ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.ЛОМОНОСОВА»

ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА «ФИЗИКО-МАТЕМАТИЧЕСКИХ МЕТОДОВ УПРАВЛЕНИЯ»

БАКАЛАВРСКАЯ РАБОТА

# «УПРАВЛЕНИЕ МЕТРОНОМОМ МЕТОДОМ НЕЙРОСЕТЕВОГО МОДЕЛИРОВАНИЯ»

Выполнил студент
442 группы
Костин Никита Сергеевич

_____

Научный руководитель:
Чл.-к РАН, д.т.н., профессор ИПУ РАН
Галяев Андрей Алексеевич

_____

Допущена к защите: _____
Зав.кафедрой: _____

Москва
2018

**Оглавление**

## Введение

В данной работе рассматриваются методы решения задачи приведения метронома в неустойчивое верхнее положение равновесия с дальнейшей его стабилизацией. Метроном представляет собой модель маятника, способного свободно вращаться вокруг своей оси. Маятник (англ. pole) помещён на каретку (англ. cart), осуществляющую одномерное движение в горизонтальной плоскости под действием силы, постоянной по модулю и переменной по направлению (англ. force), заставляющей, таким образом, маятник вращаться.

Этой проблемой занимался в 50-е гг. XX в. АН СССР П.Л.Капица и решил её теоретически [1]. Однако данная проблема не потеряла своей актуальности и сегодня, так как с ней сталкиваются исследователи при построении алгоритмов стабилизации и приведения в верхнее положение равновесия механизмов в таких областях, как управление подъемным краном в порту или в строительстве, а также при разработке роботизированных систем.

Научная значимость данной работы заключается в применении машинного обучения для развития и обогащения теоретической составляющей оптимального управления механическими объектами.

Практическая же значимость заключается в том, что машинное обучение получило новую область применения в сфере управления объектами, описываемыми уравнениями. Дальнейшее развитие данной области может привести к дальнейшему развитию робототехники.

Цель исследования: разработать и реализовать алгоритм приведения маятника в верхнее неустойчивое положение равновесия.

Задачи исследования:

- разработать среду визуализации движения системы маятник-каретка;
- построить модель нейронной сети, реализующей алгоритм Q-learning;
- написать программу для обучения нейронной сети процессу приведения и стабилизации системы маятник-каретка.

Первый этап работы заключается в подборе такой последовательности направлений силы (влево-вправо), которая осуществляет приведение маятника в неустойчивое верхнее положение равновесия. Второй этап заключается в стабилизации, то есть удержании маятника около этого положения также путем подбора последовательности направлений силы.

Данная задача была решена с помощью машинного обучения (англ. machine learning - ML), так как для ML не требуется выводить аналитические уравнения.

В качестве метода решения используется один из методов машинного обучения — обучение с подкреплением (англ. reinforcement learning - RL). Обучение с подкреплением осуществляет обучение модели, которая не имеет сведений о системе, но способна производить ряд действий с этой системой, получая взамен некоторое вознаграждение за свои действия. Исходя из полученного вознаграждения, модель может изменять своё воздействие на систему. Таким образом, обучение с подкреплением — это оптимальный метод решения задач, в ходе которых осуществляется выбор между долгосрочной и краткосрочной выгодой.

Для реализации решения алгоритма обучения с подкреплением был выбран язык Python, как самый популярный в машинном обучении. На данном языке компанией OpenAI была разработана замечательная среда с открытым исходным кодом для применения алгоритмов обучения с подкреплением к механическим системам - OpenAI Gym. Эта среда реализует анимацию происходящих в системе процессов. Из множества библиотек машинного обучения на Python была выбрана библиотека TensorFlow, так как она одна из самых хорошо описанных в документации библиотек машинного обучения. TensorFlow также является библиотекой с открытым исходным кодом и свободна в использовании.

Данная проблема в России разработана довольно мало, ею занимаются лишь американские компании такие, как OpenAI, а также открытое мировое сообщество программистов (англ. open-source community).

Работа состоит из семи глав. В главе 1 были выведены математические уравнения, описывающие исследуемую систему. Главы 2 и 3 посвящены изложению основных концепций теории машинного обучения и Q-learning. Далее в главе 4 рассказывается о методах реализации данной задачи: визуализации процесса движения системы с помощью среды OpenAI Gym. В главе 5 описывается модель нейронной сети, используемая в данной работе. В главах 6 и 7 разработаны алгоритмы решения задач для решения целей научной работы. Код программы, обеспечивающий реализацию описанных выше задач, приведен в Приложении 1.

## Глава 1. Математическое описание движения маятника

Система маятника и каретки выглядит так, как показано на Рисунке 1.

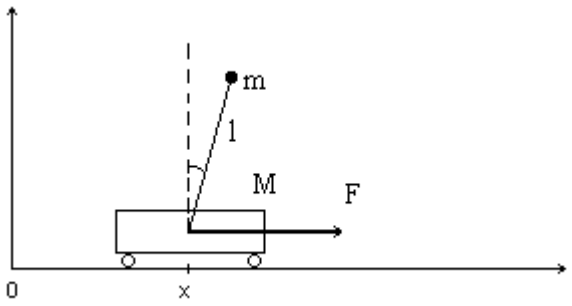



Рисунок 1. Система маятника и каретки

Запишем для этой системы 2-ое уравнение Ньютона для поступательного движения вдоль оси X:

$$(m + M)\ddot{x} = F_{\text{вн}} - f, \qquad (1)$$

где $f$ – сила инерции, которая действует на каретку со стороны маятника. Посчитаем её значение следующим образом: при повороте маятника на бесконечно малый угол $d\theta$ изменение положения каретки $dx$ приближенно можно записать как

$$dx = bd\theta, \qquad (2)$$

где $b$ есть суть расстояние от оси маятника до воображаемой моментальной оси, проходящей через массу $m$ параллельно оси X. Таким образом $b = lcos\theta$. Разделив обе части уравнения (2) на $dt$, получаем:

$$v = l\dot{\theta}cos\theta$$

А значит находим и силу $f$, которая равна:

$$f = ma = m\dot{v} = \frac{d}{dt}\left(\dot{\theta}mlcos\theta\right) = mlcos\theta\ddot{\theta} - ml\dot{\theta}^2cos\theta$$

В итоге можно записать уравнение (1) в виде:

$$(m + M)\ddot{x} = -mlcos\theta\ddot{\theta} + ml\dot{\theta}^2cos\theta + F_{\text{вн}}$$

Теперь используем 2-ой закон Ньютона для вращательного движения:

$$\frac{d\vec{L}}{dt} = \sum\vec{M},$$

где $\vec{L} = I\vec{\omega} = I\vec{\dot{\theta}}$ – момент импульса стержня, момент инерции маятника $I = ml^2 + \frac{1}{3}ml^2 = \frac{4}{3}ml^2$, а $\vec{M} = [\vec{r}, \overrightarrow{F_{\text{вр}}}]$ - момент вращающей маятник силы. В

нашем случае на маятник действуют две силы: сила тяжести и сила инерции со стороны каретки. Запишем обе эти силы:

$$\vec{F}_{\text{т}} = m\vec{g}, \qquad \vec{F}_{\text{ин}} = \vec{F}_{\text{вн}}$$

Запишем теперь значения моментов для каждой силы. Направление обоих моментов совпадает, так как оба момента стремятся повернуть маятник в сторону увеличения угла. Поэтому векторы в дальнейшем можно опустить. Момент силы тяжести запишется следующим образом:

$$M_{\text{т}} = -mglsin\theta$$

А момент силы инерции будет выглядеть так:

$$M_{\text{ин}} = -m\ddot{x}lcos\theta$$

Собирая всё воедино, получим уравнение (3):

$$\frac{4}{3}l\ddot{\theta} = gsin\theta - \ddot{x}cos\theta \qquad (3)$$

В итоге наша система маятника на каретке описывается такой системой уравнений:

$$\begin{cases} (m+M)\ddot{x} + mlcos\theta\ddot{\theta} = ml\dot{\theta}^2cos\theta + F_{\text{вн}} \\ \frac{4}{3}l\ddot{\theta} + \ddot{x}cos\theta = gsin\theta \end{cases} \qquad (4)$$

Если детерминант этой системы не равен нулю,

$$\Delta = \begin{Vmatrix} \frac{4}{3}l & cos\theta \\ mlcos\theta & m+M \end{Vmatrix} = \frac{4}{3}(m+M)l - mlcos^2\theta \neq 0$$

то система (4) имеет единственное решение, которое можно найти по формуле Крамера:

$$\begin{cases} \ddot{\theta} = \frac{1}{\Delta}\begin{Vmatrix} gsin\theta & cos\theta \\ ml\dot{\theta}^2cos\theta + F_{\text{вн}} & m+M \end{Vmatrix} = \frac{(m+M)gsin\theta - ml\dot{}^2cos^2\theta - F_{\text{вн}}cos\theta}{\frac{4}{3}(m+M)l - mlcos^2\theta} \\ \ddot{x} = \frac{1}{\Delta}\begin{Vmatrix} \frac{4}{3}l & gsin\theta \\ mlcos\theta & ml\dot{\theta}^2cos\theta \end{Vmatrix} = \frac{ml^2\dot{\theta}^2cos\theta - mglco}{\frac{4}{3}(m+M)l - mlcos^2\theta} \end{cases} \qquad (5)$$

Полученные поступательное и вращательное ускорения используются для аппроксимации скорости маятника и каретки по следующей формуле:

$$\begin{cases} \dot{\theta}_{t+1} = \dot{\theta}_t + \ddot{\theta}dt \\ \dot{x}_{t+1} = \dot{x}_t + \ddot{x}dt \end{cases} \qquad (6)$$

Аппроксимация производится на равномерной сетке с шагом *dt.*

## Глава 2. Машинное обучение и обучение с подкреплением

Машинное обучение - это область компьютерных наук, которая часто использует статистические методы, чтобы дать компьютерам возможность «учиться» (то есть постепенно улучшать производительность в конкретной задаче) с данными, не будучи заранее явно запрограммированной.

Термин «машинное обучение» был придуман в 1959 году Артуром Самуэлем. Исходя из изучения теории распознавания образов и теории вычислительного обучения в искусственном интеллекте, машинное обучение исследует разработка и построение алгоритмов, которые могут учиться и делать прогнозы на основе данных. Такие алгоритмы преодолевают строгие статические программные инструкции, предсказывая или принимая решения, основанные на имеющихся данных путем построения модели из входных данных. Машинное обучение используется в ряде вычислительных задач, где проектирование и программирование явных алгоритмов с хорошей производительностью является задачей трудной или неосуществимой. Примеры приложений включают фильтрацию электронной почты, обнаружение сетевых злоумышленников или вредоносных инсайдеров, работающих в направлении нарушения данных, а также оптическое распознавание символов, обучение ранжированию и компьютерное зрение.

Машинное обучение тесно связано (и часто совпадает) с вычислительной статистикой, которая также фокусируется на прогнозировании с использованием компьютеров. Оно имеет прочные связи с математической оптимизацией, которая предоставляет методы, теории и области приложений в сфере деятельности. Машинное обучение иногда сочетается с интеллектуальным анализом данных, где последнее больше фокусируется на анализе разведочных данных и известно, как неконтролируемое обучение. Машинное обучение также может быть неконтролируемым и использоваться для изучения и установления базовых моделей поведения для различных субъектов, а затем применяться для определения значимых аномалий и закономерностей.

В области аналитики данных машинное обучение - это метод, использу-

емый для разработки сложных моделей и алгоритмов, которые поддаются прогнозированию. В коммерческом использовании это называется прогностической аналитикой. Эти аналитические модели позволяют исследователям, ученым, инженерам и аналитикам «создавать надежные, устойчивые решения и результаты» и раскрывать «скрытые идеи» посредством изучения данных на предмет исторических отношений и тенденций.

Обучение с подкреплением (англ. reinforcement learning - RL) - это область машинного обучения, касающаяся того, как агенты программного обеспечения должны предпринимать действия в среде, чтобы максимизировать некоторое понятие награды. Проблема, в силу своей обширности, изучается во многих других дисциплинах, таких как: теория игр, теория управления, исследования операций, теория информации, оптимизация на основе моделирования, многоагентные системы, интеллект группы объектов (агентов), статистика и генетические алгоритмы. В литературе по исследованиям и контролю операций обучение с подкреплением называется приближенным динамическим программированием или нейродинамическим программированием. Проблемы интереса к обучению с подкреплением изучались также в рамках теории оптимального управления, которая в основном связана с существованием и характеристикой оптимальных решений, а также алгоритмами их точного вычисления и с обучением или с аппроксимацией, особенно в отсутствие математической модели среды. В экономике и теории игр обучение с подкреплением может быть использовано для объяснения того, как равновесие может возникать при ограниченной рациональности.

В компьютерном обучении среда обычно формулируется как процесс принятия решений по методу Маркова (марковский процесс принятия решений), так как многие алгоритмы обучения с подкреплением для этого используют методы динамического программирования. Основное различие между классическими методами динамического программирования и алгоритмами обучения с подкреплением заключается в том, что последние не предполагают знания точной математической модели марковского процесса принятия реше-

ний и нацелены на большие процессы такого типа, где точные методы становятся неосуществимыми.

Обучение с подкреплением отличается от стандартного контролируемого обучения тем, что правильные пары ввода-вывода не обязательно должны быть представлены. Вместо этого основное внимание уделяется эффективности, которая предполагает поиск баланса между разведкой неизведанной территории, т.е. долгосрочной выгодой, и эксплуатацией существующих знаний, т.е. краткосрочной выгодой. Компромисс в области разведки и эксплуатации был наиболее тщательно изучен в рамках проблемы многорукого бандита и в конечных марковских процессах принятия решений.

Типичное построение сценария обучения: агент предпринимает действия в среде, которые интерпретируется в награду и в представление состояния, которое возвращается агенту.

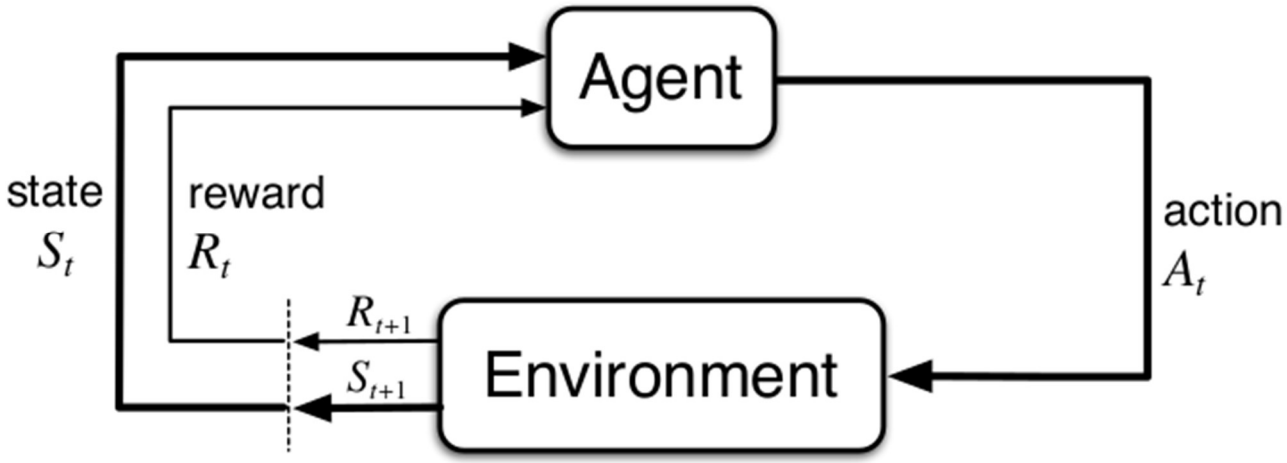
Базовое подкрепление моделируется как процесс принятия марковских решений:

- множество состояний среды и агента *S*;
- множество действий агента *A*
- вероятность $P_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ перехода из состояния *s* в состояние $s'$ при действии $a$.
- $R_a(s, s')$ является непосредственной наградой после перехода из состояния s в состояние s' при действием a.
- правила, описывающие то, что наблюдает агент.

Правила часто случайны. Наблюдение - это обычно скалярное, моментальное вознаграждение, связанное с последним переходом. Во многих работах предполагается, что агент наблюдает текущее состояние окружающей среды (полная наблюдаемость). Если нет, агент имеет частичную наблюдаемость. Иногда набор действий, доступных для агента, ограничен, т.е. нулевой баланс не может быть уменьшен.

Агент взаимодействует со своей средой в дискретных временных шагах. В каждый момент времени *t* агент получает наблюдение $o_t$, которое обычно

включает награду $r_t$. Затем он выбирает действие $a_t$ из множества доступных действий, которое затем отправляется среде. Окружающая среда переходит в новое состояние $s_{t+1}$ с наградой $r_{t+1}$, связанная с переходом $(s_t, a_t, s_{t+1})$. Целью обучающегося агента является набор как можно большего вознаграждения. Агент может выбрать (возможно, случайно) любое действие в зависимости от своей памяти.




Когда производительность агента сравнивается с производительностью оптимально действующего агента, разница в производительности порождает понятие ошибки. Чтобы действовать почти оптимально, агент должен думать о долгосрочных последствиях своих действий, т.е. максимизировать будущую награду, хотя непосредственное вознаграждение может быть и отрицательным.

Таким образом, обучение с подкреплением особенно хорошо подходит для проблем, в которых есть выбор между долгосрочной и краткосрочной выгодами. Он успешно применяется к различным проблемам, включая управление роботом, планирование движения лифта, телекоммуникации.

Два элемента упрощают обучение с подкреплением: использование примеров для оптимизации производительности и использование аппроксимации функций для работы с обширной средой. Благодаря этим двум ключевым компонентам обучение с подкреплением может использоваться в следующих ситуациях:

- известна модель среды, но аналитическое решение недоступно;
- дана только имитационная модель среды (тема оптимизации на основе моделирования). Единственный способ собрать информа-

цию об окружающей среде - это взаимодействовать с ней.

Первые две из этих проблем можно рассматривать как проблемы планирования, поскольку доступна какая-либо модель среды, в то время как последнюю можно рассматривать как проблему подлинного обучения. Тем не менее, обучение с подкреплением преобразует обе проблемы планирования в проблемы машинного обучения.

Рассмотрим понятие критерия оптимальности. Выбор действия агента моделируется как отображение, называемое политикой:

$$\pi: S \otimes A \rightarrow [0,1]$$

$$\pi(a|s) = \Pr(a_t = a|s_t = s)$$

Данное отображение выражает вероятность принятия действия *a* в состоянии *s*. Существуют также нереализуемые политики.

Рассмотрим также понятие функции значения состояния. Функция значения состояния $V_\pi(s)$ определяется как ожидаемый возврат, начиная с состояния *s*, то есть $s = s_0$ и последовательно следует политике $\pi$. Иначе говоря, получается, что функция значения оценивает, насколько «точно» она должна находиться в определенном состоянии:

$$V_\pi(s) = E[R] = E[\sum_{t=0}^{\infty} \gamma^t r_t \,|s_0 = s],$$

где случайная величина $R$ обозначает возврат и определяется как сумма будущих вознаграждений:

$$R = \sum_{t=0}^{\infty} \gamma^t r_t,$$

где $r_t$ - вознаграждение на этапе *t*, $\gamma \in [0,1]$ - дискаунт-фактор.

Алгоритм должен найти политику с максимальным ожидаемым возвратом. Из теории марковского случайного процесса принятия решений (MDP) известно, что без ограничения общности поиск может быть ограничен набором так называемых стационарных политик. Политика является стационарной, если возвращаемое ею действие зависит только от последнего посещенного состояния (из истории наблюдателя). Поиск может быть дополнительно ограничен детерминированными стационарными политиками. Детерминированная стацио-

нарная политика выбирает действия, основанные на текущем состоянии. Поскольку любая такая политика может быть отождествлена с отображением из множества состояний в набор действий, эти политики можно отождествлять с такими стационарными политиками без потери общности.

Функция значения пытается найти политику, которая максимизирует возврат, поддерживая набор оценок ожидаемых результатов для некоторой политики (обычно либо «текущая» (on-policy), либо оптимальная (off-policy)). Эти методы основаны на теории MDP, где оптимальность определяется в том смысле, который сильнее, чем предыдущий: политика называется оптимальной, если она достигает наилучшего ожидаемого возврата из любого начального состояния, т.е. исходные распределения не играют никакой роли в этом определении. И снова оптимальная политика всегда может быть найдена среди стационарных политик.

Чтобы определить оптимальность формальным образом, определим значение политики

$$V^{\pi}(s) = E[R|s, \pi],$$

где $R$ обозначает случайный возврат, связанный со следующим $\pi$ из начального состояния $s$. Определим $V^{*}(s)$ следующим образом:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s)$$

Политика, которая достигает этих оптимальных значений в каждом состоянии, называется оптимальной. Ясно, что оптимальная в этом сильном смысле политика также оптимальна в том смысле, что она максимизирует ожидаемое вознаграждение $\rho^{\pi}$, так как, $\rho^{\pi} = E[V^{\pi}(S)]$, где $S$ - это случайное выборка состояния из распределения $\mu$.

Хотя значения состояний достаточно для определения оптимальности, полезно определить значения действия. Учитывая состояние $s$, действие $a$ и политику $\pi$, значение действия пары *(s, a)* под действием $\pi$ определяется формулой:

$$Q^{\pi}(s, a) = E[R|s, a, \pi],$$

где $R$ теперь обозначает случайный возврат, связанный с первым действием $a$ в

состоянии *s* и последующим $\pi$.

Теория MDP утверждает, что если $\pi^*$ является оптимальной политикой, мы действуем оптимально (принимаем оптимальное действие), выбирая действие из $Q^{\pi^*}(s, \cdot)$ с наивысшим значением в каждом состоянии *s*. Функция действия-значения такой оптимальной политики $Q^{\pi^*}$ называется оптимальной функцией действия-значения и является обычно обозначаемый $Q^*$. Таким образом, знание оптимальной функции стоимости действия достаточно, чтобы знать, как действовать оптимально.

Если предположить полное знание MDP, то два основных подхода к вычислению оптимальной функции действия - это итерация значения и итерация политики. Оба алгоритма вычисляют последовательность функций $Q_k(k = 0,1,2, \dots)$, которые сходятся к $Q^*$. Вычисление этих функций связано с вычислением ожиданий по всему пространству состояний, что нецелесообразно для всех, кроме наименьших (конечных) MDP. В методах обучения с подкреплением ожидания аппроксимируются усреднением по выборкам и использованием методов аппроксимации функций, чтобы справиться с необходимостью представлять функции значений в больших пространствах действия состояния.

## Глава 3. Q-learning

Q-learning является одним из самых популярных способов реализации обучения с подкреплением.

Q-learning не требует модели окружающей среды, а может обрабатывать проблемы со стохастическими переходами и вознаграждениями, не требуя адаптации.

Для любого конечного марковского процесса принятия решений (FMDP) Q-learning в конечном итоге находит оптимальную политику в том смысле, что ожидаемое значение общей награды возвращается за все последующие шаги, начиная с текущего состояния, и является максимально достижимым. Q-learning может определять оптимальную политику отбора действий для любого данного FMDP.

Название функции «Q», которая возвращает вознаграждение, используемое для обеспечения подкрепления, обозначает «качество» (англ. quality) действия, предпринимаемого агентом в определенном состоянии.

Опишем алгоритм Q-learning.

Вес для шага из состояния $\Delta t$ шагов в будущее рассчитывается как $\gamma^{\Delta t}$. $\gamma$ (дискаунт-фактор) - это число от 0 до 1 ($0 \leq \gamma \leq 1$) и больше влияет на оценку вознаграждений, полученных ранее, чем на полученные позже, что отражает смысл выражения «хорошее начало». $\gamma$ также можно интерпретировать как вероятность успеха (или выживания) на каждом шаге $\Delta t$.

Следовательно, алгоритм имеет функцию, которая вычисляет качество комбинации действие-состояние:

$$Q: S \otimes A \to \mathbb{R}$$

Прежде чем начать обучение, $Q$ инициализируется, возможно, произвольным фиксированным значением, выбранным программистом. Затем в каждый момент времени $t$ агент выбирает действие $a_t$, получает награду $r_t$, переходит в новое состояние $s_{t+1}$. Это может зависеть как от предыдущего состояния $s_t$, так и от выбранного действия, и в конце значение функции $Q$ обновляется. Ядром алгоритма является простое итерационное обновление значения, исполь-

зуя средневзвешенное значение старого значения и новую информацию:

$$Q(s_t, a_t) \leftarrow (1-\alpha) \underbrace{Q(s_t, a_t)}_{\text{старое значение}} + \overbrace{\alpha}^{\text{скорость обучения}} \overbrace{(r_t + \gamma \max_a Q(s_{t+1}, a))}^{\text{значение обучения}} \quad (7)$$

Это так называемое рекурсивное уравнение Беллмана, позволяющее итерационно вычислять значение $Q(s_t, a_t)$, максимизировав при этом значение $Q(s_{t+1}, a)$ на множестве возможных действий $A$.

В уравнении (7) $r_t$ - вознаграждение, наблюдаемое в текущем состоянии $s_t$, а $\alpha$ - это скорость обучения ($0 < \alpha < 1$).

Эпизод алгоритма заканчивается, когда состояние $s_{t+1}$ является конечным состоянием. Тем не менее, Q-learning может также применяться и в неэпизодических задачах. Если дискаунт-фактор меньше 1, то значения действия конечны, даже если проблема может содержать бесконечные циклы.

Для всех конечных состояний $s_f$, $Q(s_f, a)$ не обновляется, а устанавливается значение вознаграждения r, наблюдаемое в состояния $s_f$. В большинстве случаев $Q(s_f, a)$ можно считать равным нулю.

## Глава 4. OpenAI Gym

Компанией OpenAI в 2016 году была выпущена версия проекта OpenAI Gym. OpenAI Gym направлен на предоставление достаточно простого в установке тестирования искусственного интеллекта в различных средах. OpenAI Gym можно использовать только с Python, а активная работа сосредоточена на странице GitHub (**www.github.com/openai/gym**).

Развитие обучения с подкреплением замедлялось двумя факторами:

- необходимость в лучших тестах. В контролируемом обучении прогресс был обусловлен большими размеченными наборами данных, такими как ImageNet. В обучении с подкреплением (RL) самым близким эквивалентом обучению с учителем будет большой и разнообразный набор сред. Однако существующие коллекции с открытым исходным кодом для сред RL не имеют достаточного разнообразия, и их часто сложно даже настроить и использовать.
- отсутствие стандартизации среды, используемой в публикациях.

Тонкие различия в определении проблемы, такие как функция вознаграждения или набор действий, могут кардинально изменить сложность задачи. Этот вопрос затрудняет воспроизведение опубликованных исследований и сравнение результатов различных работ.

OpenAI Gym позволяет решить обе эти проблемы и дает возможность тестирования алгоритмов обучения с подкреплением на стандартных средах, таких как «CartPole-v0». Также OpenAI Gym позволяет разработать свою собственную среду - механическую систему, описываемую заданными уравнениями. Это и было сделано в данной задаче. Пример среды OpenAI Gym Classic Control для «CartPole-v0» показан на Рисунке 2.

Созданная среда «CartPole-v1» (код среды см. www.github.com/nik31096) является производной от стандартной среды «CartPole-v0» и имеет следующие отличия:

- свобода передвижения маятника, т.е. эпизод, не заканчивается,

когда значение угла отклонения маятника от верхнего положения равновесия превышает стандартные 12°;

- пространство (экран), в котором может двигаться каретка, увеличено до размеров экрана монитора, чтобы у системы каретка-маятник было больше свободы для маневров, так как обучение системы приведению маятника в верхнее положение требует больше пространства, чем удерживание (стабилизация) маятника в верхнем положении равновесия.

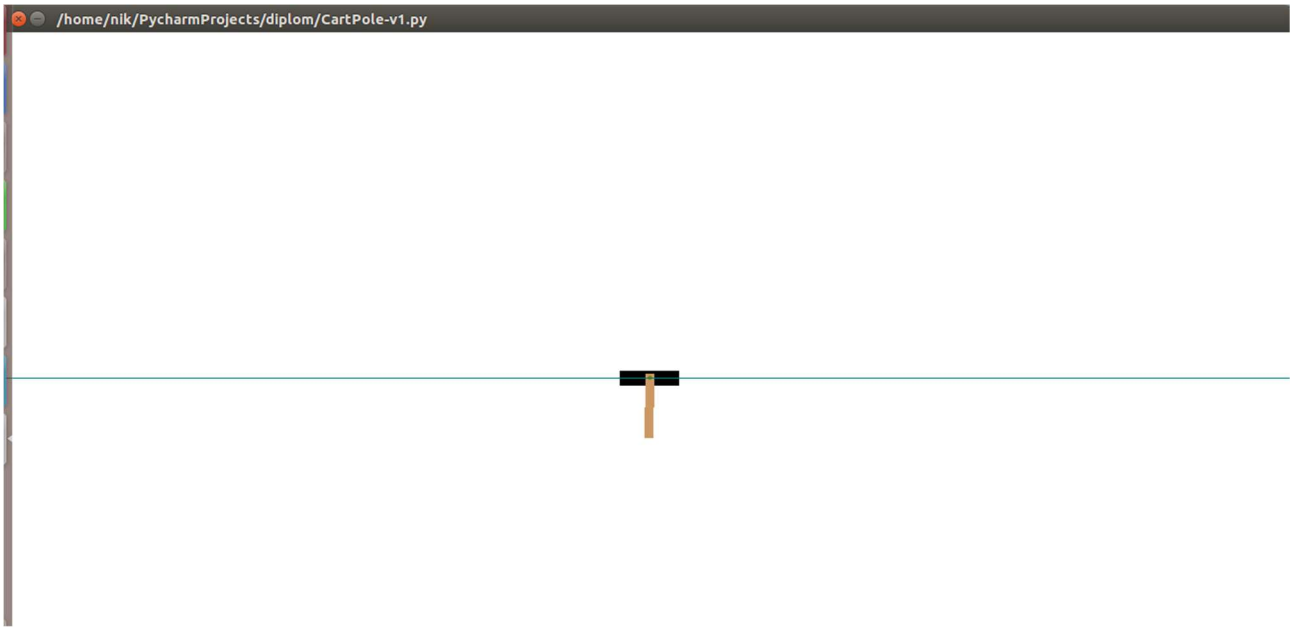



Рисунок 2. Пример интерфейса OpenAI Gym Classic Control

## Глава 5. Модель нейронной сети

В качестве фреймворка для работы с нейронной сетью был выбран фреймворк от компании Google - TensorFlow. Эта библиотека машинного обучения хорошо задокументирована на официальном сайте www.tensorflow.com. Также у этого фреймворка есть инструмент для отображения графа нейронной сети - TensorBoard.

Как было описано в главе 3 Q-learning, в данной задаче используется Deep Q-learning с двумя внутренними слоями. Сама нейронная сеть состоит из нескольких частей:

- «eval_net» или исполняющая сеть – элемент графа нейронной сети, в котором находится два внутренних слоя. В этом элементе вычисляется предсказываемое нейронной сетью значение действия.
- «train_net» или тренировочная сеть - элемент графа нейронной сети, в котором также находится два внутренних слоя. И в данном элементе путем анализа предсказанного исполняющей сетью действия для данного состояния.
- «loss» или функция потерь – элемент нейронной сети, позволяющий находить ошибку нейронной сети. Существует несколько видов функции потерь. В данной задаче используется функция квадратичной ошибки.
- «train» или оптимизатор - элемент нейронной сети, позволяющий оптимизировать параметры нейронной сети по данным, полученным от функции потерь. Существует множество видов оптимизаторов. В данной работе используется RMSProb оптимизатор.

Граф нейронной сети, используемый в данной работе, представлен на Рисунке 3 с применением TensorBoard.

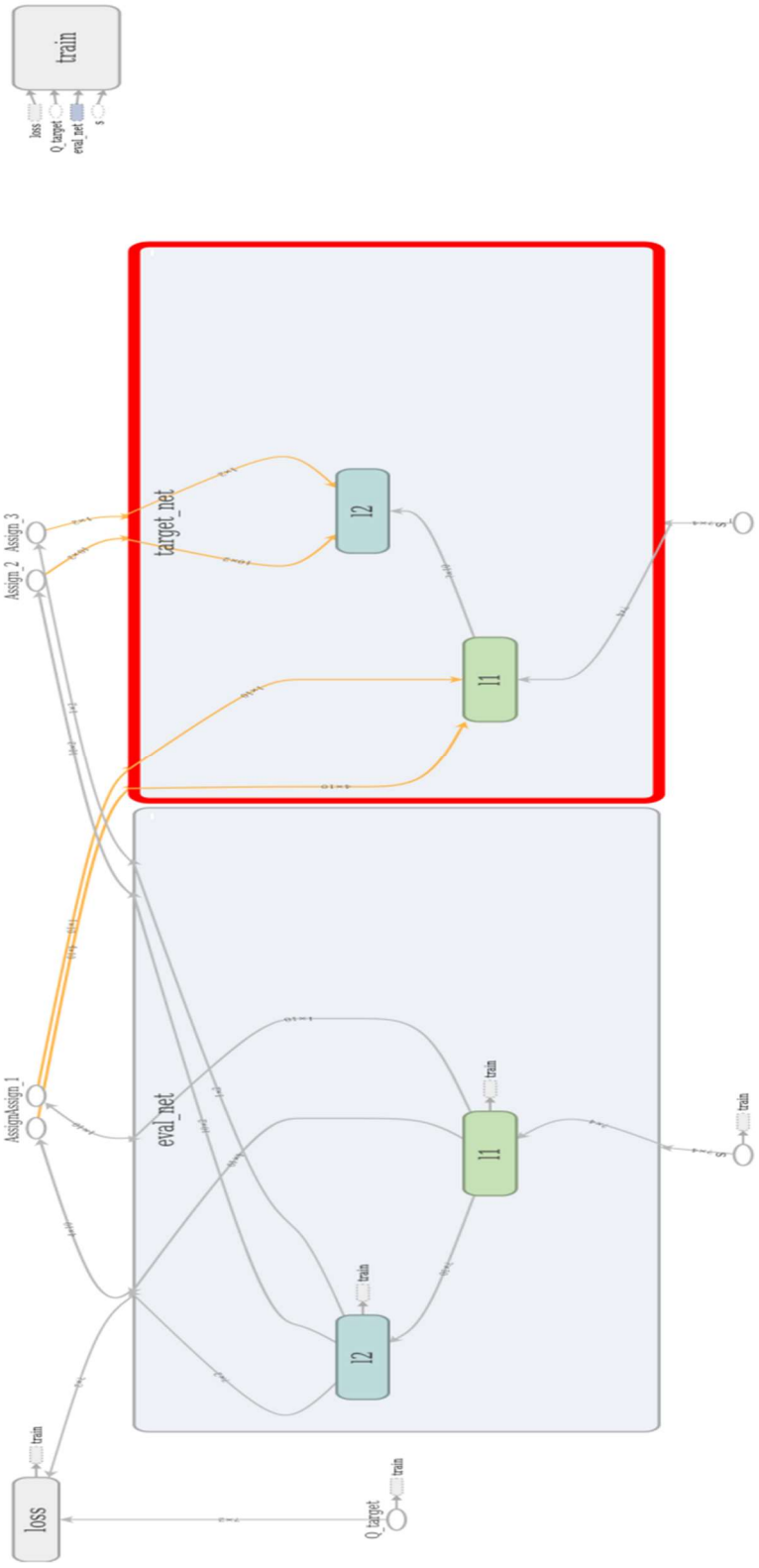



Рисунок 3. Граф нейронной сети в Tensorboard

## Глава 6. Процесс приведения маятника в верхнее положение

Процесс приведения маятника в верхнее положение из нижнего удовлетворяет следующим условиям:

- Каретка маятника может перемещаться только в заданных пределах (влево и вправо от нуля), при достижении которых эпизод заканчивается.
- Так как целью процесса является приведение маятника в верхнее положение с конечной скоростью, близкой к нулю, то при одновременном удовлетворении условий $|\theta| < \varepsilon_1$ и $\left|\dot{\theta}\right| < \varepsilon_2$ эпизод также заканчивается.

Для реализации данного процесса необходимо правильно составить функцию вознаграждения. Действия, которые приводят к требуемому результату, мы поощряем, добавляя награду к функции вознаграждения агента за эпизод. А действия, приводящие к нежелательным результатам, мы пресекаем, штрафуя агента путем вычитания из функции вознаграждения некоторого значения. Таким образом, функция вознаграждения должна удовлетворять следующим условиям:

- при попадании каретки маятника в зону, близкую к краю, со скоростью, направленной к этому краю, необходимо штрафовать агента;
- когда угол маятника становится меньше 90° по модулю, мы должны добавлять к функции вознаграждения некоторое значение;
- когда угол маятника близок к верхнему положению равновесия и скорость маятника мала, то мы также должны добавлять к функции вознаграждения некоторое значение;
- также в функции вознаграждения должно быть слагаемое, поощряющее маятник подниматься вверх.

Учитывая данные требования к функции вознаграждения, можно записать её следующим образом:

$$R(x,\theta) = A|cos\theta - 1| + Bf_{\dot{\theta}} + Cf_x + Df_\theta, \quad (8)$$

где константы *A, B, C, D* следует подбирать вручную, пробуя различные предельные значения. При большом значении какого-либо коэффициента агент будет большее внимание уделять именно удовлетворению соответствующей коэффициенту функции. От подобранных значений зависит, насколько хорошо и быстро агент сможет понять, что ему нужно делать. Вид неизвестных пока функций $f_{\dot{\theta}}$, $f_x$ и $f_\theta$ определяется следующим образом:

$$f_{\dot{\theta}} = \begin{cases} \varepsilon_2 - \left|\dot{\theta}\right|, \text{при } |\theta| < \varepsilon_1 \\ 0 \quad , \text{при } |\theta| \geq \varepsilon_1 \end{cases}, \quad f_\theta = \begin{cases} 1, \text{при } |\theta| < \frac{\pi}{2} \\ 0, \text{при } |\theta| \geq \frac{\pi}{2}, \end{cases}$$

$$f_x = \begin{cases} 1, \text{при } \left(x \leq -x_{\text{кр}} \ \cup \ \dot{x} < 0\right) \cap \left(x \geq x_{\text{кр}} \ \cup \ \dot{x} > 0\right) \\ 0, \ \text{в противном случае} \end{cases}$$

Значения $\varepsilon_1$ и $\varepsilon_2$ также следует подбирать вручную. Сила $f_{\dot{\theta}}$ отвечает за награду агенту, когда тот при значении угла $\theta$ меньше $\varepsilon_1$ имеет скорость $\dot{\theta}$ меньше $\varepsilon_2$. Сила $f_x$ дает отрицательную награду (забирает награду) в ситуации, когда агент находится близко к краю и имеет скорость, направленную в сторону края экрана. Сила $f_\theta$ поощряет нахождение конца маятника выше прямой линии, по которой двигается каретка.

Используя полученные в главе 1 уравнения (5) для ускорений маятника и каретки, а также уравнения (6), описывающие аппроксимацию угловой скорости маятника и поступательной скорости каретки. Сила $F_{\text{вн}}$ формуле (5) заменяется нами на дискретную силу, зависящую от времени по закону:

$$F_{\text{вн}} = \begin{cases} F, \text{при значении действия } a_t = 1 \\ -F, \text{при значении действия } a_t \neq 1\text{'} \end{cases} \quad (9)$$

где F – const, [F]=Н.

Применим полученную функцию вознаграждения (8) к задаче приведения маятника в верхнее положение. Параметры *A, B, C, D,* с помощью которых удается получить неплохие результаты, имеют значения: *A=1, B=1000, C=-60, D=400.* Значения же малых $\varepsilon_{1,2}$ получились следующими: $\varepsilon_1 = 0.5$, $\varepsilon_2 = 0.3$. Подставив значения всех параметров в формулу (7), после обучения нейронной сети получаем следующие результаты: после 25 эпизодов агент начинает пони-

мать, какой тип движения необходим для приведения маятника в верхнее положение. После же 61 эпизода у него получилось сделать это за 173 временных отрезка, или за 5.19 секунд. График управления, которое применяет агент для приведения маятника в неустойчивое верхнее положение, приведён на рисунке 4.

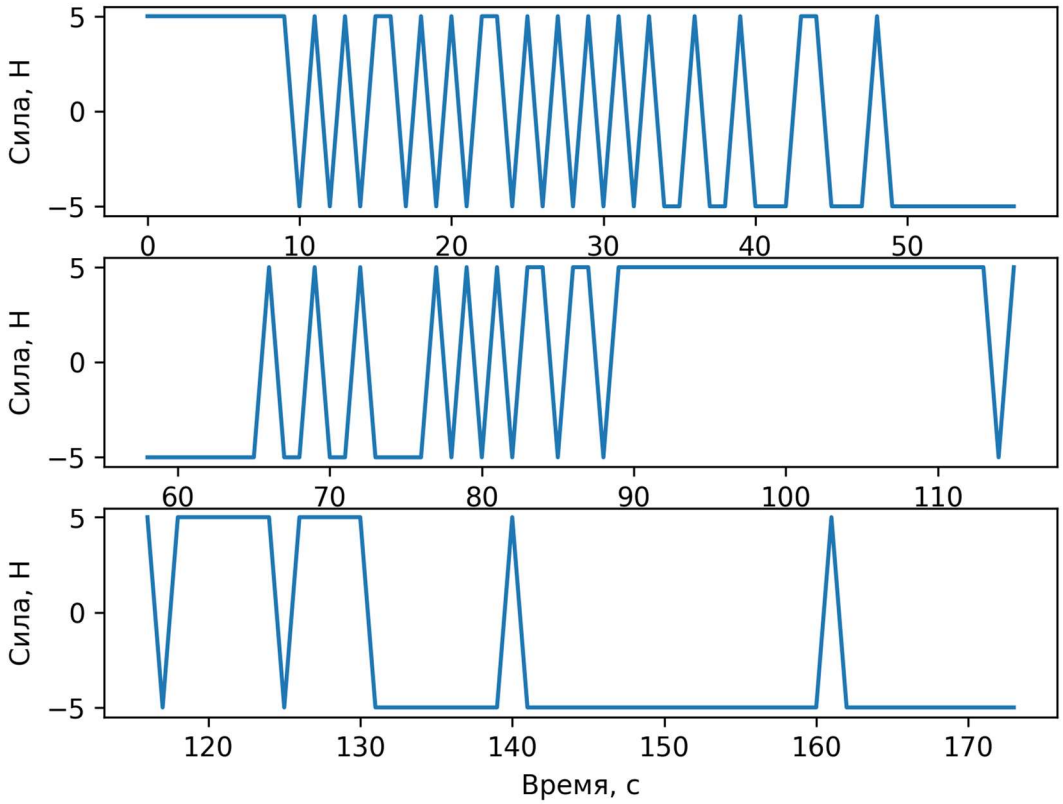



Рисунок 4. График зависимости управления от времени, приводящий маятник из нижнего устойчивого положения в верхнее неустойчивое положение равновесия.

В дальнейшем, совершенствуя вид функции (8), можно получить большое быстродействие, т.е. более качественный алгоритм.

## Глава 7. Процесс стабилизации маятника у верхнего положения равновесия.

Процесс стабилизации маятника у верхнего положения равновесия удовлетворяет следующим условиям:

- Угол изменяется в заданном пределе, при превышении которого эпизод заканчивается.
- При удержании маятника в заданном интервале углов заданное время эпизод также заканчивается.

Для реализации такой системы также необходимо правильно составить функцию вознаграждения (англ. reward). В данном процессе она должна удовлетворять следующим условиям:

- В функции вознаграждения должно быть слагаемое, отвечающее за положение каретки маятника такое, чтобы наибольшая награда давалась, если каретка находится в центре, так как там у неё есть больше пространства для маневров по удержанию маятника в равновесии, а при приближении к краю увеличивается вероятность окончания эпизода.
- Также в функции вознаграждения должно быть слагаемое, отвечающее за положение вершины маятника: чем ближе вершина маятника к положению равновесия, тем большую награду получает агент. Другими словами, чем выше вершина маятника, тем лучше.

Учитывая данные требования к функции вознаграждения, можно записать её следующим образом:

$$R = \left\{\frac{x_{\text{кр}} - |x|}{x_{\text{кр}}} - A\right\} + \left\{\frac{\theta_{\text{кр}} - |\theta|}{\theta_{\text{кр}}} - B\right\}, \qquad (10)$$

где константы $A$ и $B$ также определяются простым подбором, а значения $x_{\text{кр}}$ и $\theta_{\text{кр}}$ – есть крайнее положение каретки, при котором эпизод заканчивается, и максимально возможный угол отклонения от горизонтали, при превышении которого маятник считается упавшим и, следовательно, эпизод также заканчива-

ется.

Применим полученное уравнение (9) для обучения агента для значений параметров $A$ и $B$: $A=0.8$, $B=0.5$. Значение постоянной $F$ из формулы (9) равно: $F=5$. После обучения нейронной сети получаем следующие результаты: после 89 попыток, закончившимися либо падением маятника ниже 12°, либо превышением кареткой координаты $x_{кр}$. После 89 эпизодов агент может успешно удерживать маятник в верхнем положении более 200 временных шагов, т.е. около 6 секунд. График управления, которое применяет агент для стабилизации маятника, приведён на Рисунке 5.

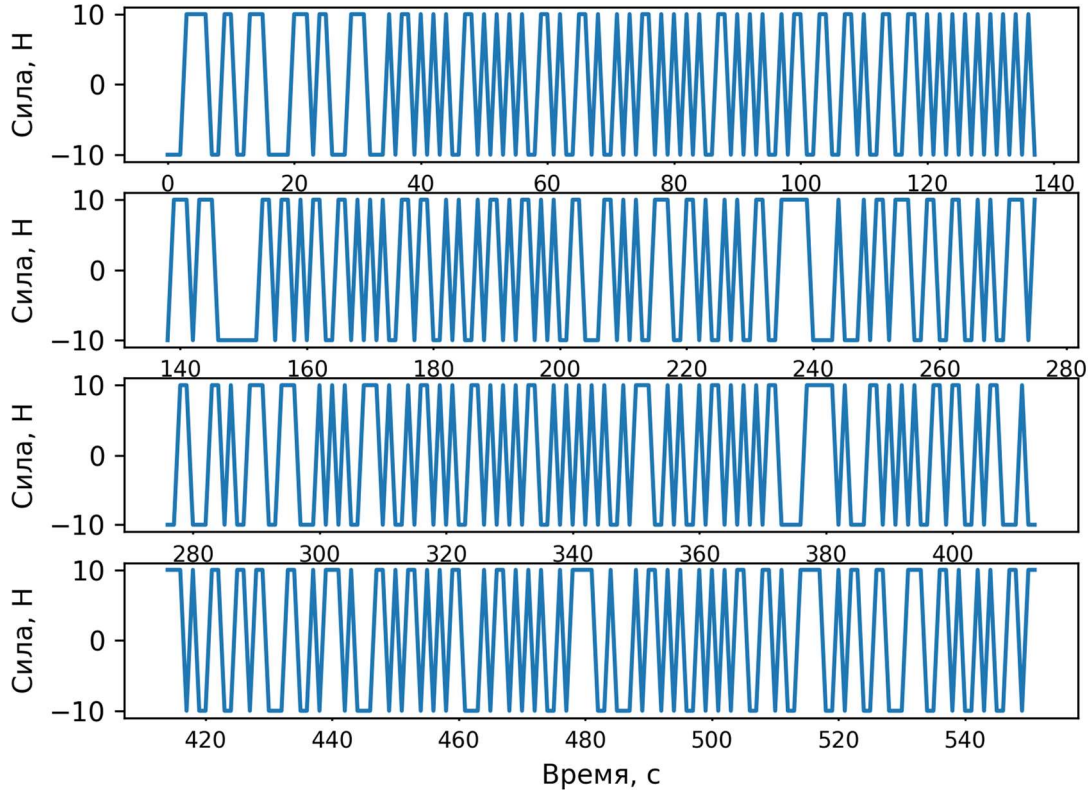



Рисунок 5. График зависимости управления от времени для агента на 89 эпизоде обучения.

## Заключение

В данной работе были получены следующие результаты:

- была построена математическая модель системы маятник-каретка;
- была построена система маятник-каретка в среде OpenAI Gym;
- к данной системе была применена модель глубокой нейронной сети типа Deep Q-learning;
- путем применения Q-learning алгоритма и обучения интеллектуальной системы получено управление, которое приводит систему маятник-каретка в неустойчивое верхнее положение;
- также получено управление, которое осуществляет стабилизацию маятника в неустойчивом верхнем положении в течении некоторого времени.

Подводя итоги работы, можно сказать, что применение машинного обучения, нейронных сетей, в особенности Q-learning, в теории управления является одной из самых перспективных областей. В данной работе были наглядно продемонстрированы возможности искусственного интеллекта к самообучению, используя метод «проб и ошибок».

Дальнейшее развитие идеи приведения маятника в верхнее неустойчивое положение путем обучения системы маятник-каретка может заключаться в добавлении в среду «CartPole-v0» ещё одного маятника. Данная задача представляет интерес как с практической, так и теоретической точки зрения. Задача приведения одного маятника в верхнее неустойчивое положение является классической задачей теории управления и была решена довольно давно. Однако задача приведения двух и более маятников, находящихся на одной каретке, в верхнее положения равновесия и дальнейшая их стабилизация является неразрешимой для современной теории управления.

Для машинного обучения задача приведения двух и более маятников в верхнее положение равновесия ничем не отличается от задачи приведения одного маятника. Таким образом, применяя машинное обучение в теории управ-

ления, возможно получить управление (даже оптимальное) сложной системой маятников на каретке, которое не может быть получено теоретически.

Также необходимость в развитии применения машинного обучения в теории управления может мотивироваться развитием робототехники, так как робот - это механическая система, описываемая уравнениями механики.

Нет сомнения в том, что в недалеком будущем бурно развивающая теория машинного обучения закрепится в качестве одного из основных методов теории управления.

## Литература

[1] П.Л.Капица - «Маятник с вибрирующим подвесом», УФН (Май 1951г.)

Доступ в интернете:

http://science.bagmanov.ru/Резонанс/Маятник/Маятник%20Капицы.pdf

[2] Перевернутый маятник на каретке как объект управления.

Доступ в интернете: https://studfiles.net/preview/735182/page:2/

[3] Сайт англоязычной Википедии

Доступ в интернете: https://en.wikipedia.org/wiki/Reinforcement_learning

[4] Richard S. Sutton and Andrew G. Barto - Reinforcement Learning:
An Introduction

[5] Martin T. Hagan1, Howard B. Demuth and Orlando De Jesus - An introduction to the use of neural networks in control systems
(INTERNATIONAL JOURNAL OF ROBUST AND NONLINEAR CONTROL)

# Приложение 1. Код программы «Cartpole-v1.py».

```
"""
Deep Q network,
Using:
Tensorflow: 1.6
gym: 0.10.5
"""

import gym
import gymCartpoleBottom.gym_cartpolebottom # for new enviroment
from RL_brain import DeepQNetwork
from matplotlib import pyplot as plt
import math
import tensorflow as tf

env = gym.make('CartpoleBottom-v0')

print(env.action_space.n)
print(env.observation_space)
print(env.observation_space.high)
print(env.observation_space.low)

eps1 = 0.5 # for theta
eps2 = 0.3 # for theta_dot

def theta_dot_reward(thta, thta_dot):
    rwrd = 0
    if abs(thta) < eps1:
        rwrd += 1000*abs(eps2 - abs(thta_dot))
    return rwrd

def x_reward(x_thres, X, X_dot):
    rwrd = 0
    if (X < -(x_thres - 1) and X_dot < -0.1) or (X > (x_thres - 1) and X_dot > 0.1):
        rwrd -= 60
    return rwrd

def theta_more_half_pi(thta):
    rwrd = 0
    if abs(thta) < math.pi/2:
        rwrd += 400
    return rwrd
```

```
RL = DeepQNetwork(n_actions=env.action_space.n,
                  n_features=env.observation_space.shape[0],
                  learning_rate=0.001, e_greedy=0.9,
                  replace_target_iter=300, memory_size=8000,
                  e_greedy_increment=0.00008,
                  output_graph=True)

total_steps = 0

time = 1
times = []
actions = []

episodes = []
rewards = []
epsilons = []
count = 0

for i_episode in range(500):
    observation = env.reset()
    ep_r = 0
    while True:
        env.render()
        action = RL.choose_action(observation)
        times.append(time)
        observation_, reward, done, info, force = env.step(action)
        actions.append(force)
        x, x_dot, theta, theta_dot = observation_
        r2 = (math.cos(theta) - 1)

        reward = r2 + theta_dot_reward(thta=theta, thta_dot=theta_dot) + \
                 x_reward(x_thres=env.x_threshold, X=x, X_dot=x_dot) + theta_more_half_pi(thta=theta)

        RL.store_transition(observation, action, reward, observation_)
        ep_r += reward
        if total_steps > 100:
            RL.learn()
        if done:
            episode_reward = round(ep_r, 2)
            RL_epsilon = round(RL.epsilon, 2)
            episodes.append(i_episode)
            rewards.append(episode_reward)
```

```
            epsilons.append(RL_epsilon)
            print('episode: ', i_episode,
                  'ep_r: ', episode_reward,
                  ' epsilon: ', RL_epsilon)
            break
        time += 1
        observation = observation_
        total_steps += 1
        count += 1
        if count > 10000:
            break

    count = 0
    if env.status_done:
        env.status_done = False
        print(f"Episode {i_episode} DONE with {time} timesteps and reward {ep_r}")
        if time < 200:
            print("Model's saved in path /tmp/model.ckpt")
            amt_div = 3
            in1 = len(times) // amt_div
            fig = plt.figure()
            for i in range(1, amt_div + 1):
                ax = fig.add_subplot(amt_div, 1, i)
                ax.plot(times[(i - 1) * in1:i * in1], actions[(i - 1) * in1:i * in1])
                ax.set_ylabel("Сила, Н")
                ax.set_xlabel("Время, с")
                plt.savefig("bringing01.png", dpi=300)
            plt.show()
            break

    actions = []
    times = []
    time = 0
plt.plot(episodes, rewards, 'r-')
plt.show()
```