ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

Национальный исследовательский университет "Высшая школа экономики"

Факультет экономических наук
Образовательная программа "Экономика"

КУРСОВАЯ РАБОТА

На тему "Применение обучения с подкреплением в экономическом моделировании"

Студент группы БЭК-162
Даниелян Всеволод Эдуардович

Научный руководитель:
Старший преподаватель Демешев Борис Борисович

Москва 2019

Содержание

1 Введение

Обучение с подкреплением представляет собой разновидность машинного обучения, сконцентрованную на взаимодействии агентов, максимизирующих свои совокупные вознаграждения, с окружением. Гибкость постановки проблемы позволяет с успехом применять алгоритмы обучения с подкреплением к самому широкому спектру задач — от игры в Старкрафт [1] до управления квадрокоптером [2].

В этой работе я рассматриваю возможность применения обучения с подкреплением в экономическом моделировании на примере задачи о динамической дуополии Курно. Цель работы - продемонстрировать, как достижение экономического равновесия в динамической дуополии Курно может быть объяснено через последовательное взаимодействие друг с другом и средой независимо обучающихся агентов.

В качестве алгоритма обучения агентов я использовал MADDPG, который является [3] одним из наиболее продвинутых на настоящий момент алгоритмов обучения с подкреплением для окружений с несколькими агентами.

2 Основная часть

2.1 Основные концепции

В обучении с подкреплением главными действующими лицами являются агенты и окружение. В каждый момент времени t агент получает какие-то наблюдения o_t о состоянии окружения s_t, совершает действие a_t, получает вознаграждение $r_t = R(s_t, a_t, s_{t+1})$ в зависимости от состояния и действия, и переходит в следующее состояние s_{t+1}. Для удобства состояние и действие текущего периода могут обозначаться как просто s и a, а будущего как s' и a'.

Траектория τ - это последовательность состояний и действий:

$$\tau = (s_0, a_0, s_1, a_1, \ldots). \tag{1}$$

Переход из состояния в состояние осуществляется в соответствии с законами, которым подчиняется окружение, и зависисит от последнего совершенного действия. Функция перехода может быть детерминистической:

$$s' = f(s, a), \tag{2}$$

или стохастической:

$$s' \sim P(\cdot|s, a). \tag{3}$$

Формализацией такого последовательного процесса принятия решений являются марковские процессы принятия решений (Markov Decision Processes — MDP).

Определение 2.1. MDP — это кортеж, $\langle S, A, R, P, \rho_0 \rangle$, где

- S - множество состояний,
- A - множество действий,
- $R : S \times A \times S \mapsto \mathbb{R}$, где $r_t = R(s_t, a_t, s_{t+1})$,
- $P : S \times A \mapsto \mathcal{P}(S)$, где $P(s_{t+1}|s_t, a_t)$ — функция, задающая вероятность перехода из s_t в s_{t+1}. При этом P подчиняется марковскому свойству, то есть $P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = P(s_{t+1}|s_t, a_t)$.

Почти все алгоритмы обучения с подкреплением принимают в качестве предпосылки марковское свойство. Интуиция тут такова — мы хотим быть уверены, что вся информация, необходимая для принятия решения, содержится в состоянии s текущего периода. С теоретической

точки зрения, марковское свойство используется в доказательствах сходимости алгоритмов там, где оно вообще может быть получено.

Агенты в рамках MDP совершают действия в соответсвии со своей стратегией. Если стратегия детерминистическая, то она обычно обозначается как μ, и представляет собой отображение из множества состояний в множество действий:

$$a_t = \mu(s_t) \tag{4}$$

Если стратегия стохастическая, то она обычно обозначается как π и представляет собой условное распределение над пространством действий:

$$a_t \sim \pi(\cdot|s_t) \tag{5}$$

Далее мы часто будем иметь дело с аппроксимацией стратегий, в первую очередь с использованием нейросетей. Мы параметризуем функцию стратегии каким-то набором параметров (в случае нейросети это будут веса нейросети), обозначаемых обычно θ или ϕ. Параметризованные таким образом стратегии соответственно обозначаются как:

$$a_t = \mu_\theta(s_t) \tag{6}$$

$$a_t \sim \pi_\theta(\cdot|s_t) \tag{7}$$

Нам часто будет важно знать ожидаемое вознаграждение, если мы будем следовать стратегии π начиная с текущего состояния и до конца траектории (если траектория конечна):

$$V^\pi(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau)|s_0 = s\right], \tag{8}$$

или начиная с текущего состояния s, с учетом того, что мы совершили в нем определенное действие a:

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau)|s_0 = s, a_0 = a\right]. \tag{9}$$

Здесь $R(\tau) = \sum_{t=0}^{\infty}\gamma^t r_t$ — дисконтированная сумма вознаграждений, полученных агентом после прохождения траектории τ.

Задача обучения с подкреплением - выучить стратегию, максимизирующую математическое ожидание будущих вознаграждений:

$$\pi^* = \arg\max_\pi J(\pi) = \arg\max_\pi \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau)\right] \tag{10}$$

2.2 Уравнения Беллмана и Q-learning

Q-функции текущего и будущего периода связаны следующим соотношением, известным как уравнение Беллмана:

$$Q^{\pi}(s,a) = \mathbb{E}\left[r(s,a) + \gamma \mathop{\mathbb{E}}_{a' \sim \pi}\left[Q(s',a')\right] | s,a\right] \tag{11}$$

Для оптимальной стратегии уравнения Беллмана выглядят следующим образом:

$$Q^{*}(s,a) = \mathbb{E}\left[r(s,a) + \gamma \max_{a'} Q^{*}(s',a') | s,a\right]. \tag{12}$$

На форме уравнений Беллмана и основана идея глубокого Q-обучения (Q-learning). Мы можем аппроксимировать Q-функцию нейросетью, и обучить ее с помощью минимизации следующей функции потерь:

$$L(\theta) = \mathop{\mathbb{E}}_{s,a,r,s'}\left[(Q^{*}_{\theta}(s,a) - y)^2\right] \tag{13}$$

$$y = r(s,a) + \gamma \max_{a'} Q^{*}(s',a'), \tag{14}$$

где θ - параметры аппроксимирующей функции (веса нейросети).

Действительно, если мы знаем, что для оптимальной статегии должно выполняться соотношение 12, то минимизация функции потерь 13 должна приблизить нашу аппроксимацию к оптимальной Q-функции. Для понимания функции потерь 13 может помочь и следующая интуиция: в y, в отличие от $Q(s,a)$ фигурирует актуальное вознаграждение r. Благодаря этому сам y ближе к реальной Q-функции, чем $Q(s,a)$, и минимизируя функцию потерь, мы приближаем таким образом нашу аппроксимацию к тому, что мы аппроксимируем. Соответственно y выступает в качестве цели, к которой мы хотим приблизиться.

Сама минимизация как правило выполняется с помощью стохастического градиентного спуска, или подобного ему алгоритма. Следует учесть, что при взятии градиента функции потерь, y обычно полагают константой, хотя $Q_{\theta}(s',a')$ зависит от параметров, по которым мы берем градиент. Такую технику называют полу-градиентом (semi-gradient) в главе 9 [4]. Во-первых это уменьшает вычислительную сложность. Во-вторых, можно вспомнить интуицию стоящую за функцией потерь. В случае, если бы мы не считали y за константу, при шаге градиентного спуска мы двигали бы не только нашу аппроксимацию в направлении, приближающем ее к цели, но и двигали бы саму цель y в направлении, приближающем ее к нашей аппроксимации. Что еще хуже, так как сдвиг

y происходил бы засчет сдвига $Q_\theta(s', a')$, получалось бы, что прошлое вознаграждение r влияло бы на будущие, представленные соответствующим значением Q-функции, что совершенно контр-интуитивно и не соответствует парадигме марковских процессов принятия решений.

Обновление весов Q_θ выглядит следующим образом:

$$\theta_{i+1} = \theta_i + \alpha(Q^*_\theta(s, a) - y)\nabla_\theta Q^*_\theta(s, a) \tag{15}$$

, где α — коэффициент скорости обучения.

Целевые нейросети. Даже несмотря на то, что мы считаем y за константу при взятии градиента функции потерь, обновление весов все равно сдвигает y, поскольку в него входит $Q^*_\theta(s', a')$, которая зависит от θ. Это делает процесс обучения неустойчивым, и нам хотелось бы этого избежать. С этой целью применяют целевые нейросети (target networks) [5]. Суть этого метода в том, что вместо одной нейросети, представляющей Q-функцию агента, мы используем две. Одну из них мы обновляем по правилу 15. Вторая же используется для вычисления y, и называется целевой нейросетью. Параметры θ' этой второй нейросети обновляются с помощью усреднения Поляка:

$$\theta' = p\theta' + (1 - p)\theta \tag{16}$$

Где p - гиперпараметр, принимающий значения в интервале $(0, 1)$. Обновление θ' происходит как правило с меньшей периодичностью, чем θ. Это помогает стабилизировать обучение, поскольку теперь y значительно менее подвержен колебаниям при обновлении весов основной нейросети.

Поскольку мы учим Q-функцию сразу для оптимальной стратегии, для ее обновления мы можем использовать сэмплы вида (s, a, r, s'), полученные при следовании агентом произвольной стратегии. На практике, однако, как правило используется выборка, полученная при следовании агентом ϵ-жадной стратегии:

$$a = \begin{cases} \arg\max\limits_a Q_\theta(s, a), & \text{с вероятностью } 1 - \epsilon. \\ \text{случайный } a, & \text{иначе.} \end{cases} \tag{17}$$

К сожалению, необходимость поиска действия, максимизирующего Q-функцию в данном состоянии, ограничивает применение Q-обучения окружениями с дискретным пространством действий.

2.3 Градиент стратегии и DDPG

Основная идея, стоящая за методами, использующими градиент детерминистической стратегии, следующая — вместо того, чтобы выводить оптимальную стратегию из знания оптимальной Q-функции, мы будем искать ее в явном виде, то есть в виде функции 4. Это позволит нам эффективно искать оптимальную стратегию в том числе и в непрерывных пространствах действий, что было проблемой для Q-обучения. Теперь, вместо того, чтобы искать Q-функцию для оптимальной стратегии, мы будем двигать стратегию в направлении, максимизирующем Q-функцию, как было предложено в работе [6].

Вспомним, что наша задача максимизировать матожидание будущих вознаграждений:

$$J(\mu) = \mathop{\mathbb{E}}_{\tau\sim\mu}[R(\tau)] \tag{18}$$

Которое в свою очередь может быть представлено через Q-функцию:

$$J(\mu) = \mathop{\mathbb{E}}_{s\sim\rho^{\mu}}[Q(s,a)]\,, \tag{19}$$

где $s \sim \rho^{\mu}$ обозначает, что состояния берутся из распределения соответствующего стратегии μ. И поскольку действие a представляет собой функцию от состояния s, $\mu(s)$, мы можем взять градиент относительно параметров θ:

$$\nabla_{\theta}J(\mu) = \nabla_{\theta}\mathop{\mathbb{E}}_{s\sim\rho^{\mu}}[Q(s,a)] = \mathop{\mathbb{E}}_{s\sim\rho^{\mu}}[\nabla_{a}Q(s,a)\nabla_{\theta}\mu(s)] \tag{20}$$

Как показано в статье [6], мы можем брать матожидание по состояниям, полученным из произвольной стратегии β:

$$\nabla_{\theta}J(\mu) = \mathop{\mathbb{E}}_{s\sim\rho^{\beta}}[\nabla_{a}Q(s,a)\nabla_{\theta}\mu(s)] \tag{21}$$

Это дает нам возможность использовать буфер воспроизведения. То есть мы можем сохранять пройденные агентом траектории в буфер, а потом использовать случайную выборку оттуда для обновления параметров. Использование буфера разрешает проблему скоррелированности сэмплов. Действительно, если бы мы обновляли параметры, используя сэмплы полученные в близкие периоды времени, то значительная часть информации была бы потеряна из-за их корреляции. Использование буфера помогает решить эту проблему и повышает эффективность алгоритма относительно размера требуемой выборки [7].

Для оценки градиента стратегии нам необходимо знать Q-функцию или ее аппроксимацию. Как получить аппроксимацию мы уже знаем - минимизацией функции потерь 13.

Соответственно алгоритм представляет собой взаимодействие двух частей - критика $Q(s, a)$ и актора $\mu(s)$. Мы получаем какую-то траекторию τ с помощью актора μ_θ, сохраняем ее в буфер, получаем из буфера случайную выборку участков траектории вида (s, a, r, s') обновляем критика Q_ϕ и, используя критика, обновляем актора μ_θ с помощью градиентного вохождения. Данный алгоритм известен в литературе под названием глубокого градиента детерменистической стратегии (Deep Deterministic Policy Gradient - DDPG).

2.4 MADDPG

Рассмотрим случай одновременного обучения нескольких агентов. Мы могли бы использовать наивный подход, при котором каждый агент независимо от других взаимодействует с окружением, используя какой-нибудь из алгоритмов для случая одного агента. Недостаток такого подхода в том, что окружение становится нестационарным [8]. Под нестационарностью подразумевается изменение вероятностей перехода i-го агента $P(s'|s, a_i)$ с изменением стратегий других агентов. Действительно, будущее состояние s' теперь зависит не только от действия i-го агента a_i, но и от действий других агентов, которые в свою очередь зависят от их стратегий. И когда в ходе обучения мы обновляем стратегии агентов, мы тем самым меняем и вероятности перехода, причем с точки зрения i-го агента, изменение это зависит от предыдущих действий и состояний в которых оказывался агент. А это значит, что нарушается марковское свойство 2.1, выполнение которого принимается в качестве предпосылки для большинства алгоритмов обучения с подкреплением.

Один из способов побороть эту проблему — применить концепцию централизованного обучения с децентрализованным исполнением. В статье [9] авторы предложили расширение алгоритма DDPG на случай взаимодействия нескольких агентов. Основная идея следующая — во время обучения использовать информацию о действиях других агентов, а во время исполнения — только информацию о состоянии. DDPG состоит из двух составляющих, актора и критика, причем во время обучения используются оба, а во время исполнения — только актор. Поэтому мы можем обучать критика, используя информацию о действиях других агентов, обучать актора, используя обученного таким образом критика, и при этом после завершения обучения актору будет достаточно только знания состояния. При этом стационарность окружения сохраняется,

поскольку

$$P(s'|s, a_1, \ldots, a_N, \pi_1, \ldots, \pi_N) = P(s'|s, a_1, \ldots, a_N) = P(s'|s, a_1, \ldots, a_N, \pi'_1, \ldots, \pi'_N), \tag{22}$$

даже в том случае, если $\pi_i \neq \pi'_i$.

Пусть у нас есть N агентов с детерминистическими стратегиями, параметризованными вектором $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_N\}$, и наблюдениями $\mathbf{x} = \{o_1, \ldots, o_N\}$. Тогда градиент стратегии i-го агента будет выглядеть следующим образом:

$$\nabla_{\theta_i} J(\mu_i) = \mathop{\mathbb{E}}_{\mathbf{x}, a \sim \mathcal{D}} \left[\nabla_{\theta_i} \mu_i(o_i) \nabla_{a_i} Q_i(\mathbf{x}, a_1, \ldots, a_N)|_{a_k = \mu_k(o_k)} \right] \tag{23}$$

Где $Q_i^{\mu_i}(\mathbf{x}, a_1, \ldots, a_N)$ — централизованный критик, который оценивает вектор наблюдений $\mathbf{x}$ и действия всех агентов, и оценивает Q-функцию i-го агента. Q-функции каждого агента обучаются отдельно от других, что позволяет агентам иметь различные системы вознаграждений (в том числе и противоположные друг другу, в случае их конкуренции). $\mathcal{D}$ — буфер воспроизведения, куда сохраняются кортежи вида
$(\mathbf{x}, \mathbf{x}', a_1, \ldots, a_N, r_1, \ldots, r_N)$.

Централизованная Q-функция обучается с помощью минимизации следующей функции потерь:

$$L(\theta_i) = \mathop{\mathbb{E}}_{\mathbf{x}, a, r, \mathbf{x}'} \left[(Q_i^{\mu_i}(\mathbf{x}, a_1, \ldots, a_N) - y)^2 \right], \quad y = r_i + \gamma Q_i^{\mu'_i}(\mathbf{x}', a'_1, \ldots, a'_N)|_{a'_j = \mu'_j(o_i)}, \tag{24}$$

где $\mu_{\theta'_1}, \ldots, \mu_{\theta'_1}$ — множество целевых стратегий с отложенными параметрами θ'_i, по аналогии с тем, как мы это делали в Q-обучении 2.2.

Эксплорация. Поскольку необходимо, чтобы во время обучения агенты получали новую информацию, их действия на этот период полагают случайной величиной. В случае непрерывного пространства, действия могут быть получены из нормального распределения с матожиданием $\mu(s)$, и дисперсией либо выбираемой как гиперпараметр, либо выступающей в качестве еще одного параметра, который учит нейросеть аппроксимирующая μ. Если же пространство действий дискретно, можно использовать репараметризацию с помощью распределения Гумбеля (Gumbel-Softmax trick). Идея данного трюка такова — нам нужно получать действия из какого-то дискретного распределения, и при этом сохранить возможность взять градиент $\nabla_\theta \mu_\theta(s)$. Использование распределения Гумбеля с функцией софтмакса поверх него позволяет получить близкую к требуемому распределению аппроксимацию, сохраняя

при этом возможность взять градиент получаемых выборочных значений относительно весов нейросети [10].

2.5 Эксперименты

Я применил MADDPG для анализа динамической олигополии Курно. В этой модели два агента конкурируют по выпуску, выбирая его одновременно в каждом периоде. Выпуск дискретизирован и выбирается из конечного множества возможных выпусков $\{0, 10, \ldots, 1000\}$. Цена в каждом периоде определяется функцией спроса:

$$P_t(q_t^1, q_t^2) = 500 - \frac{q_t^1 + q_t^2}{2}, \tag{25}$$

где q_t^1, q_t^2 - выпуски соотвествующих агентов в период t. Агенты воспринимают цену, как непрерывную величину, и поэтому в случае необходимости функцию спроса можно сделать стохастической или подчиняющейся определенной динамике. В каждом периоде агенты получают прибыль:

$$r_t^i(q_t^i) = P_t q_t^i - 50 q_t^i \tag{26}$$

В начале каждого периода агенты наблюдают цену и прибыль за предыдущий период. Цель каждого агента - максимизировать свою совокупную дисконтированную прибыль. Промежуток времени не был ограничен сверху, но для удобства обучения в действительности использовались эпизоды длиной в 100 периодов. Агентам во время обучения были известны выпуски других агентов в прошлых периодах и эпизодах.

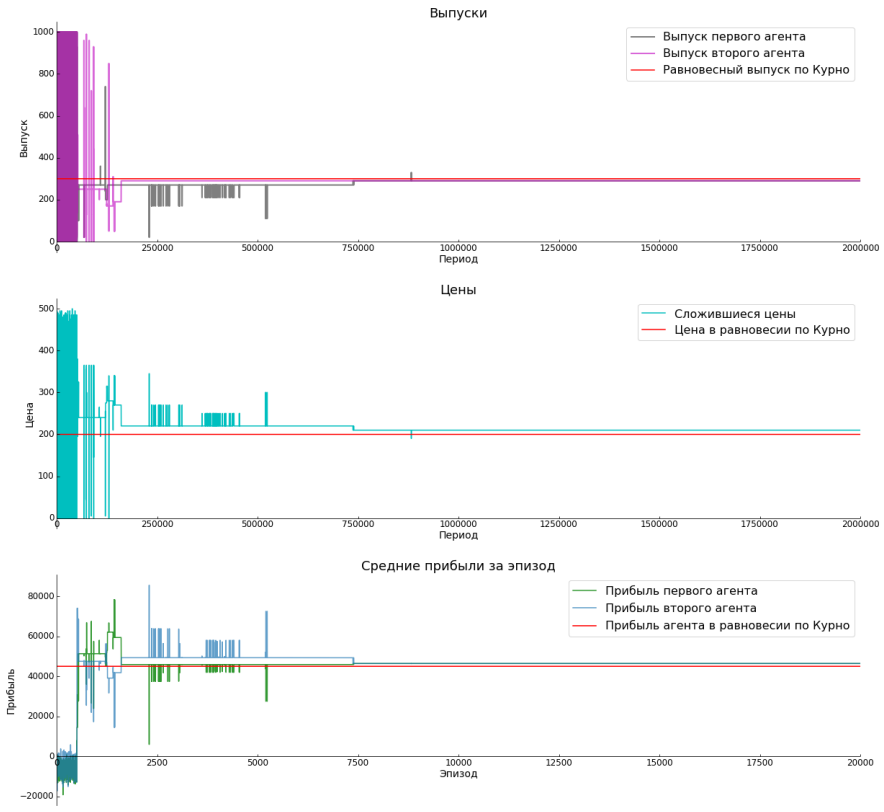

Рис. 1: $\gamma = 0.3$

На рисунке 1 изображены результаты экспериментов при $\gamma = 0.3$. Хаотичный выбор действий вначале объясняется тем, что агенты начинают обучаться только когда сохраненных траекторий накопится достаточно много. До этого момента их действия выбираются случайно. В отличие от реальных экономических агентов, у них нет априорных знаний о модели, и этот начальный период призван это компенсировать.

Во всех случаях (больше изображений траекторий обучения можно найти в приложении 4) агенты сходились к определенному равновесию. Кроме того, в этом равновесии по крайней мере один агент всегда получал прибыль больше, чем в случае статического равновесия Курно. Тем не менее, даже в случае нулевого дисконт-фактора статическое равнове-

сие Курно не было достигнуто, что может быть объяснено погрешностью аппроксимации и особенностями обучения. Агенты приобретают слишком большую уверенность в своих действиях до того, как статическое равновесие достигнуто, а поколебать эту уверенность могла бы только новая информация, которую в равновесном состоянии брать неоткуда.

Таблица 1: Характеристики статического равновесия Курно

γ	Цена	Прибыль 1-го агента	Прибыль 2-го агента
0	200	45000	45000

Таблица 2: Характеристики равновесия при различных γ

γ	Цена	Прибыль 1-го агента	Прибыль 2-го агента
0	245	42900	56550
0.3	210	46400	46400
0.6	210	44800	48000
0.9	290	50400	50400

С точки зрения человека вид траекторий выпуска не отвечает ожиданиям от действий разумных экономических агентов. Это определяется особенностями алгоритма обучения. Во-первых, агенты не учат модель, они учат только действия, которые принесут максимальную полезность в той или иной ситуации. То есть поведение агентов в MADDPG рефлекторное. В то время как реальные экономические агенты могут эксплицитно использовать свое представление о функции спроса и стратегиях других агентов при принятии решения. Во-вторых, агенты в MADDPG учатся исключительно методом проб и ошибок, что связанно с первым пунктом. Они не используют свое знание о модели, чтобы предсказать к каким последствиям может привести то или иное действие, вместо этого они вынуждены его пробовать. Обе эти проблемы можно было бы решить используя обучение с планированием, то есть заставить агентов учить и эксплицитно использовать информацию о модели и стратегиях других агентов при принятии решений.

3 Заключение

В данной работе я применил алгоритм MADDPG для обучения агентов в модели динамической олигополии Курно. Обученные агенты сходятся к определенному равновесию. Однако даже при выполнении предпосылок для сходимости к статическому равновесию Курно, оно не достигается. Кроме того, действия агентов во время обучения не похожи на действия рациональных экономических агентов. Для решения последней проблемы в будущих работах по теме, я предлагаю использовать обучение с планированием, когда агенты учат модель и стратегии других агентов, и используют это знание при принятии решений.

Список литературы

[1] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M. Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Yuhuai Wu, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[2] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[3] Ang Li, Michael Kuchnik, Yixin Luo, and Rohan Sawhney. Deep reinforcement learning in continuous multi agent environments.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[6] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[7] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[8] Guillaume J Laurent, Laëtitia Matignon, Le Fort-Piat, et al. The world of independent learners is not markovian. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 15(1):55–64, 2011.

[9] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2017.

[10] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

4 Приложение

Algorithm 1 MADDPG [9]

for эпизод = 1 до M **do**
  Инициализировать случайный процесс $\mathcal{N}$ для эксплорации
  Получить начальное состояние $\mathbf{x}$
  for $t = 1$ до максимальной длины эпизода **do**
    для каждого агента i выбрать действие $a_i = \mu_{\theta_i}(o_i) + \mathcal{N}_t$ с учетом текущей стратегии
    выполнить действия $a = (a_1, \ldots, a_N)$, получить значения вознаграждения r и состояния $\mathbf{x}'$
    сохранить $(\mathbf{x}, a, r, \mathbf{x}')$ в буфер $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for агента $i = 1$ до N **do**
      Получить случайную выборку размера S кортежей вида $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ из $\mathcal{D}$
      $y^j \leftarrow r_i^j + \gamma Q_i^{\mu'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)|_{a'_k = \mu'_k(o_j^k)}$
      Обновить критика, минимизируя фукнцию потерь $L(\theta_i) = \frac{1}{S}\sum_j (y^j - Q_i^{\mu}(\mathbf{x}^j, a_1^j, \ldots, a_N^j))^2$
      Обновить актора, используя выборочный градиент стратегии:

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

    end for
    Обновить параметры целевой нейросети для каждого агента i:

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

  end for
end for

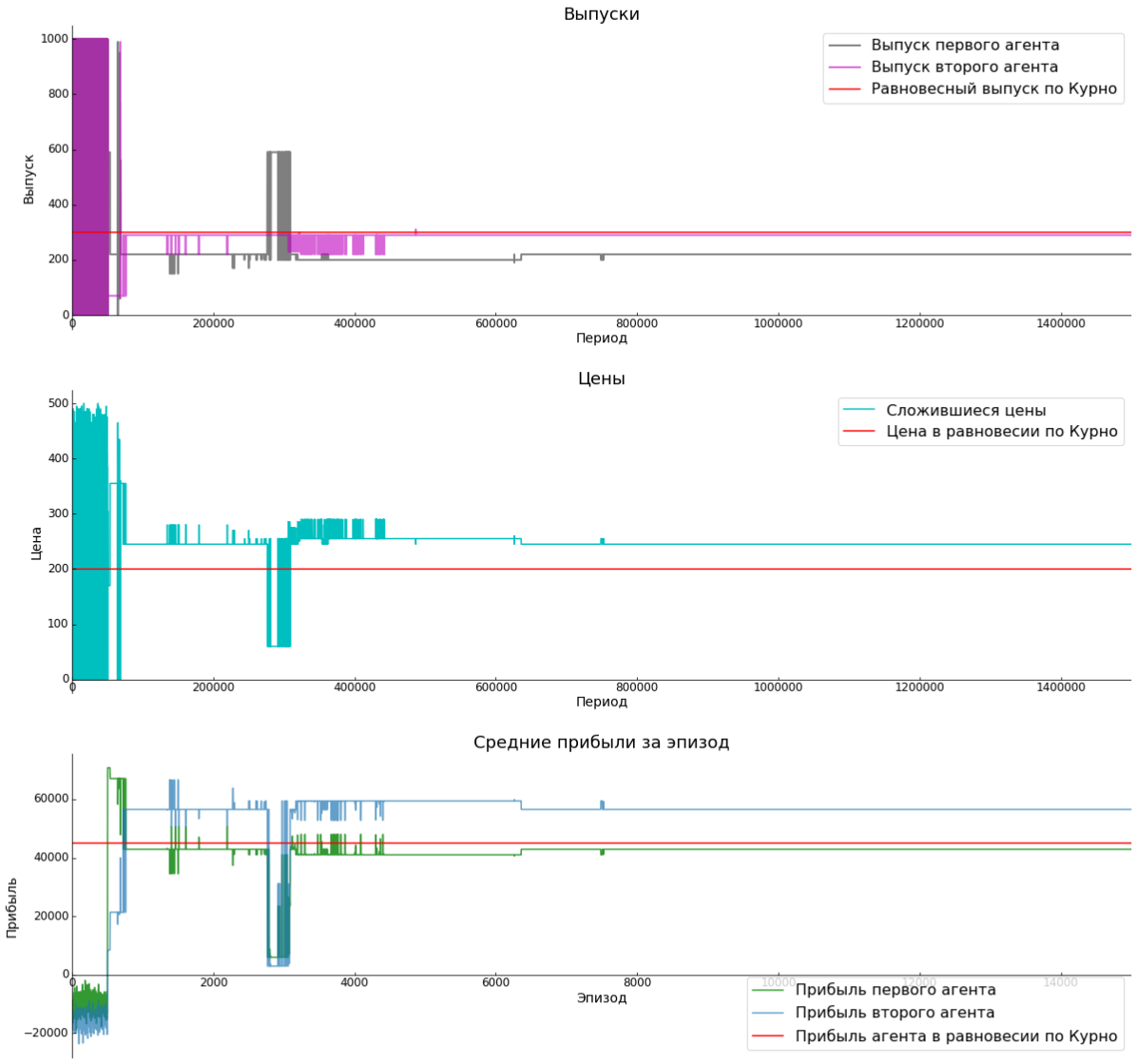
Выпуски
Выпуск первого агента
Выпуск второго агента
Равновесный выпуск по Курно
Выпуск
Период
Цены
Сложившиеся цены
Цена в равновесии по Курно
Цена
Период
Средние прибыли за эпизод
Прибыль первого агента
Прибыль второго агента
Прибыль агента в равновесии по Курно
Прибыль
Эпизод

Рис. 2: $\gamma = 0$

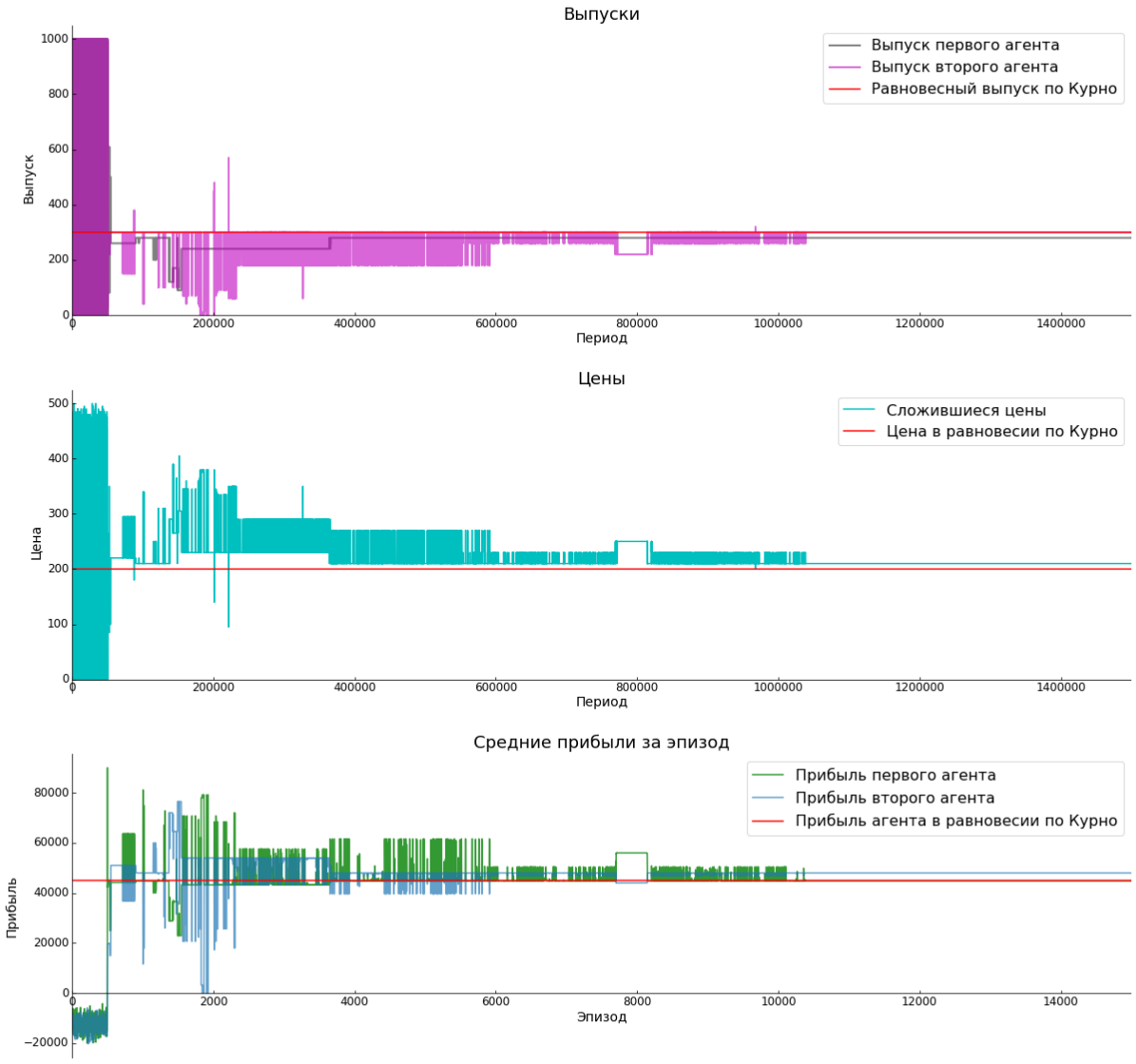

Рис. 3: $\gamma = 0.6$

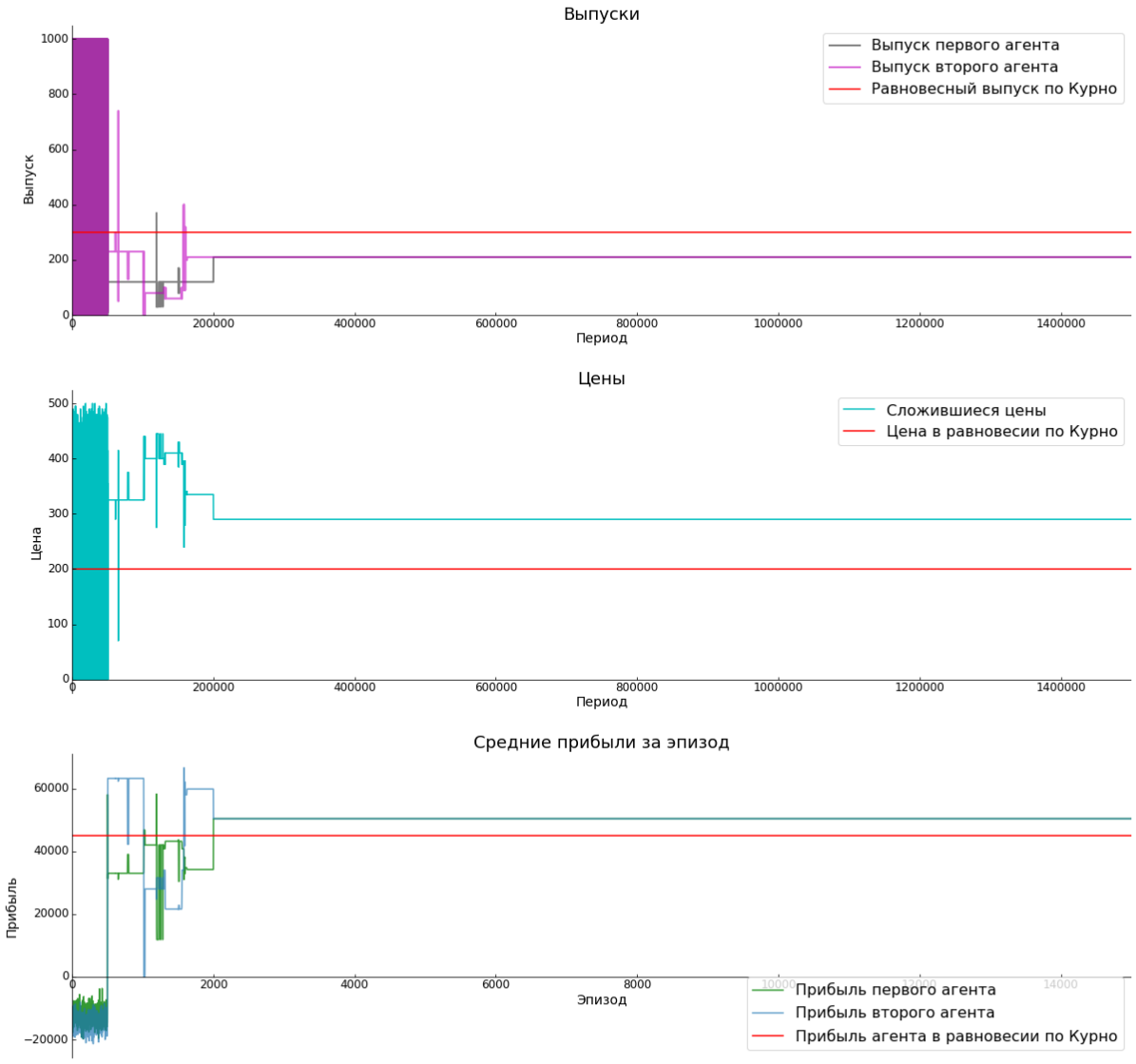

Рис. 4: $\gamma = 0.9$