множество действий у полевых игроков следующее: шаг вперёд, назад, влево, вправо и вращения влево и вправо. А у голкипера те же действия, но без вращений. Наблюдениями занимается не сам агент, а надстройка, называемая «Brain»в этом плагине. Наблюдения составляются по 14 лучам, расположенных на 180 градусов перед агентами, различающих 7 возможных типов объектов, а также расстояние до них.

За каждый забитый гол в чужие ворота полевым игрокам даётся 1 очко, а голкиперу 0.1. За пропуск мяча у них снимается 0.1 очка, а у голкипера 1. Если в эпизоде не удалось забить мяч, то у полевых игроков отнимается 0.001 очка, а вратарю даётся 0.001 очка.

Эпизодом называется время, которое равно 600 шагам игры. Эпизод заканчивается либо когда мяч попадает в ворота, либо когда проходит 600 шагов игры.

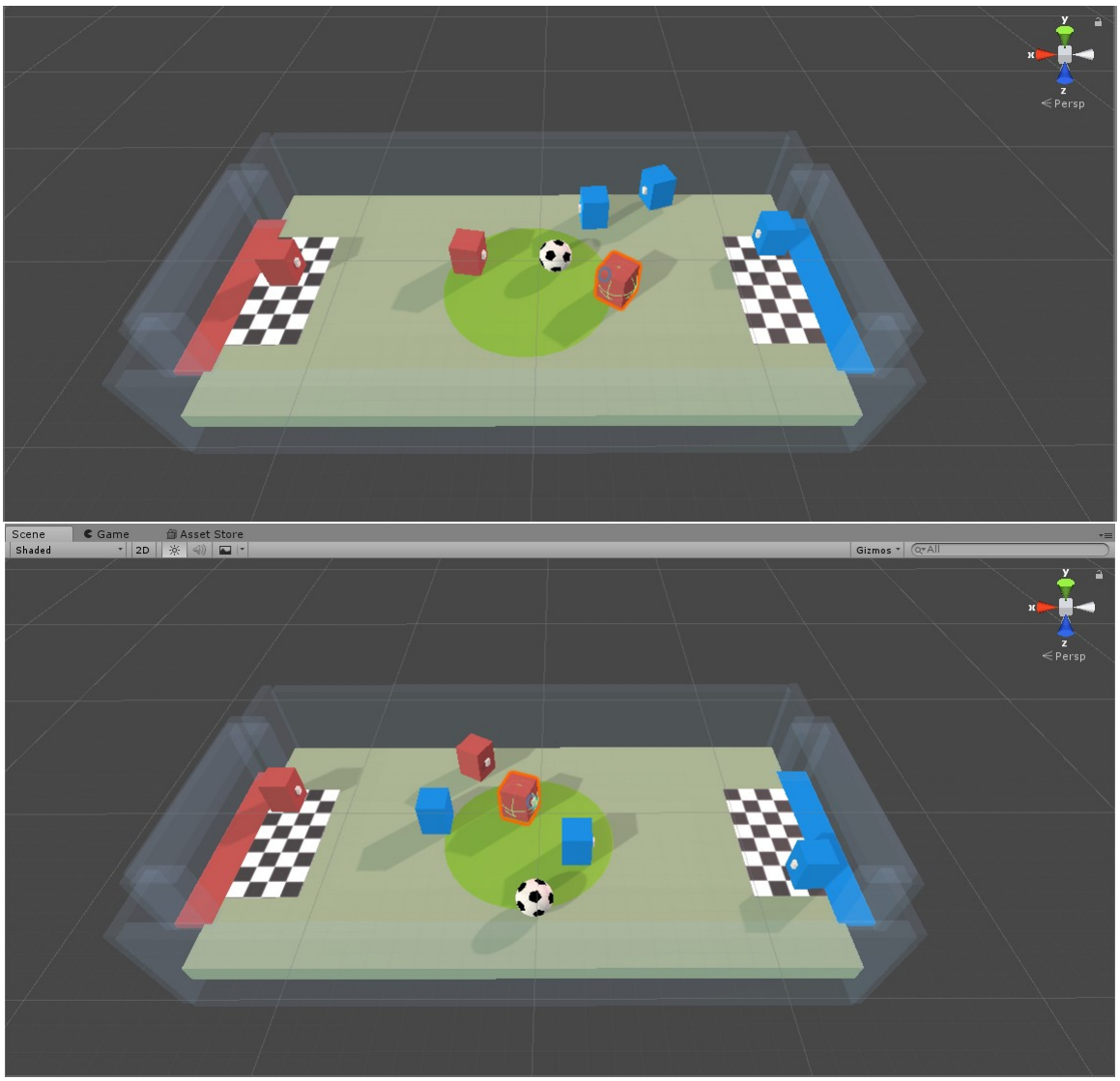

Рис. 8: Визуализация модели с использованием плагина ML-Agent в среде Unity3D.

2.2 Дополнительные настройки к алгоритму

Как говорилось при описании моделей обучения с подкреплением, в них есть определённые проблемы. Для их решения используются следующие подходы.

Эпсилон жадная стратегия

Следуя вышеописанному алгоритму COMA каждый агент будет выбирать действие в зависимости от своей стохастической политики(стратегии), которое стремится максимизировать величину $A^a(s_t^a, \mathbf{u})$. Но эта политика не самая эффективная. Когда нейронная сеть инициализируется случайным образом, она будет предрасположена к случайному выбору опреде-

лённых неоптимальных действий. Это может привести к тому, что агенты впадут в неоптимальные модели поведения без тщательного изучения игры и соответствия действие/награда. Таким образом агенты могут не найти лучшие стратегии для игры.

Здесь полезно ввести две концепции - разведка и эксплуатация. В начале задачи оптимизации лучше всего позволить исследовать пространство задачи в надежде найти хорошие локальные (или даже глобальные) минимумы. Однако, после того, как пространство задачи было должным образом изучено, для алгоритма оптимизации лучше всего сосредоточиться на использовании того, что он нашел, сходясь на лучших минимумах, чтобы прийти к хорошему решению.

Поэтому в обучении с подкреплением лучше всего допустить некоторую случайность в выборе действий в начале обучения. Эта случайность определяется параметром ϵ. По сути, агент генерирует случайное число между 0 и 1, и если оно меньше ϵ, то агент выбирает случайное действие. Если нет, действие выбирается на основе выходных данных нейронной сети. Переменная ϵ обычно задаётся вначале обучения близкой к 1, и постепенно уменьшается до нуля во время тренировки. Это позволяет вначале широко исследовать игру, но затем затухание значения эпсилона позволяет сети сосредоточиться на хорошем решении.

Батчинг в обучении с подкреплением(Batching)

Если нейронные сети актёра и критика обучаются на каждом этапе игры, то есть после того, как каждое действие выполнено и получено вознаграждение, существует серьезный риск перенастройки в сетях. Это связано с тем, что игровой процесс сильно коррелирован, то есть, если игра начинается с одного и того же места, а агенты выполняют одни и те же действия, то, скорее всего, результаты будут одинаковыми каждый раз (хотя и не совсем одинаковыми из-за случайности в некоторых играх). Поэтому после каждого действия рекомендуется добавлять все данные о состоянии, награде, действии и новом состоянии в какую-то память. Затем из этой памяти могут быть случайным образом отобраны партии, на которых проведутся шаги обучения, чтобы избежать риска перенастройки.

Поэтому сети можно по-прежнему обучать после каждого шага, ес-

ли необходимо, но она извлекает обучающие данные не из упорядоченных шагов агента в игре, а скорее из случайных партий из памяти предыдущих шагов и результатом, которые получали агенты.

2.3 Настройки обучения

Как уже говорилось в предыдущей главе, COMA использует 3 основных идеи:

1. Использование централизованного критика. Он используется только во время обучения. Во время исполнения действует только актёр.

2. Используется контрфактуальная базисная линия.

3. Используется представление критика, которое позволяет эффективно рассчитывать контрфактуальную базисную линию. Q-значения могут быть вычислены за 1 пакетный прямой проход.

Данные идеи сохранены в реализации обучения - для всей команды футболистов на этапе обучения используется централизованный критик, который оценивает функцию преимущества длявсех агентов. Также структура функции преимущества соответствует описанной в статье:

$$A^a(s, \mathbf{u}) = Q(s, \mathbf{u}) - \sum_{u'^a} \pi^a(u'^a|\tau^a) Q(s, (\mathbf{u}^{-a}, u'^a)).$$

В качестве апроксиматора данного значения использовалась нейронная сеть прямого распространения. Она состоит из двух полносвязных слоёв с функцией активации «ELU»[9]и одного скрытого слоя без функции активации, как и слой выхода.

Для действия агентов, то есть исполнителей в модели actor-critic(метод COMA) использовалась LSTM сеть[10]. Её преимущества показаны в предыдущей главе. Здесь агент имеет дело с последовательностью действий. Логично предположить, что, запоминая предыдущие действия, агенты будут более эффективно учиться следующим действиям.

Для обновления параметров сетей использован оптимизатор «Adam»[8], который является методом стохастической оптимизации. Он хорош тем, что во время обучения сам контролирует шаг обучения(learning rate).

Все вышеописанные структуры реализовывались с помощью языка программирования Python и библиотеки TensorFlow[14], которая является мощным средством реализации методов машинного обучения.

Поскольку мы используем аппроксимацию в вычислении целевых значений $TD(\lambda)$, то мы используем одни и те же веса для оценки целевых значений и Q - значений. Как следствие, между ними возникает большая корелляция. Это означает, что на каждом этапе обучения наши значения Q меняются вместе с целевыми значениями. Таким образом, мы приближаемся к нашей цели, но цель тоже движется. Это приводит к большим колебаниям в тренировках[42]. Чтобы этого избежать, мы оцениваем целевые значения $TD(\lambda)$ на фиксированных параметрах сети. Это отражено в алгоритме ниже.

Обучение идёт по следующему алгоритму:

1. Инициализируются нейронные сети - для исполнителей с параметрами θ^{π}, для критика с параметром θ_1^c и ещё одна для запоминания параметров критика $\hat{\theta}_1^c$
2. Для каждого тренировочного эпизода e повторяем:
3. Очищаем буфер
4. Для e_c от 1 до $BatchSize/n$ повторяем:
5. s_1 - начальное состояние, $t = 0$, $h_0^a = 0$ для каждого агента a
6. Пока $s_t \neq$ конечное состояние и $t < T$ повторяем:
7. $t = t + 1$
8. Для каждого агента a повторяем:
9. $h_t^a = \text{Actor}(o_t^a, h_{t-1}^a, u_{t-1}^a, a, u; \theta_i)$ (скрытые состояния LSTM сети получаем из нейросети актёров, подавая на вход значения, указанные в скобках)

10. Выбираем действие u_t^a на основе своей политики $\pi(h_t^a, \epsilon(e))$ и делаем это действие

11. Получаем награду r_t и следующее состояние s_{t+1}

12. Добавляем эпизод в буффер

13. Собираем эпизоды из буффера в один батч

14. Для t от 1 до T повторяем: (начиная здесь, все агенты обрабатывают эпизоды, записанные в батч)

15. Батч разворачивается LSTM-сетями с использованием состояний, действий и наград

16. Вычисляются целевые значения TD(λ) y_t^a используя фиксированные параметры $\hat{\theta}_i^c$

17. Для t от T до 1 повторяем:

18. $\Delta Q_t^a = y_t^a - Q(s_j^a, \mathbf{u})$

19. $\Delta\theta^c = \nabla_{\theta^c}(\Delta Q_t^a)^2$ (вычисляем градиент критика)

20. $\theta_{i+1}^c = \theta_i^c - \alpha\Delta\theta^c$ (обновляем веса критика)

21. Каждые C шагов обновляем $\hat{\theta}_i^c = \theta_i^c$

22. Для t от T до 1 повторяем:

23. $A^a(s_t^a, \mathbf{u}) = Q(s_t^a, \mathbf{u}) - \sum_u Q(s_t^a, u, \mathbf{u}^{-a})\pi(u|h_t^a)$ (рассчитываем COMA)

24. $\Delta\theta^\pi = \Delta^\pi + \nabla_{\theta^\pi} \log \pi(u|h_t^a) A^a(s_t^a, \mathbf{u})$ (накапливаем градиент исполнителей)

25. $\theta_{i+1}^\pi = \theta_i^\pi + \alpha\Delta\theta^\pi$ (обновляем веса исполнителей)

2.4 Результаты экспериментов

Для оценивания эффективности обучения, одна команда была заранее обучена встроенным в пакет ML-Agents методом обучения с подкреплением. После чего шло обучение второй команды. Гиперпараметры:

- learning_rate - начальная скорость обучения(начальный шаг) градиентного спуска;
- batch_size - количество опытов в каждой итерации градиентного спуска;
- ϵ - значение, используется для применения ϵ - жадной стратегии;
- γ - фактор дисконтирования;
- MaxSteps - количество шагов обучения.

Таким образом, подобрав гиперпараметры эмпирическим методом, были получены следующие результаты.

Рис. 9: Совокупная награда агентов, длина 1 эпизода, скорость обучения.

На рисунке 9 мы видим, что совокупная награда агентов растёт с увеличением количества шагов, а также длина эпизода уменьшается, это значит, что агенты учатся забиванию голов правильно. Но так как количество шагов обучения не достаточно большое, мы не можем с уверенностью сказать, что обучение прошло эффективно. Данные графики получены при обучении модели со следующими гиперпараметрами: learning_rate $= 3{\cdot}10^{-4}$, batch_size $= 1024$, $\epsilon = 0.2$, $\gamma = 0.99$, MaxSteps $= 310$k (Буква k - означает тысячу).

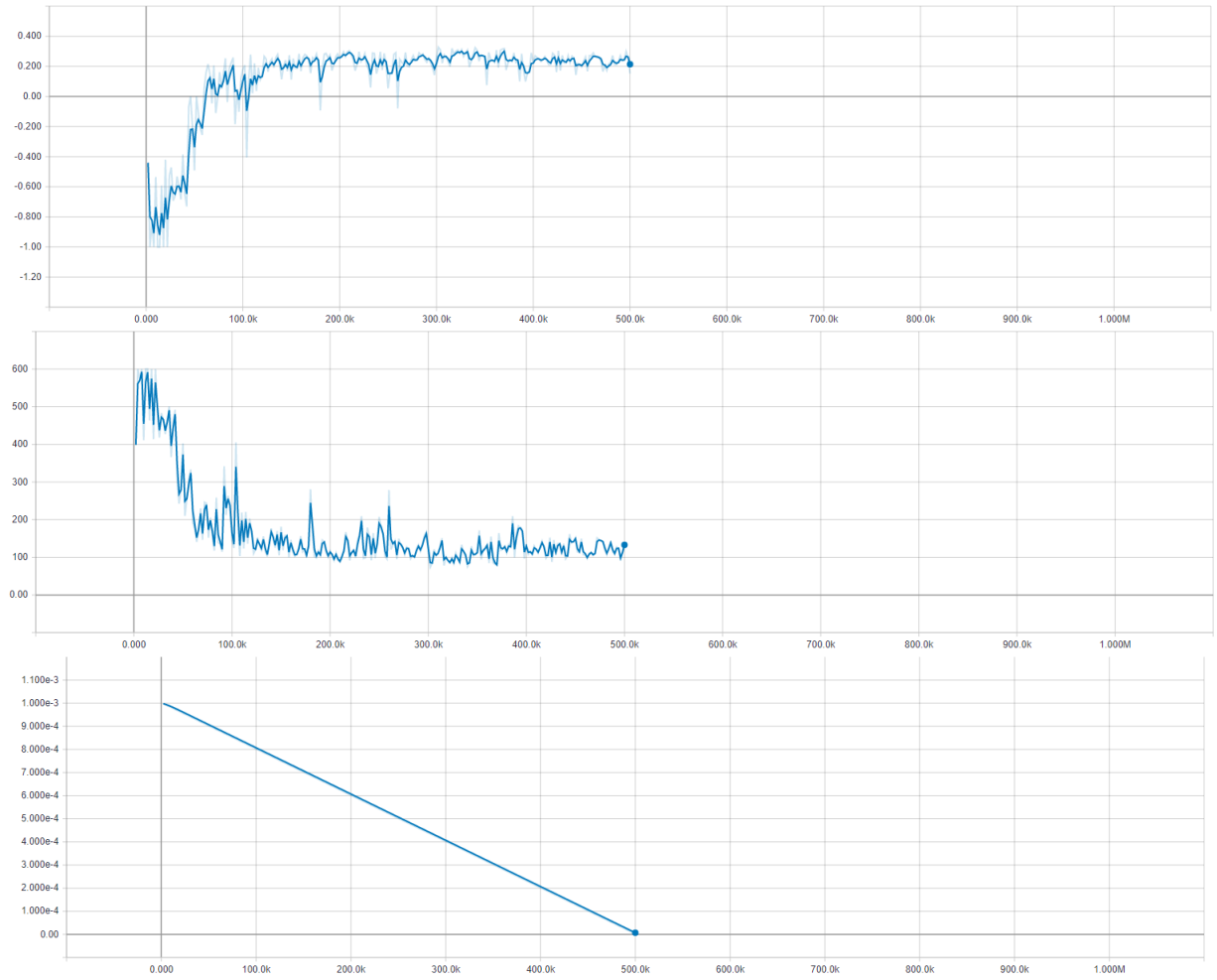

Рис. 10: Совокупная награда агентов, длина 1 эпизода, скорость обучения.

На рисунке 10 мы видим, что совокупная награда агентов растёт с увеличением количества шагов, а также длина эпизода уменьшается, это значит, что агенты учатся забиванию голов правильно. Также мы видим, что роста совокупной награды больше не происходит с увеличением шагов, линия совокупной награды колеблется около отметки в 0.2 единицы. Длина эпизода также перестаёт уменьшаться с течением времени и колеблется около 100 шагов. Из этого мы можем сделать вывод, что обучение прошло эффективно. Данные графики получены при обучении модели со следующими гиперпараметрами: learning_rate $= 3{\cdot}10^{-4}$, batch_size $= 1024$, $\epsilon = 0.2$, $\gamma = 0.99$, MaxSteps $= 500$k. Чтобы проверить, что этот результат не случайный, проведём обучение на таких же гиперпараметрах, за исключением количества шагов. Сделаем эксперимент ещё раз для 500k шагов и 1000k шагов.

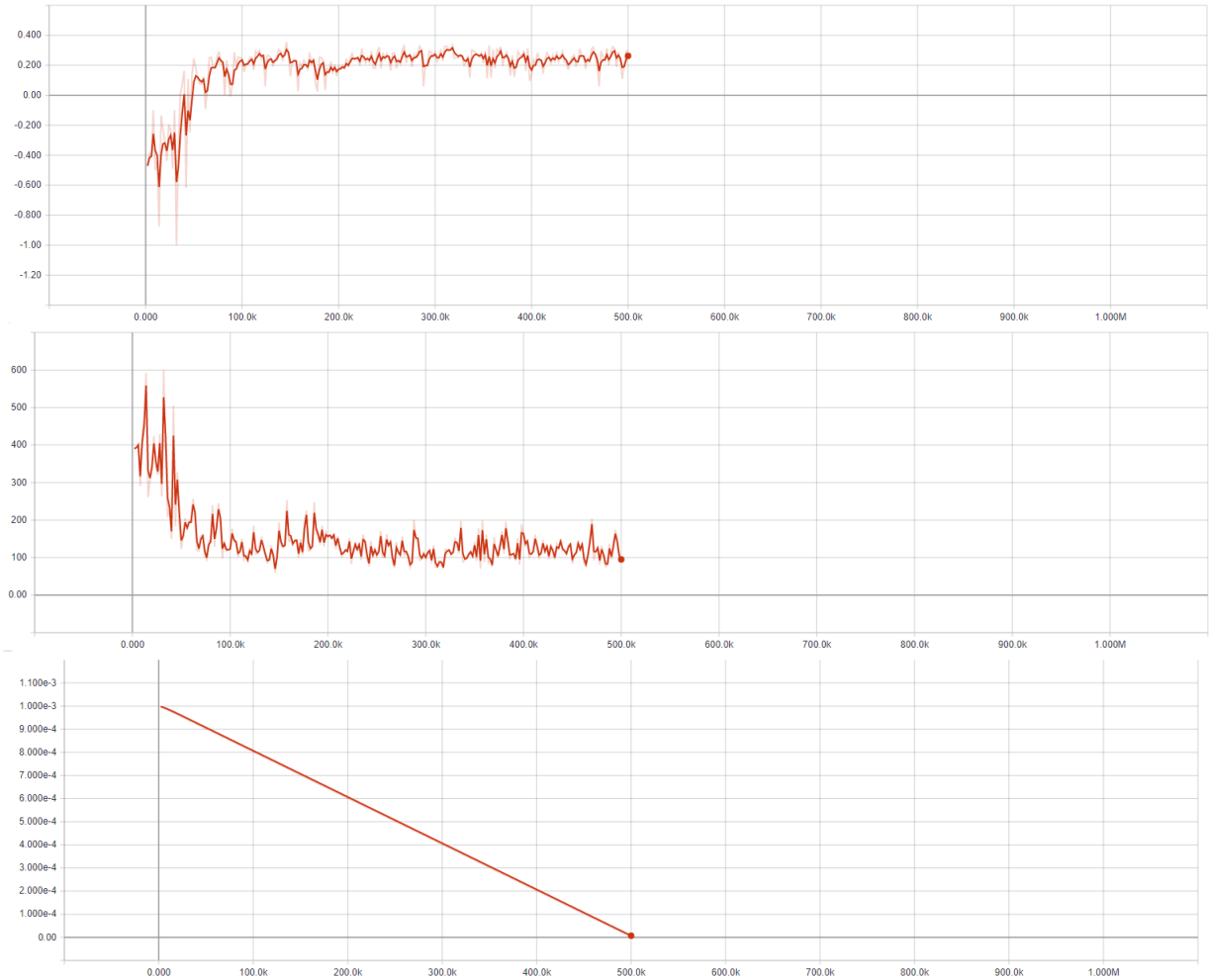

Рис. 11: Совокупная награда агентов, длина 1 эпизода, скорость обучения.

Как мы видим на рисунке 11, предыдущий результат повторился. Теперь увеличим количество шагов до 1000k.

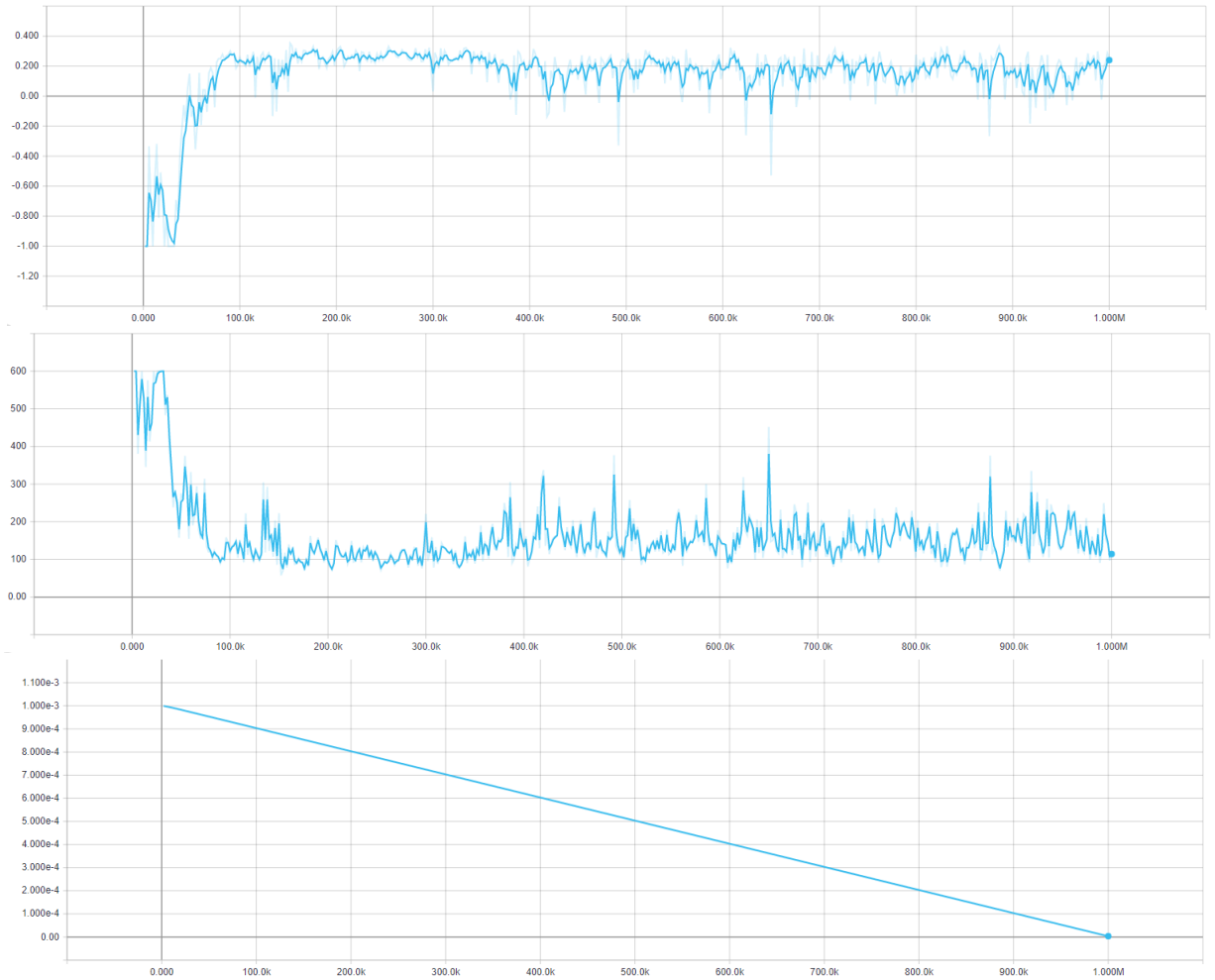

Рис. 12: Совокупная награда агентов, длина 1 эпизода, скорость обучения.

Как мы видим на рисунке 12, предыдущий результат также повторился и на 1000k шагов.

Рис. 13: Совокупная награда агентов, длина 1 эпизода, скорость обучения.

Рисунок 13 приведён для того, чтобы показать, что при неправильной настройке гиперпараметров очень легко не получить хороших результатов. Здесь есть интересный факт - длина 1 эпизода колеблется около максимальной границы в 600 шагов. Это значит, что голов забивается очень мало, ведь эпизод заканчивается при забитии гола, либо по достижении 600 шагов. Следовательно, встроенный в ML-Agents метод обучения с подкреплением не особо эффективен, раз не может забивать много голов плохо обученным агентам. Данные графики получены при обучении модели со следующими гиперпараметрами: learning_rate $= 3{\cdot}10^{-4}$, batch_size $= 2048$, $\epsilon = 0.2$, $\gamma = 0.95$, MaxSteps $= 1000$k.

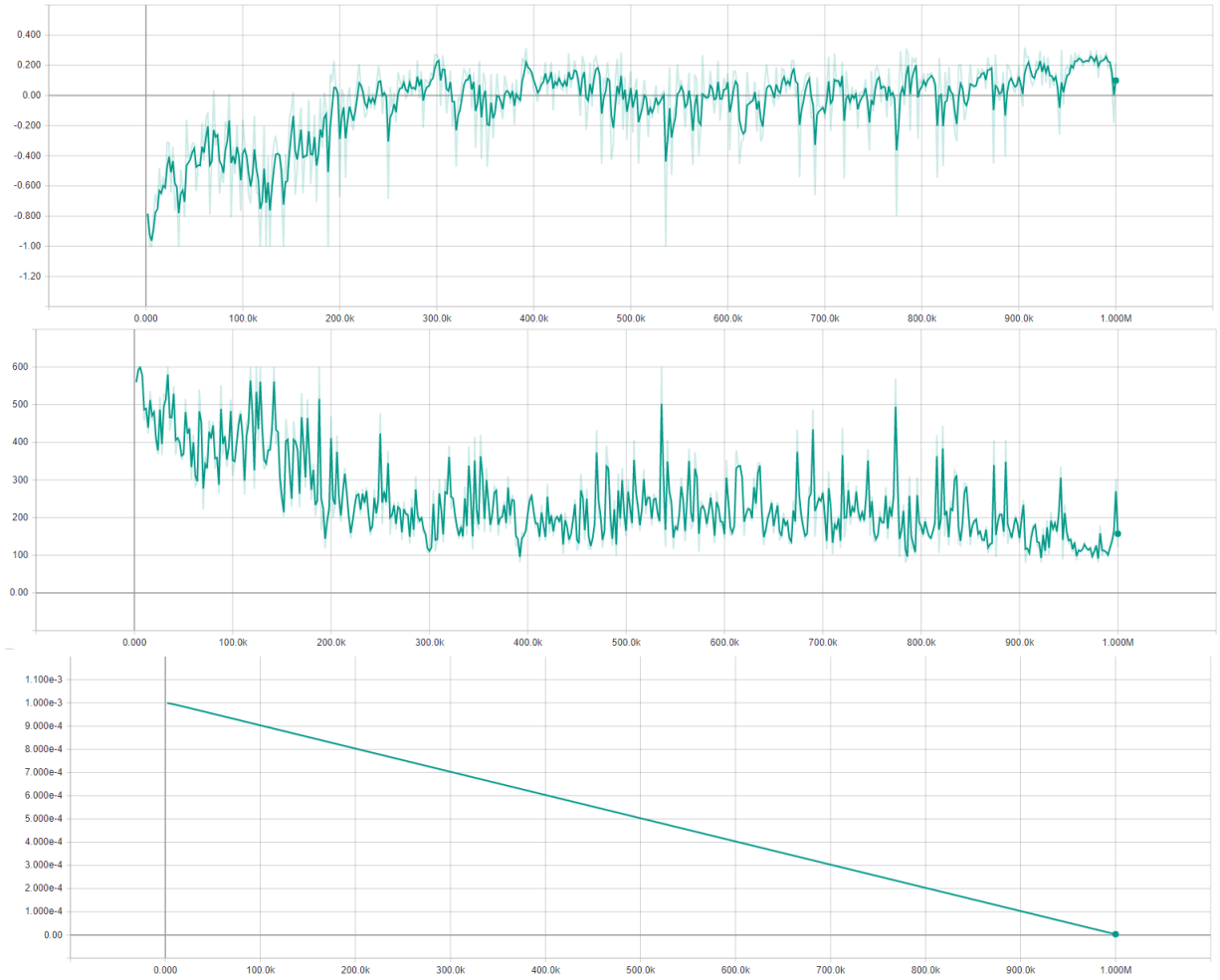

Рис. 14: Совокупная награда агентов, длина 1 эпизода, скорость обучения.

И если на предыдущем рисунке обучение не эффективно совсем, то на рисунке 14 результат не плохой, но всё же не такой устойчивый как на рисунке 12, например. Мы видим, что графики суммарной награды агентов и длины одного эпизода сильно зашумлены и колеблются в больших пределах, хоть и находятся около эффективных границ. Данные графики получены при обучении модели со следующими гиперпараметрами: learning_rate $= 3{\cdot}10^{-4}$, batch_size $= 1024$, $\epsilon = 0.2$, $\gamma = 0.95$, MaxSteps $=$ 1000k.

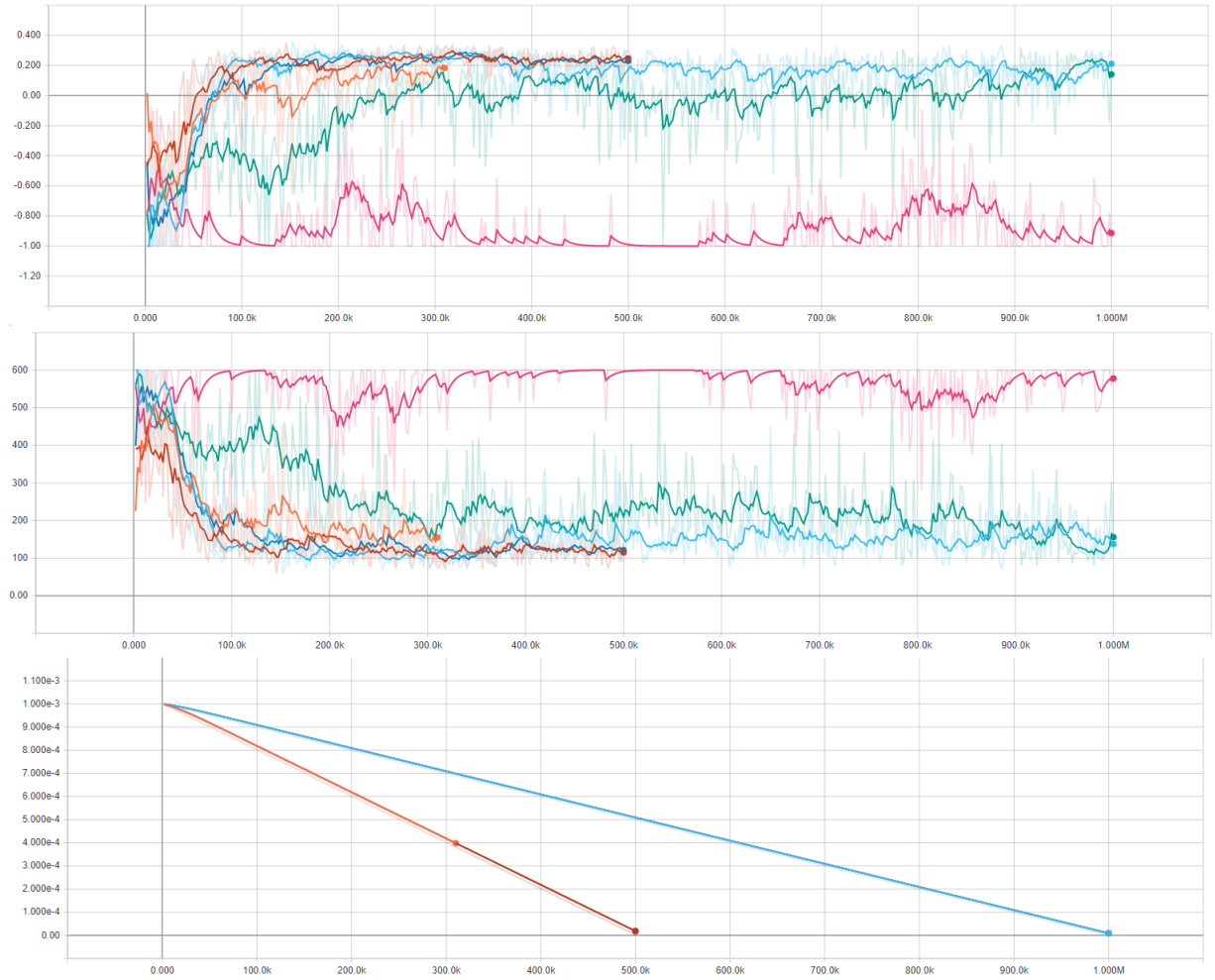

Рис. 15: Совокупная награда агентов, длина 1 эпизода, скорость обучения.

На рисунке 15 представлены все графики, которые были рассмотрены выше.

Заключение

По результатам данной работы можно сделать вывод, что цель, поставленная перед работой, достигнута. В ходе данной работы были достигнуты следующие результаты:

1. Был представлен обзор и подробный анализ существующих подходов к решению одноагентных и мультиагентных задач обучения с подкреплением;

2. Был тщательно рассмотрен один из современнных алгоритмов для решения мультиагентной задачи обучения с подкреплением, который был изменён для применения к мультиагентной модели игры в футбол;

3. Были описаны сложности, возникающие в обучении с подкреплением, а также пути их решения;

4. Были исследованы инструменты создания моделей для проведения экспериментов по обучению с подкреплением, а также модель футбола, встроенного в ML-Agents, изменена под необходимую задачу;

5. Была реализована программа с использованием языка Python и библиотеки машинного обучения Tensorflow, которая эффективно обучает мультиагентную модель футбола;

6. Были проведены вычислительные эксперименты для нахождения лучших гиперпараметров обучения, а также с помощью вычислительных экспериментов с данными гиперпараметрами показана эффективность алгоритма COMA для использования в данной модели. Предоставлены графики показателей - совокупной награды, длительности 1 эпизода, а также изменение learning_rate.

Список литературы

[1] Черепанов Е.А. «Разработка программы управления персонажем в многопользовательской онлайн игре». 2018. https://clck.ru/G5bU

[2] Adithya M. Devraj, Ioannis Kontoyiannis, Sean P. Meyn «Differential Temporal Difference Learning». arXiv:1812.11137. 2018.

[3] Alex Graves «Generating Sequences With Recurrent Neural Networks». arXiv:1308.0850. 2013.

[4] Andrej Karpathy «The Unreasonable Effectiveness of Recurrent Neural Networks ». 2015. http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[5] Busoniu, L.; Babuska, R.; and De Schutter, B. «A comprehensive survey of multiagent reinforcement learning. ». IEEE Transactions on Systems Man and Cybernetics Part C Applications and Reviews 38(2) 2008. pp.156.

[6] Chang, Y.-H.; Ho, T.; and Kaelbling, L. P «All learning is local: Multi-agent learning in global reward games». In NIPS, 2003. pp.807–814.

[7] Das, A., Kottur, S., Moura, J. M., Lee, S., and Batra, D. «Learning cooperative visual dialog agents with deep reinforcement learning.». arXiv preprint arXiv:1703.06585. 2017.

[8] Diederik P. Kingma, Jimmy Ba. «Adam: A Method for Stochastic Optimization». arXiv:1412.6980. v7. 2017.

[9] Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter. «Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)». arXiv:1511.07289. v5. 2016.

[10] Felix A. Gers, Jürgen Schmidhuber, Fred A. Cummins. «Learning to Forget: Continual Prediction with LSTM ». Technical Report IDSIA-01-99. 1999.

[11] Foerster, J., Assael, Y. M., de Freitas, N., and Whiteson, S. «Learning to communicate with deep multi-agent reinforcement learning. ». In Advances in Neural Information Processing Systems. 2016. pp.2137–2145.

[12] Foerster, J., Nardelli, N., Farquhar, G., Torr, P., Kohli, P., Whiteson, S., et al. «Stabilising experience replay for deep multi-agent reinforcement learning». In Proceedings of The 34th International Conference on Machine Learning. 2017.

[13] Foerster J., Farquhar G., Afouras T., Nardelli N., Whiteson S. «Counterfactual Multi-Agent Policy Gradients». arXiv:1705.08926. 2017.

[14] Google AI «TensorFlow». https://www.tensorflow.org/

[15] Gupta, J. K., Egorov, M., and Kochenderfer, M. «Cooperative multi-agent control using deep reinforcement learning. ». 2017.

[16] Himanshu Sahni «Reinforcement Learning never worked, and 'deep' only helped a bit». https://himanshusahni.github.io/2018/02/23/reinforcement-learning-never-worked.html

[17] Jan Koutník, Klaus Greff, Faustino Gomez, Jürgen Schmidhuber «A Clockwork RNN». arXiv:1402.3511. 2014.

[18] Jayanth Koushik «Understanding Convolutional Neural Networks». arXiv:1605.09081. 2016.

[19] Jorge, E., Kageback, M., and Gustavsson, E. «Learning to play guess who? and inventing a grounded language as a consequence». arXiv preprint arXiv:1611.03218. 2016.

[20] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, Chris Dyer «Depth-Gated LSTM». arXiv:1508.03790. 2015.

[21] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, Jürgen Schmidhuber «LSTM: A Search Space Odyssey». arXiv:1503.04069. 2017.

[22] Konda V. R., and Tsitsiklis J. N. «Actor-critic algorithms». In Advances in neural information processing systems. 2000. pp.1008-1014.

[23] Kraemer, L., and Banerjee, B. «Multi-agent reinforcement learning as a rehearsal for decentralized planning.». Neurocomputing. 2016. pp.82–94.

[24] Lazaridou, A., Peysakhovich, A., and Baroni, M. «Multi-agent cooperation and the emergence of (natural) language». arXiv preprint arXiv:1612.07182. 2016.

[25] Leibo, J. Z., Zambaldi, V., Lanctot, M., Marecki, J., and Graepel, T. «Multi-agent reinforcement learning in sequential social dilemmas». arXiv preprint arXiv:1702.03037. 2017.

[26] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. «Multi-agent actor-critic for mixed cooperative-competitive environments». arXiv preprint arXiv:1706.02275. 2017.

[27] Mateusz Kurek «Deep Reinforcement Learning in Keepaway Soccer». Poznań University of Technology Institute of Computing Science. 2015.

[28] Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., Riedmiller M. «Playing Atari with Deep Reinforcement Learning». arXiv:1312.5602. 2013.

[29] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al.«Humanlevel control through deep reinforcement learning». Nature. 2015. pp. 529–533.

[30] Mordatch, I., and Abbeel, P. «Emergence of grounded compositional language in multi-agent populations». arXiv preprint arXiv:1703.04908. 2017.

[31] Oliehoek, F. A., Spaan, M. T. J., and Vlassis, N. «Optimal and approximate Q-value functions for decentralized POMDPs». 2008. pp.289–353.

[32] Peng P., Yuan Q., Wen Y., Yang Y., Tang Z., Long H., and Wang J. «Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games». arXiv preprint arXiv:1703.10069. 2017.

[33] Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever «An Empirical Exploration of Recurrent Network Architectures». 2015. http://proceedings.mlr.press/v37/jozefowicz15.pdf

[34] Samuel A. L. «Some Studies in Machine Learning Using the Game of Checkers». IBM Journal of Research and Development. 1959. pp. 210-229.

[35] Shoham, Y., and Leyton-Brown, K. «Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations». New York: Cambridge University Press. 2009.

[36] Siqi Liu, Guy Lever, Josh Merel, Saran Tunyasuvunakool, Nicolas Heess, Thore Graepel «Emergent coordination through competition». Conference paper at ICLR. 2019.

[37] Sukhbaatar, S., Fergus, R., et al. «Learning multiagent communication with backpropagation». In Advances in Neural Information Processing Systems. 2016. pp.2244–2252.

[38] Sutton, R. S.«Learning to predict by the methods of temporal differences ». Machine learning. 1988.

[39] Sutton R.S., Barto A.G. «Reinforcement Learning». The MIT Press. Second Edition. 2018.

[40] Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., and Vicente, R. «Multiagent cooperation and competition with deep reinforcement learning». arXiv preprint arXiv:1511.08779. 2015.

[41] Tan, M. «Multi-agent reinforcement learning: Independent vs. cooperative agents». In Proceedings of the tenth international conference on machine learning. 1993. pp.330–337.

[42] Thomas Simonini. «Improvements in Deep Q Learning: Dueling Double DQN, Prioritized Experience Replay, and fixed Q-targets». https://clck.ru/G8oAQ

[43] Usunier, N., Synnaeve, G., Lin, Z., and Chintala, S. «Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks». arXiv preprint arXiv:1609.02993. 2016.

[44] Vihar Kurama «Reinforcement Learning with Python». https://towardsdatascience.com/reinforcement-learning-with-python-8ef0242a2fa2

[45] Williams R. J. «Simple statistical gradient-following algorithms for connectionist reinforcement learning». Machine learning. 1992. pp.229–256.

[46] Weaver L., and Tao N. «The optimal reward baseline for gradient-based reinforcement learning». In Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence. 2001. pp.538-545.

[47] Yang, E., and Gu, D. «Multiagent reinforcement learning for multi-robot systems: A survey». Technical report, tech. rep. 2004.

[48] «Unity». https://unity.com/ru

[49] «Unity Machine Learning Agents Toolkit». https://github.com/Unity-Technologies/ml-agents

[50] «Wikipedia» https://en.wikipedia.org/wiki/Markov_decision_process