МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

федеральное государственное бюджетное образовательное учреждение высшего образования «Красноярский государственный педагогический университет им. В.П. Астафьева»

(КГПУ им. В.П. Астафьева)

Институт математики, физики и информатики

Базовая кафедра информатики и информационных технологий в образовании (ИиИТО)

Гилязов Руслан Талгатович

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Подготовка обучающихся инженерно-технического профиля по направлению «Машинное обучение»

Направление подготовки: 44.03.01 Педагогическое образование

Направленность (профиль) образовательной программы: Физика и информатика

ДОПУСКАЮ К ЗАЩИТЕ

Зав. кафедрой ИиИТО

д-р пед. наук, проф.

_____ Н.И. Пак

(дата и подпись)

Руководитель

канд. пед. наук, доцент каф. ИиИТО

_____ Д.В. Романов

(дата и подпись)

Обучающийся

_____ Р.Т. Гилязов

(дата и подпись)

Дата защиты_____

Оценка (прописью)_____

Красноярск 2018

Оглавление

Введение

Такие технологии, как искусственный интеллект (ИИ) и машинное обучение (machine learning, ML) активно внедряются в производство [7] [6]. Как технология и как раздел науки, это самобытная область на стыке программирования, линейной алгебры, математического анализа, теории игр, теории вероятностей и математической статистики. Предоставляемые ИИ новые решения могут в ближайшее время вытеснить широкий класс промышленных задач, ранее решавшихся специалистами или очень сложным программированием [5]. Также появились и уже получили широкое распространение языки программирования, пригодные для создания мощных моделей, подходящих для использования при обучении основам ИИ в школе (язык Питон - один из языков для сдачи заданий ЕГЭ, в отдельных учреждениях читается уже с восьмого класса [4]).

Большинство алгоритмов ИИ требует комбинированного использования сразу нескольких высокоуровневых абстракций одновременно. В процессе работы (само) обучающегося алгоритма часто отсутствует детерминизм, вследствие чего отладка и настройка программы требует наличия ранее полученного опыта даже просто для понимания того, что ошибки уже были допущены, и уж тем более для поиска источника ошибок и его природы, если ошибки были допущены. Культура такого мышления может быть дана или широким классическим образованием, или аккуратным выполнением нескольких, тщательно спроектированных примеров, построенных из первых принципов, ab initio. [9]

Рынок труда и государственный заказ также обуславливают актуальность обучения фундаменту ИИ-дисциплин уже в школе. Например, участники олимпиады НТИ должны активно использовать такие технологии, как поиск пути, распознавание образов, обучение с подкреплением и многие другие разделы ИИ для решения задач второго и заключительного туров. [8]

Основной гипотезой работы является следующая: если разработка курса по машинному обучению будет построена на основе эмулятора, демонстрирующего все используемые понятия в компактном, ясном, читаемом и исполняемом виде, без сторонних библиотек, из первых принципов, то это позволит заложить фундамент, необходимый для понимания природы используемых алгоритмов, направления дальнейшего научного поиска, и для будущей эффективной инженерной подготовки школьников. Систему заданий курса можно также построить программированием модели в эмуляторе. Подобный подход уже показал свою эффективность при обучении модели ИИ[10].

Для этого в главе 1 проведён анализ дисциплины “Искусственный интеллект” с целью обзора основных разделов для выделения предмета изучения, даны ссылки для дальнейшей работы.

Глава I Искусственный интеллект

История искусственного интеллекта

Искусственный интеллект - это раздел информационных технологий, рассматривающий моделирование интеллектуальной деятельности человека. Появился этот раздел информатики 700 лет назад в средневековой Испании [1]. Он оформился в самостоятельную научную область в середине XX в.

Путь искусственного интеллекта был очень извилист и состоял из многократных метаний между чрезвычайным оптимизмом и необоснованным скептицизмом. В наше время искусственный интеллект получил огромное практическое применение [12], открывающее перспективы, без которых немыслимо дальнейшее развитие человечества.

По настоящему развиваться искусственный интеллект как новое научное направление начал в середине XX века. К тому времени уже появилось множество предпосылок его зарождения [11]: философы давно спорили о природе человека и процессе познания мира, нейрофизиологи и

психологи разработали ряд теорий о том как работает человеческий мозг и как происходит процесс мышления, экономисты и математики решали вопросы об оптимальных расчета и о том как представляются знания о мире в формализованном виде; В тоже время, зародилась основа математической теории вычислений – теория алгоритмов – и появились первые компьютеры.

Благодаря методам искусственного интеллекта появилась возможность создавать эффективные компьютерные программы в самых разнообразных, ранее считавшихся недоступными для формализации и алгоритмизации, сферах человеческой деятельности. Например в таких как: медицина [2], биология, зоология, социология, культурология, политология, экономика, бизнес, криминалистика т.п.

Приемы обучения и самообучения искусственного интеллекта, накопления знаний, приёмы обработки нечетких и неконкретных знаний позволили создать программы, о которых ранее человечество даже не могло и подумать. Компьютеры побеждают в шахматы и го [3] чемпионов мира [13], занимаются творческой деятельностью, создают музыкальные и поэтические произведения, распознают образы и сцены, распознают, понимают и обрабатывают человеческую речь, тексты на естественном человеческом языке. Нейрокомпьютеры, созданные по модели человеческого мозга, могут успешно заниматься управлением сложными техническими объектами, диагностируют человеческие заболевания[2], неисправности сложных техническихустройств, занимаются предсказанием погоды и курсов валют, результатов голосований, могут без особого труда выявить хакеров и потенциальных банкротов [1], помогают абитуриентам правильно выбрать специальность на основе их предпочтений и анализируя их оценки и т.д.

1.2 Задачи Искусственного интеллекта

В настоящее время ни у кого не вызывает удивление то, что компьютеры «умнеют» буквально на глазах, а компьютерные программы становятся с каждым днем все интеллектуальнее и интеллектуальнее [14, 15]. Само по себе понятие интеллекта претерпевает постоянные изменения с развитием науки и человечества. Задачи, которые основаны на математических операциях: сложения, умножения, деления - давно уже не считаются интеллектуальными. Также не считается интеллектуальной задача взятия интеграла от дифференциального уравнения, ведь для нее известно много строго детерминированных алгоритмов. Сейчас принято считать интеллектуальными те задачи задачи, которые пока еще в наше время не поддаются алгоритмизации. Это задачи, для решения которых требуются манипуляции с нечеткой, неконкретной, ненадежной, расплывчатой и даже нетрадиционной информацией.

Сама дисциплина по мере развития разделилась на несколько направлений. Одно было посвящено созданию техник отчуждения знаний в виде кода и программ на декларативных (Пролог) и динамических, очень высокого уровня абстракции (Лисп). В рамках этого направления были созданы техники написания экспертных систем, системы автоматического доказательства теорем и так далее.

Второе направление было посвящено автоматическому поиску корреляций в нерегулярных, разреженных данных огромного объёма методами статистики. Это направление широко известно как BigData - “Большие данные” [17].

Третье направление копировало не результаты работы, а само устройство мозга. Про современный уровень понимания работы мозга, его структуры можно посмотреть работы С.А. Шумского [23], осознанно и аккуратно адаптированные для старшей школы. Это одно из самых плодотворных направлений в настоящее время [3, 2, 14, 16]. Как известно из

биологии, мозг человека состоит из большого количества взаимосвязанных нервных клеток — нейронов. Поэтому главной целью нейрокибернетиков является сосредоточение всех сил на разработке элементов, в основе которых будет лежать структура и механизм работы нейронов, и объединении этих элементов в системы, которые принято называть нейросетями и нейрокомпьютерами. Впервые нейросети и нейрокомпьютеры были предложены и созданы американскими учеными В. Мак-Каллоком, В. Питгсом и Ф. Розенблатгом [1] в конце 1950-го года. Это были устройства, которые могли моделировать работу человеческого органа зрения - глаза и его взаимодействие с мозгом. Устройства могли распознавать буквы алфавита, однако были чувствительны к их написанию.

Сегодня нейрокомпьютерные и нейросетевые технологии являются одной из наиболее перспективных и быстроразвивающихся областей в искусственном интеллекте. В двухтысячных годах больших успехов в этой области добились японские исследователи. Ими создан компьютер Ш поколения — нейрокомпьютер, который моделирует структуру мозга и имеет обширную базу [1] знаний. Значительных успехов в этой области добились и российские ученые. Отечественные нейрокомпьютеры уже давно используются в управлении сложными техническими военными объектами. Новый виток этому направлению дают современные специализированные ускорители фирмы NVidia, оптимизированные только для расчёта нейронных сетей, NVIDIA Deep Learning Accelerator инициатива[21] с новой архитектурой ядер Tesla V100, Tensor Core [22].

В отличие от нейрокибернетики, существует еще такое направление как кибернетика «черного ящика», которая не придает значения принципу действия мыслящего устройства - главное, чтобы оно адекватно моделировало его функциональную деятельность. Это направление области искусственного интеллекта направлено на поиски алгоритмов решения

интеллектуальных задач, используя существующие компьютеры, независимо от их аппаратной базы.

Поставив перед собой задачу моделирования функций мозга, ученые столкнулись с серьезной проблемой. Оказалось, что несмотря на многовековую историю исследований ни одна из существующих наук (философия, психология, лингвистика и др.) не смогла предложить сколько-нибудь конкретный алгоритм человеческого мышления. Поэтому кибернетикам пришлось создавать собственные модели мышления.

В конце 50-х гг. XX в. появилась модель лабиринтного поиска [1]. При таком подходе решение задачи искусственного интеллекта выполняется путем перебора огромного количества вариантов, который представляется в виде движения по лабиринту. Создание таких алгоритмов, по словам их критиков, было неразумно, по их словам это было равносильно попытке заново написать все книги, хранящиеся в Британском музее, посадив за пишущие машинки обезьян и надеясь, что обезьяны рано или поздно чисто случайно сумеют напечатать осмысленное слово, фразу или страницу [1]. В настоящее время модель лабиринтного поиска признается тупиковой и имеет ограниченное применение в компьютерных играх. В отдельных разделах этот метод называется поиском в ширину, и без подходящей эвристики [18] число операций, которые требуется выполнить, слишком велико.

В начале 1960-х гг. началась эпоха эвристического программирования. По словам автора этого термина, американского математика Пойа, цель эвристики — исследовать методы и правила, как делать открытия и изобретения. Это очень сложная проблема. Дело в том, что Архимед, выпрыгнувший из ванны с криком «Эврика!», не объяснил, каким образом он догадался, что тело, погруженное в жидкость, теряет в своем весе ровно столько, сколько весит вытесненный им объем воды. Ньютон открыл закон всемирного тяготения, наблюдая за падением яблока. Менделеев пришел к принципу построения периодической таблицы во сне. Поэтов и музыкантов

вдохновляют к творческим поискам возвышенные чувства, разобраться в которых в принципе невозможно.

Чтобы понять механизмы творческого мышления, создатели эвристического подхода провели эксперимент: была отобрана группа студентов, не знакомых с математической логикой и каждый студент должен был доказать самостоятельно одну или несколько теорем из учебника, не заглядывая в него. При этом обязательно нужно было рассуждать вслух, делать любые записи. Прекращать работу можно было только тогда, когда становилось ясно, что выбран неверный путь, в таком случае нужно было начинать все сначала.

После обработки магнитофонных записей и черновиков студентов, программисты нашли эвристики — способы, которыми пользовались студенты, доказывая теоремы. А затем с помощью эвристик была составлена программа, известная под названием «Логик-теоретик» [1], которую принято считать родоначальницей эвристического программирования. И эта программа доказала все теоремы, которые были в учебнике, помимо этого она еще и сформулировала дополнительно те теоремы, которых не хватало до полной логической завершенности курса.

Кроме указанных выше двух подходов к проблеме моделирования процесса мышления и создания искусственного интеллекта существует третий, который называется эволюционным программированием (моделированием). Смысл этого подхода состоит в том, что процесс моделирования человека заменяется моделированием процесса его эволюции.

Серьезный прорыв в практических приложениях искусственного интеллекта произошел в середине 1970-х гг., когда, отказавшись от поисков универсального алгоритма мышления, программисты начали моделировать конкретные знания специалистов и экспертов в разных областях. Так

появилось новое направление искусственного интеллекта, которое впоследствии назвали — экспертные системы [1]. С появлением экспертных систем бизнес в сфере интеллектуальных информационных технологий впервые становится рентабельным.

С середины 1980-х гг. искусственный интеллект — это одно из наиболее привлекательных для инвестирования направлений компьютерной индустрии. Растут ежегодные капиталовложения, создаются промышленные и военные экспертные системы. В качестве альтернативы экспертным системам появляются и успешно завоевывают рынок нейросетевые и нейрокомпьютерные технологии, в которых моделируются процессы, происходящие в мозге: знания транслируются по межнейронным связям, а процесс программирования системы заменяется ее обучением.

Наиболее ярким успехом в этом направлении можно назвать победу программы над чемпионом мира по Го.[3] Без создания аналога интуитивной оценки стоимости позиции, только методами перебора или эвристиками, задача не решаема, т.к. коэффициент ветвления для игры Го в десятки раз больше, чем в шахматах, а объём работы растёт экспоненциально от глубины поиска. Это направление весьма плодотворно развивается, в него инвестируются большие деньги и усилия, создано множество отличных инструментов, выпущенных под свободными лицензиями. В качестве примера можно привести AI Gym[29] - набор API для получения визуального изображения игр Atari [30] с возможностью подать управляющий сигнал на эмулятор контроллера. Компьютер в прямом смысле слова играет, только глядя на экран и не имея никакой модели или эвристики, специфичной для игры. Один и тот же агент может пройти двумерного Марио и трёхмерный Дум лучше, чем человек, если поучится методом проб и ошибок достаточно долго. Интерес настолько велик, что можно найти несколько десятков курсов по машинному обучению от самых престижных ВУЗов мира, включая читаемые Эндрю ЫНом (Adrew Ng), Андреем Карпаты [32] (Andrej Karpathy)

и другими учёными и разработчиками, создающими эту область в настоящее время в прямом смысле этого слова.

Замечание: причину выбора языка Питон можно проиллюстрировать одним примером - после установки Питона, все необходимые зависимости для начала работы с Gym-ом ставятся одной командой

pip install gym

и всё [33]. Большинство библиотек по работе с большими данными написаны на R и на Питоне, с машинным обучением - на Питоне и Си++.

1.3 Направление развития искусственного интеллекта

Сегодня искусственный интеллект — это обширная область исследований и разработок интеллектуальных систем, предназначенных для работы в таких областях человеческой деятельности, которые очень трудно алгоритмизировать и формализовать. Для задач, решаемых методами искусственного интеллекта, характерно наличие большого числа степеней свободы с числом вариантов поиска решений, приближающимся к бесконечности (большинство задач NP-полные). В отличие от жестко детерминированных компьютерных программ, системы искусственного интеллекта сами ищут пути решения поставленной задачи. При этом они могут менять свои параметры и структуру, совершенствоваться и развиваться, жить самостоятельной, независящей от воли разработчика жизнью.

Разработка интеллектуальных систем, основаных на знаниях, до недавнего времени считалась основным и наиболее плодотворным в развитии искусственного интеллекта. Она связана с разработкой модели представления знаний, созданием баз знаний, образующих ядро экспертных систем. Большие надежды возлагались на новые математические модели программирования и языки программирования, такие, как Пролог [1].

Нейросетевые и нейрокомпьютерные технологии. Это направление является альтернативным предыдущему как в идеологическом, так и в практическом плане. Искусственные нейронные сети и нейрокомпьютеры в значительной мере заимствуют принципы работы головного мозга. Знания в них не отделены от процессора, а равномерно распределены и существуют неявно в виде сил синаптических связей. Такие знания не закладываются изначально, а приобретаются в процессе обучения.

Распознавание образов. К распознаванию образов в искусственном интеллекте относят широкий круг проблем: распознавание изображений, символов, текстов, запахов, звуков, шумов. На рынке программных средств имеются системы, основанные на распознавании по признакам, оснащенные базами данных и знаний, имеющих возможность адаптации и обучения. Однако в последнее время становятся популярными гибридные системы, в которых наряду с технологиями экспертных систем используются и нейросетевые технологии.

Заметный прорыв в этой области произошёл в 2012 году, когда сеть AlexNet [24] превзошла предыдущие state-of-the-art сети с заметным отрывом и открыла эпоху CNN сетей в задаче классификации изображений ImageNet [25]. Взрывной рост интереса продолжается до сих пор, изучены интересные вопросы природы обучения в целом к крайне необычных точек зрения (русский перевод и анализ материалов симпозиума 2016г в Стенфорде [26], посвящённого пересечению deep learning и neuroscience сотрудником фирмы [6] С. Козловым дан здесь [27] и здесь [28]). Эти работы интересны по многим причинам: показан процесс возникновения иерархий признаков при росте объёма выборки и числа признаков; хорошо проиллюстрированы огромное количество источников и смещение от академических работ до обсуждений в специфических сообществах и публикаций в arxiv.

Задачи распознавания изображений интересны ещё и тем, что при их решении впервые компьютер сам выделил признаки (features), что и ознаменовало начало эпохи deep learning (глубокого обучения).

Игры и творчество. Традиционно искусственный интеллект включает в себя интеллектуальные задачи, решаемые при игре в шахматы, шашки, го, каллах. В основе этого направления лежит один из ранних подходов это лабиринтная модель плюс эвристики. Кроме того, в современных программах-игроках наиболее полно удалось реализовать центральную идею искусственного интеллекта — обучение, самообучение и самоорганизацию.

1.4 Как обучаются машины

У людей существует два вида запоминания информации, это: механическое заучивание и интеллектуальное осмысление. Заучивание телефонных номеров или инструкций, без сомнения, тоже относится к процессу обучения. Но, как правило, под этим понятием мы подразумеваем кое-что другое.

Ребенок, играющий с друзьями, замечает реакцию других членов группы на свои действия. Этот опыт влияет на его будущее поведение в социуме. Но он не вспоминает и не проигрывает заново свое прошлое, а опирается на прошлый опыт своих взаимодействий с другими членами социальной группы. Взаимодействия самые разные: дети на детской площадке, одноклассники в школе, родители, сестры и братья, друзья, незнакомые люди, взрослые, дети, в помещении или на улице. Оценка новой ситуации базируется на признаках, с которыми ему доводилось сталкиваться раньше. При этом обучение - это уже не просто обычный сбор информации. Формируется то, что можно назвать аналитической оценкой.

Это очень похоже на обучение по картинкам ребенка тому, как отличать собаку от кошки. Показанная картинка кладется в ту или иную стопку, в зависимости от того, правильный был сделан выбор или нет. Со временем растет и эффективность распознавания. Самое интересное тут то, что не нужно специально учить ребенка распознавать собак и кошек. Человеческое сознание обладает встроенными механизмами классификации. Ему требуются только образцы. После работы с образцами, ребенок сможет распознать практически любое изображение кошки или собаки, не говоря уже о реальных животных. Эта способность классифицировать объекты по признакам и обобщать те, которые обладают схожими характеристиками, применяя полученные в процессе тренировок данные к тем, которые только поступили и ранее не встречались, является ключевой особенностью как человеческого, так и машинного обучения.

Разумеется, процесс получения знаний человеком превосходит своей сложностью самые совершенные алгоритмы машинного обучения, но у компьютера есть преимущество в виде большей емкости для запоминания, извлечения и обработки данных [31].

Глава II Машинное обучение

Машинное обучение является подмножеством искусственного интеллекта в области информатики, который часто использует статистические методы, чтобы дать компьютерам возможность «учиться» (то есть постепенно улучшать производительность по конкретной задаче) с данными, не будучи явно запрограммированной. Машинное обучение было придумано в 1959 году Артуром Самуэлем. Оно было основано на изучении теории распознавания образов и теории вычислительного обучения в области искусственного интеллекта, машинное обучение исследует изучение и построение алгоритмов, которые могут учиться и делать прогнозы по данным - такие алгоритмы преодолевают строгие статические программные инструкции, делая предсказания или решения, управляемые данными, путем построения модели из входных данных образца. Машинное обучение используется в ряде вычислительных задач, где проектирование и программирование явных алгоритмов с хорошей производительностью является трудным или неосуществимым; примеры приложений включают фильтрацию электронной почты, обнаружение сетевых злоумышленников или вредоносных инсайдеров, работающих в направлении нарушения данных, оптическое распознавание символов (OCR), обучение ранжированию и компьютерное зрение.

Машинное обучение тесно 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 поле. Машинное обучение иногда сочетается с интеллектуальным анализом данных, где последнее подполе больше фокусируется на анализе разведочных данных и известно как неконтролируемое обучение. Машинное обучение также может быть неконтролируемым и использоваться для изучения и установления

базовых поведенческих профилей для различных объектов, а затем для поиска значимых аномалий.

В области аналитики данных машинное обучение - это метод, используемый 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, повторяемые решения и результаты» и раскрывать «скрытые идеи» посредством изучения исторических отношений и тенденций в данных.

2.1 Задачи машинного обучения

Задачи машинного обучения обычно подразделяются на две широкие категории, в зависимости от наличия обучающего «сигнала» или «обратной связи», доступного для системы обучения: *Контролируемое обучение:* компьютер представлен примерными входными данными и их желаемыми результатами, заданными «учителем», и цель состоит в том, чтобы изучить общее правило, которое отображает входные данные для выходов. В качестве особых случаев входной сигнал может быть только частично доступен или ограничен специальной обратной связью.

Полу-контролируемое обучение: компьютеру предоставляется только неполный обучающий сигнал: тренировочный набор с некоторыми (часто многими) целевыми выходами отсутствует.

Активное обучение: компьютер может получать только учебные ярлыки для ограниченного набора экземпляров (на основе бюджета), а также должен оптимизировать выбор объектов для получения меток. При использовании в интерактивном режиме они могут быть представлены пользователю для маркировки.

Обучение с подкреплением: данные обучения (в виде вознаграждений и наказаний) даются только как обратная связь с действиями программы в

динамической среде, например, вождение транспортного средства или игра против противника.

Неконтролируемое обучение: никакие метки не даются алгоритму обучения, предоставляя его самому себе, чтобы найти структуру во входе.

Неконтролируемое обучение может быть самоцелью (обнаружение скрытых шаблонов в данных) или средством достижения цели (обучения объектам).

2.2 Подходы машинного обучения

Изучение дерева решений

Дерево принятия решений использует дерево решений как предсказательную модель, которая отображает наблюдения относительно позиции на выводы о целевом значении предмета.

Обучение правилам ассоциации

Обучение правилам ассоциации - это метод обнаружения интересных связей между переменными в больших базах данных.

Искусственные нейронные сети

Алгоритм обучения искусственной нейронной сети (ANN), обычно называемый «нейронной сетью» (NN), является алгоритмом обучения, который смутно вдохновлен биологическими нейронными сетями. Вычисления структурированы в терминах взаимосвязанной группы искусственных нейронов, обрабатывая информацию с использованием подхода-подключителя к вычислению. Современные нейронные сети представляют собой нелинейные инструменты моделирования статистических данных. Они обычно используются для моделирования

сложных отношений между входами и выходами, для поиска шаблонов в данных или для сбора статистической структуры в неизвестном совместном распределении вероятностей между наблюдаемыми переменными.

Глубокое обучение

Глубокое обучение (также известное как глубокое структурированное обучение или иерархическое обучение) является частью более широкого семейства методов машинного обучения, основанных на представлении данных обучения, в отличие от алгоритмов, ориентированных на конкретные задачи. Обучение может контролироваться, полуконтролироваться или не контролироваться.

Глубокие системы обучения, такие как глубокие нейронные сети, сети глубоких убеждений и повторяющиеся нейронные сети, были применены к областям, включая компьютерное зрение, распознавание речи, обработку естественного языка, распознавание звука, фильтрацию социальных сетей, машинный перевод, биоинформатику, разработку лекарств и программы настольной игры , где они дали результаты, сопоставимые и в некоторых случаях превосходящие человеческие ожидания.

Индуктивное логическое программирование

Индуктивное логическое программирование (ILP) - это подход к обучению правилам с использованием логического программирования в качестве единого представления для входных примеров, знаний фона и гипотез. Учитывая кодирование известных фоновых знаний и набор примеров, представленных в виде логической базы данных фактов, система ILP выведет гипотезную логическую программу, которая влечет за собой все положительные и отрицательные примеры. Индуктивное программирование является связанной областью, которая рассматривает любые языки программирования для представления гипотез (а не только логического

программирования), таких как функциональные программы.

Поддерживающие векторные машины

Поддержка векторных машин (SVM) - это набор связанных методов контроля, используемых для классификации и регрессии. Учитывая набор примеров обучения, каждый из которых отмечен как принадлежащий к одной из двух категорий, алгоритм обучения SVM создает модель, которая предсказывает, попадает ли новый пример в одну категорию или другую.

Кластеризация

Кластерный анализ представляет набор наборов наблюдений в подмножества (так называемые кластеры), так что наблюдения в пределах одного и того же кластера схожи в соответствии с некоторыми предрасположенными критериями, в то время как наблюдения, проведенные из разных кластеров, неодинаковы. Различные методы кластеризации делают разные предположения о структуре данных, часто определяемые некоторой метрикой подобия и оцениваемые, например, внутренней компактностью (сходство между членами одного и того же кластера) и разделение между различными кластерами. Другие методы основаны на предполагаемой плотности и связности графиков. Кластеризация - это метод неконтролируемого обучения и общий метод анализа статистических данных.

Байесовские сети

Байесовская сеть, сеть убеждений или ориентированная ациклическая графическая модель -это вероятностная графическая модель, представляющая набор случайных величин и их условных независимых величин посредством направленного ациклического графа (DAG). Например, байесовская сеть может представлять вероятностные отношения между заболеваниями и симптомами. При наличии симптомов сеть может использоваться для вычисления вероятностей наличия различных

заболеваний. Существуют эффективные алгоритмы, которые выполняют вывод и обучение.

Обучение с представлением

Несколько алгоритмов обучения, в основном неконтролируемые алгоритмы обучения, направлены на то, чтобы лучше понять представления, предоставленные во время обучения. Классические примеры включают анализ основных компонентов и кластерный анализ. Алгоритмы обучения представлению часто пытаются сохранить информацию на своем входе, но преобразуют ее таким образом, чтобы она была полезной, часто как шаг предварительной обработки перед выполнением классификации или предсказаний, позволяя восстанавливать входы, поступающие от неизвестного источника, генерирующего распределение, в то время как не будучи обязательно верными для конфигураций, которые неправдоподобны при этом распределении.

Алгоритмы обучения в коллекторе пытаются сделать это при ограничении, что изученное представление является малоразмерным. Алгоритмы разреженного кодирования пытаются сделать это при ограничении, что ученое представление разреженное (имеет много нулей). Многолинейные алгоритмы обучения подпространства направлены на изучение низкоразмерных представлений непосредственно из тензорных представлений для многомерных данных без их преобразования в (высокомерные) векторы.

Сходство и метрическое обучение

В этой задаче обучающему устройству даются пары примеров, которые считаются схожими, и пары менее похожих объектов. Затем ему необходимо изучить функцию подобия (или метрическую функцию расстояния), которая может предсказать, будут ли схожие объекты. Это иногда используется в системах рекомендаций.

Машинное обучение, основанное на правилах

Машинное обучение на основе правил является общим термином для любого метода машинного обучения, который идентифицирует, учит или развивает «правила» для хранения, управления или применения знаний. Определяющей характеристикой машинного учащегося агента, основанного на правилах, является идентификация и использование набора реляционных правил, которые в совокупности представляют знания, полученные системой. Это контрастирует с другими машинами, которые обычно идентифицируют уникальную модель, которая может быть универсально применена к любому экземпляру, чтобы сделать предсказание. Основанные на правилах подходы к компьютерному обучению включают системы классификаторов обучения, обучение правилам ассоциации и искусственные иммунные системы.

2. 3 Обучение с подкреплением

Обучение с подкреплением (reinforcement learning, RL) - это область машинного обучения, вдохновленная бихевиористской психологией, касающаяся того, как агенты программного обеспечения должны предпринимать действия в среде, чтобы максимизировать некоторое понятие кумулятивной награды. Проблема, в силу своей общности, изучается во многих других дисциплинах, таких как теория игр, теория управления, исследования операций, теория информации, оптимизация на основе моделирования, многоагентные системы, интеллект роя, статистика и генетические алгоритмы. В литературе по исследованиям и управлению операциями обучение с подкреплением называется приближенным динамическим[19] программированием или нейродинамическим программированием. Проблемы теории обучения с усилением изучены также в теории оптимального управления, которая в основном связана с существованием и характеристикой оптимальных решений и алгоритмов их точного вычисления, и меньше - с учебой или приближением, особенно в

отсутствие математической модели среды. В экономике и теории игр обучение подкрепления может быть использовано для объяснения того, как равновесие может возникать при ограниченной рациональности.

Определение обучения с подкреплением дается не через описание методов обучения, а через выявление свойств, которые характерны для самой задачи обучения.

Идея обучения с подкреплением в том чтобы уловить и зафиксировать аспекты, которые являются наиболее важными для реальной поставленной задачи обучения, имеется в виду организация взаимодействия агента со средой, которое необходимо для достижения некоторой поставленной цели. Ясно, что агент такого рода должен быть способен реагировать на изменения состояния среды, а также выполнять ряд действий, которые могут повлиять на состояние среды. Агент должен обладать одной или несколькими целями, которые в свою очередь должны быть связаны с состоянием среды. При формулировке задачи обучения с подкреплением необходимо учитывать три этих важных аспекта: восприятие среды, действие, направленное на изменение состояния среды и цели, которые связаны со средой.

Обучение с подкреплением отличается от обучения с учителем, которое рассматривается в большинстве современных исследований по машинному обучению, распознаванию статических образов, нейронным сетям. Обучение с учителем – это обучение на основе примеров, которые предъявляются извне. Это важный вид обучения, но без привлечения дополнительных средств он не годится для обучения через взаимодействие агента со средой. В задачах, которые решаются через взаимодействие, очень часто непрактичным является получение примеров, требуемых для поведения агента, которые могли бы являться корректными и представительными для всех ситуаций одновременно, в которых должен действовать агент. В ситуациях, когда обучение более всего нужно, агент должен уметь обучаться основываясь только на своем собственном опыте.

Самая наиболее возникаемая проблема в обучении с подкреплением, которой нет в других видах обучения - это проблема поиска компромисса между изучением и применением. Для того чтобы получить наибольшее вознаграждение, агент, который обучается с подкреплением, должен применять действия, которые он уже применял ранее в прошлой деятельности и которые эффективны с точки зрения получения награды. Однако, для того чтобы обнаружить такие действия, надо чтобы агент также выполнял действия, которые не выполнялись ранее. Агент должен применять те действия, которые ведут к максимальному поощрению, но также и те, которые ранее он не выполнял, для того чтобы в будущем делать лучший выбор. Проблема заключена в том, что нельзя использовать только те действия, которые проверены агентом и ведут к максимальному поощрению или только искать эффективные действия, потому что это зачастую приводит к провалу попыток решения задачи. Агент должен делать попытки принимать разнообразные действия и отдавать предпочтение тем, которые окажутся с максимальным вознаграждением. Для задач стохастического характера каждое действие должно повторяться много раз, чтобы получить надежную оценку ожидаемого поощрения. Эта проблема исследуется математиками уже несколько десятилетий.

Характерной чертой обучения с подкреплением также является то, что в явном виде рассматривается целостная проблема агента, который взаимодействует со средой.

Отличается это тем, что во многих других подходах рассматриваются отдельные подзадачи без указаний, как эти задачи связываются в единую картину. Обучение с подкреплением исходит из целостного взаимодействия со средой. У агентов, которые обучаются по методу обучения с подкреплением присутствуют явно выраженные цели, а также эти агенты способны воспринимать особенности среды и выбирать те действия, которые влияют на эту среду. Обычно с самого начала предполагается, что агент

должен предпринимать какие-то действия, даже если существует высокая неопределенность в среде.

Как известно, агент может обучиться игре в шахматы с помощью контролируемого обучения, в котором ему предъявляются примеры игровых ситуаций наряду с наилучшими ходами для этих ситуаций. Но если нет дружелюбного учителя, предоставляющего готовые примеры, то что может сделать агент? Опробуя случайно выбранные ходы, агент может в конечном итоге составить прогностическую модель своей среды, т.е. предсказать, как будет выглядеть доска после того, как он сделает данный конкретный ход, и даже как, скорее всего, ответит противник в такой ситуации. Но при этом возникает следующая проблема: без какой-либо обратной связи, говорящей о том, какой ход является хорошим и какой плохим, агент не будет иметь оснований для принятия решения о том, какой ход следует сделать. Агент должен знать, что его выигрыш это благоприятный исход, а проигрыш неблагоприятный. Обратная связь такого рода называется вознаграждением, или подкреплением. В играх, подобных шахматам, подкрепление дается только в конце игры.

В других вариантах среды вознаграждения могут поступать более часто. В настольном теннисе как вознаграждение может рассматриваться каждое выигранное очко, а при обучении новобранцев способам перемещения ползком достижением становится каждое движение вперед.

Оптимальной является такая стратегия, которая максимизирует ожидаемое суммарное вознаграждение. Задача обучения с подкреплением состоит в том, чтобы обеспечить использование наблюдаемых вознаграждений для определения в процессе обучения оптимальной (или почти оптимальной) стратегии для данной среды.

Во многих сложных проблемных областях обучение с подкреплением является единственным осуществимым способом, с помощью которого

можно провести обучение некоторой программы, чтобы она могла действовать с высокой производительностью. Например, в случае ведения игр для человекаявляется очень трудной задачей предоставление точных и согласованных оценок большого количества позиций, что требуется для определения в процессе обучения функций оценки непосредственно из примеров. Вместо этого программе можно сообщать, когда она выиграла или проиграла, а сама программа может использовать такую информацию для определения с помощью обучения такой функции оценки, которая предоставляла бы достаточно точные оценки вероятности выигрыша из любой конкретной позиции. Аналогичным образом, чрезвычайно трудно запрограммировать агента так, чтобы он научился вести вертолет; но, предоставляя соответствующие отрицательные вознаграждения за столкновение, болтанку или отклонение от заданного курса, можно дать агенту возможность научиться летать на вертолете самостоятельно. Обучение с подкреплением может рассматриваться как задача, охватывающая всю тематику искусственного интеллекта: агента помещают в какую-то среду и обязывают его обучиться успешно действовать в ней.

По большей части предполагается, что среда является полностью наблюдаемой, поэтому информация о текущем состоянии поступает с результатами каждого восприятия. С другой стороны, считается, что агент не знает, по каким принципам действует среда или какими являются результаты его действий, поэтому допускается наличие вероятностных результатов действий.

В машинном обучении среда обычно формулируется как процесс принятия решения по методу Маркова (MDP), поскольку многие алгоритмы обучения с подкреплением для этого контекста используют методы динамического программирования. Основное различие между классическими методами динамического программирования и алгоритмами обучения усилению заключается в том, что последние не предполагают знания точной

математической модели MDP и нацеливаются на большие MDP, где точные методы становятся неосуществимыми.

Обучение с подкреплением отличается от стандартного контролируемого обучения тем, что правильные пары ввода / вывода не нуждаются в разъяснении, а субоптимальные действия не должны быть явно исправлены. Вместо этого основное внимание уделяется эффективности, которая предполагает поиск баланса между разведкой (неизведанной территории) и эксплуатацией (существующих знаний). Компромисс между разведкой и эксплуатацией был наиболее тщательно изучен в рамках многорукой бандитской проблемы и в конечных MDP [20].

MDP определяется как триплет M = (X, A, P0), где X - счетное непустое множество состояний, A - счетный непустой набор действий. Ядро вероятности перехода P0 присваивает каждой паре состояния (x, a) $\in$ X × A вероятностную меру над X × R, которую мы будем обозначать через P0 (· | x, a). Семантика P0 такова: для U $\subset$ X × R P0 (U | x, a) дает вероятность того, что следующее состояние и связанное с ним вознаграждение принадлежат множеству U, при условии, что текущее состояние x и принятое действие это a.

Ядро вероятности перехода порождает ядро вероятности перехода состояния P, которое для любого (x, a, y) $\in$ X × A × X триплет дает вероятность перехода из состояния x в какое-либо другое состояние y при условии, что действие a было выбрано в состоянии x:

P (x, a, y) = P0 ({y} × R | x, a).

В дополнение к P, P0 также порождает непосредственную функцию вознаграждения r: X × A → R, которая дает ожидаемое немедленное вознаграждение, полученное, когда действие a выбирается в состоянии x: Если (Y (x, a), R (x , a)) ~ P0 (· | x, a), то

$$r(x,a) = \mathbb{E}\left[R_{(x,a)}\right]$$

Вознаграждения ограничены некоторой величиной R> 0: для любого (x, a) ∈ X × A, | R (x, a) | ≤ R почти наверняка. Непосредственно, если случайные вознаграждения ограничены R, тогда krk∞ = sup (x, a) ∈X × A | r (x, a) | ≤ R также выполняется. MDP называется конечным, если оба X и A конечны.

Марковские процессы принятия решений (MDP) - это инструмент для моделирования последовательных проблем принятия решений, когда агент, принимающий решения, взаимодействует с системой последовательно.

Обучение с подкреплением — это обучение тому, что надо делать, как следует отображать ситуации в действия, чтобы максимизировать некоторый сигнал поощрения (вознаграждения), принимающий числовые значения. Обучаемому не говорят, какое действие следует предпринять, как это имеет место в большинстве подходов к обучению машин. Вместо этого он, пробуя выполнять различные действия, должен найти, какие из них принесут ему наибольшее вознаграждение. В наиболее интересных и важных случаях действия могут влиять не только на вознаграждение, получаемое немедленно, но также и на возникающую ситуацию, а через нее — на все последующие поощрения. Эти две характеристики — поиск методом проб и ошибок, а также отсроченные поощрения — представляют собой две наиболее важные отличительные черты обучения с подкреплением.

Определение обучения с подкреплением дается не через описание методов обучения, а путем выявления характерных свойств задачи обучения. Можно сказать, что основная идея состоит просто в том, чтобы уловить и зафиксировать наиболее важные аспекты реальной задачи, имея в виду организацию взаимодействия агента со средой для достижения некоторой цели. Совершенно ясно, что такого рода агент должен иметь возможность в какой-то мере воспринимать состояние среды, а также быть в состоянии предпринимать действия, которые могут повлиять на состояние среды. Агент должен иметь также цель или цели, связанные с состояниями среды.

Формулировка задачи обучения с подкреплением должна учитывать все три аспекта (восприятие, действие, цели) в их наиболее простых формах, но без их вульгаризации.

Обучение с подкреплением отличается от обучения с учителем, которое рассматривается в большинстве современных исследований по обучению машин, статистическому распознаванию образов, искусственным нейронным сетям. Обучение с учителем — это обучение по примерам, предъявляемым некоторой информированной внешней инстанцией. Это важный вид обучения, однако без привлечения дополнительных средств он не пригоден для обучения через взаимодействие. В задачах, решаемых на основе взаимодействия, зачастую непрактично пытаться получать примеры требуемого поведения, которые были бы одновременно корректными и представительными для всех ситуаций, в которых должен действовать агент. На «ничейных» территориях, в ситуациях, когда обучение более всего и нужно, агент должен быть в состоянии учиться только на основе своего собственного опыта.

Одна из наиболее серьезных проблем, возникающих в обучении с подкреплением и отсутствующих в других видах обучения, — это проблема поиска компромисса между изучением и применением). Чтобы получить большее вознаграждение, агент, обучающийся с подкреплением, должен предпочитать действия, которые он уже проверил в прошлой своей деятельности и обнаружил, что они эффективны с точки зрения получения поощрения. Однако, чтобы обнаруживать их, надо пробовать выполнять такие действия, которые еще не выполнялись ранее. Агент должен применять те действия, про которые уже известно, что они позволяют получить вознаграждение, но он должен также и изучать новые действия, чтобы иметь возможность делать лучший выбор в будущем. Проблема состоит в том, что нельзя только использовать уже проверенные действия или только искать новые эффективные действия, поскольку это ведет к провалу попыток

решения задачи. Агент должен пытаться предпринимать разнообразные действия и благоприятствовать тем из них, которые окажутся лучшими. В задачах стохастического характера каждое из действий должно быть повторено многократно, чтобы добиться получения надежной оценки ожидаемого вознаграждения. Дилемма изучения - применения интенсивно исследуется математиками вот уже в течение нескольких десятилетий. Пока же просто отметим, что в области обучения с учителем, в том виде, как эту область принято обычно определять, проблема равновесия между изучением и применением в полном объеме даже и не возникает.

Еще одна характерная черта обучения с подкреплением состоит в том, что в явном виде рассматривается целостная проблема целенаправленного агента, взаимодействующего с неопределенной средой). Это существенно отличается от многих других подходов, где рассмотрение проводится на уровне подзадач без всяких указаний на то, как эти подзадачи можно увязать в общую картину. Например, уже упоминалось, что в большинстве исследований в области обучения машин внимание фокусируется на обучении с учителем, причем явно никак не указывается, как такого рода возможность можно использовать на практике для получения каких-либо полезных результатов. В других исследованиях разрабатываются теории планирования, оперирующего с общими целями, но без рассмотрения роли планирования в процессах принятия решения, идущих в реальном масштабе времени, или же вопроса о том, откуда взять предсказывающие модели, которые необходимы для планирования. Такой подход позволил получить много важных результатов, однако концентрация внимания в нем на изолированных подзадачах существенно ограничивает возможности его применения.

Обучение с подкреплением ставит прямо противоположную задачу и исходит из целостного, взаимодействующего со средой, целенаправленного агента. Все агенты, обучающиеся с подкреплением, имеют явным образом

выраженные цели, могут воспринимать особенности среды, а также выбирать действия, влияющие на эту среду. Более того, обычно с самого начала предполагается, что агент должен действовать, несмотря на существенную неопределенность в среде. Когда в обучение с подкреплением вовлекается планирование, приходится предпринимать усилия для обеспечения взаимодействия между планированием и выбором действий в реальном масштабе времени, а также для поиска способов получения и улучшения моделей среды. Рассмотрение специфичных для данной задачи соображений относительно того, какие характеристики критичны, а какие нет, может привести к включению в обучение с подкреплением средств обучения с учителем. Чтобы добиться успеха в области обучения, надо выделить и изучить важные подзадачи общей задачи обучения, но это должны быть такие подзадачи, роль которых ясна с точки зрения целостного, взаимодействующего со средой, целенаправленного агента, даже если нет возможности во всех подробностях описать этого агента.

2.4 Типы агентов

Агент, действующий с учетом полезности, определяет с помощью обучения функцию полезности состояний и использует ее для выбора действий, которые максимизируют ожидаемую полезность результата.

- Агент, действующий по принципу Q-обучения, определяет с помощью обучения функцию ‘‘действие-значение’’, или Q-функцию, получая сведения об ожидаемой полезности выполнения данного конкретного действия в данном конкретном состоянии.
- Рефлексный агент определяет с помощью обучения стратегию, которая непосредственно отображает состояния в действия. Агент, действующий с учетом полезности, для принятия решений должен также иметь модель среды, поскольку он должен знать, в какие состояния приведут его выполненные им действия. Например, для того

чтобы программа игры в нарды могла использовать функцию оценки для нард, она должна иметь информацию о том, каковыми являются допустимые ходы и как они влияют на позицию в игре. Это единственный способ, позволяющий применить функцию полезности к результирующим состояниям. Агент, действующий по принципу Q-обучения, с другой стороны, может сравнивать значения, характеризующие доступные ему варианты действий, без необходимости знать их результаты, поэтому ему не требуется модель среды. Тем не менее агенты, действующие по принципу Q-обучения, не могут прогнозировать будущую ситуацию, поскольку не имеют информации о том, к чему приведут их действия; это может серьезно ограничить способность таких агентов к обучению.

Исследования в области обучения с подкреплением можно считать частью общего процесса, развивающегося в последние годы. Он состоит во все расширяющихся контактах и взаимодействиях между искусственным интеллектом и другими инженерными дисциплинами. Еще совсем недавно искусственный интеллект рассматривался как область исследований, никак не связанная с такими областями, как теория управления и статистика. Искусственный интеллект имел дело с логикой и символами, не с числами. Его моделями были большие программы на языке LISP, а не дифференциальные уравнения, соотношения линейной алгебры и статистики. В последние десятилетия эта точка зрения постепенно размывалась. Современные исследователи в области искусственного интеллекта допускают использование моделей статистики и алгоритмов управления, например, в качестве вполне подходящих конкурирующих методов или же просто как часть арсенала методов анализа. Области, которым ранее не придавали значения, находящиеся на стыке искусственного интеллекта и традиционных инженерных методов, сейчас развиваются наиболее активно. В их число входят такие новые направления, как нейронные сети,

интеллектуальное управление, а также предмет нашего рассмотрения — с подкреплением. В обучении с подкреплением развиваются идеи, почерпнутые в теории оптимального управления и в стохастической аппроксимации, следуя более общим и более амбициозным целям искусственного интеллекта [34].

2.5 Алгоритм обучения с подкреплением

Обучение с подкреплением включает в себя агент, набор состояний S и набор действий для каждого состояния A. Выполняя действие a $\in$ A, агент переходит из старого состояния в новое состояние. Выполнение действия в определенном состоянии предоставляет агенту вознаграждение (численный балл). Целью агента является максимизация его общей (будущей) награды. Он делает это, добавляя максимальную награду, достижимую из будущего состояния, в награду в ее текущем состоянии, эффективно влияя на текущее действие потенциальной наградой в будущем. Эта награда представляет собой взвешенную сумму ожидаемых значений вознаграждений всех будущих шагов, начиная с текущего состояния.

Рассчитывается функция полезности по формуле:

$Q(s, a) \leftarrow R(s, a , s') + \gamma \cdot \max Q(s', a')$

Здесь R(s, a, s') - это награда при действии a из состояния s в состояние s', γ – коэффициент диспропорциональности, max Q(s', a') - максимальная выгода по всем возможным действиям действии a' из состояния s'

Глава III Элективный курс «Машинное обучение»

3.1 Рабочая программа элективного курса по машинному обучению

Пояснительная записка

Типология курса: предметный.

В России такое направление как машинное обучение еще только начинает развиваться. В вузах только начинают апробировать ввод курсов по машинному обучению, например, в Новосибирском Государственном университете на кафедре АФТИ в этом году провели курс по нейросетям и машинному обучению[7]. В Красноярском государственном педагогическом университете имени В. П. Астафьева разработан элективный курс для включения машинного обучения и искусственного интеллекта в целом в профильный курс школьной программы. Учитывая растущий спрос на специалистов в этой области, курс востребован [5].

Для того, чтобы успевать за новыми технологиями, школьная программа содержит вариативный и дополнительный компоненты, призванный скомпенсировать непрерывную модернизации в сфере информационных технологий. Он как раз и предназначен вследствие острой необходимости в дополнительных курсах, которые будут направлены на устранение таких пробелов в стандартном курсе информатики, чтобы обучающиеся были более подготовлены к реальным требованиям работодателей на рынке информационных технологий и дальнейшему освоению современных разделов информатики в вузах.

Курс использует язык Питон, поскольку уже сформировалась достаточная методическая база по его преподаванию в старшей школе, и уже идут работы по введению языка в среднюю школу [4] Сам язык - один из факторов успешного развития области, поскольку соединяет необычайную чистоту исходных кодов с удобством использования мощных библиотек для выполнения ресурсоёмких задач.

Ожидаемый результат.

В результате освоения курса учащийся должен: уметь работать с алгоритмом машинного обучения с подкреплением посредством работы в среде программирования Python, знать основы алгоритма машинного обучения с подкреплением, уметь использовать библиотеку pygame, уметь анализировать свою деятельность и применять на практике полученные знания.

Цели курса

Основная цель:

Подготовка учащихся инженерно-технического профиля по направлению Машинное обучение.

Образовательные

Получение представления о машинном обучении и обучении с подкреплением.

Освоение технологии машинного обучения с подкреплением.

Формирование умений машинное обучение с подкреплением в будущей профессиональной деятельности.

Развивающие

Развитие способностей аналитического мышления.

Развитие представлений о новом подходе к обработке информации.

Развитие творческих способностей в процессе моделирования систем искусственного интеллекта в основе которых лежит машинное обучение с подкреплением с алгоритмом Q-Learning.

Воспитательные

Воспитание ответственности за результаты своей работы в коллективе, адекватная оценка своего вклада в общее дело.

Воспитание отношения к информационной культуре и информационной безопасности.

Воспитание информационной культуры.

Задачами данной программы являются:

Представление о машинном обучении, задачах, решаемых с его помощью.

Изучение основ программирования на языке Python.

Изучение алгоритма машинного обучения с подкреплением Q-Learning

Практическое применение алгоритма машинного обучения с подкреплением — Q-Learning

Результатом реализации данной программы является

Код, реализующий работу алгоритма машинного обучения Q-Learning

Содержание программы

В программу курса входить 32 академических часа:

1 вводное задание

Что такое машинное обучение, где применяется машинное обучение, почему оно востребовано. Какие задачи можно решать с помощью машинного обучения, какие основные подходы машинного обучения существуют.

12 занятий по основам языка программирования python

Типы данных, циклы, условные операторы, функции, объектно-ориентированное программирование.

13 занятий по работе с библиотекой pygame
Установка библиотеки, структура программы, объекты, поверхности, изображения, обработка событий клавиатуры и мыши, обработка коллизий объектов.

2 занятия на теоретические основы алгоритма Q-Learning

Алгоритм машинного обучения с подкреплением Q-Learning, особенности, как реализуется.

4 занятия на практическую часть по Q-Learning алгоритму.

Моделирование алгоритма Q-Learning посредством языка python и библиотеки pygame.

Учебно-тематическое планирование

Тема урока	**Рассматриваемые вопросы**	**Количество часов**
Введение в машинное обучение	Машинное обучение	1
Основы языка программирования Python	Типы данных в Python	3
Основы языка программирования Python	Циклы и условия в Python	2
Основы языка программирования Python	Функции в Python	3
Основы языка программирования Python	Классы в Python	3
Библиотека Pygame	Введение в Pygame, установка, структура программы	3
Библиотека Pygame	Объекты в Pygame	3
Библиотека Pygame	Surfaces and rects	2
Библиотека Pygame	Обработка событий клавиатуры и мыши	2
Библиотека Pygame	Коллизия объектов	2
Библиотека Pygame	Изображения	2
Машинное обучение	Q-Learning введение	2
Машинное обучение	Q-Learning практика	4

Рекомендации к проведению уроков

Следует поделить учащихся на группы в несколько человек.

Проверять записи в тетрадях, следить чтобы на практических занятиях все учащиеся выполнили задания.

Контролировать процесс усвоения информации на протяжении всего курса.

После каждого занятия давать домашнее задание на дом на закрепление.

Критерии оценивания

В конце курса, все учащиеся должны завершить написание своего алгоритма машинного обучения с подкреплением посредством использования таких инструментов как язык программирования Python и библиотека Pygame. Учитель, который ведет курс может оценивать работы учащихся по следующим критериям:

Элементы оценивания	**Критерии оценивания**
Алгоритм машинного обучения с подкреплением Q-Learning	точность алгоритма полнота алгоритма наглядность алгоритма комментирование кода алгоритма понимание алгоритма

Заключение

В настоящее время, машинное обучение становится все более востребованным на рынке информационных технологий и набирает популярность, так как расширяется область применения алгоритмов машинного обучения. Однако в России ощущается острая нехватка квалифицированных кадров. Причин этому несколько. Первая - это отсутствие фундамента для подготовки соответствующих специалистов. Так как область новая, преподавать машинное обучение попросту некому, нужно переквалифицировать учителей информатики, а это требует введения новых гостов и стандартов, что в свою очередь займет много времени. Вторая причина - это отставание Российского образования от современных тенденций в информационных технологиях, вызванная разрывом многих технологических и образовательных цепочек в ходе перестройки 1990-х годов, прервавшей цепочку преемственности во многих научных школах. В школах до сих пор учат по устаревшей программе. Преподают старые языки программирования, которые уже не имеют практического применения, ими никто не пользуется вне школьной или университетской программы. В школьной программе нет такой области как искусственный интеллект.

На мой взгляд, включение машинного обучения в образовательную программу положительно отразится на всем процессе обучения информатики и принесет большой вклад в развитие российского информационного сообщества в целом. Новые школы уже существуют, многие инициативы запущены, языки и подходы к работе эволюционируют в сторону простоты и удобства. Проведённый обзор литературы показал существование большого количества наработок, которые готовы и/или даже адаптированы для внедрения в старшую школу. В этих условиях успешное выполнение учебного проекта с реализацией алгоритма самообучения, выполненного из первых принципов, сможет внести ясность в саму методологию

искусственного интеллекта, типичные сложности при написании и использовании программы.

Остались открытыми крайне интересные вопросы использования представленных наработок в школьной робототехнике и многих других областях, которые, безусловно, будут сделаны.

Библиографический список

1. Л.Н. Ясницкий. Введение в искусственный интеллект 2-е издание 2005г
2. [Электронный источник] https://www.ibm.com/watson/
3. [Электронный источник] https://deepmind.com/research/aplhago
4. [Электронный источник] https://yandexlyceum.ru
5. [Электронный источник] http://atlas100.ru/
6. [Электронный источник] https://www.instrumental.com/about
7. [Электронный источник] https://habr.com/post/414165/
8. [Электронный источник] http://nti-contest.ru/publications/
9. Гусаковский, Михаил Антонович. "Метод проектов. Научно-методический сборник. Выпуск 2. Мн.: РИВШ БГУ, 2003. 240 с." (2003).
10. [Электронный источник] https://www.nature.com/articles/nature14422
11. Масленникова О.Е., Попова И.В. Основы искусственного интеллекта
12. [Электронный источник] https://robo-sapiens.ru/stati/oblasti-primeneniya-iskusstvennogo-intellekta/
13. [Электронный источник] http://www.kasparov.com/timeline-event/deep-blue/
14. [Электронный источник] https://www.apple.com/ios/siri/
15. [Электронный источник] https://assistant.google.com/
16. [Электронный источник] https://alice.yandex.ru/
17. {Big Data} Introduction to Big Data Xiaomeng Su, Institutt for informatikk og e-læring ved NTNU course IINI3012 Big Data
18. [Электронный источник] https://www.google.com/url?q=http://ai.berkeley.edu/slides/Lecture%25202%2520--%2520Uninformed%2520Search/SP14%2520CS188%2520Lecture%25202%2520--%2520Uninformed%2520Search.pptx&sa=D&ust=1530216328277000&usg=AFQjCNGpOECDXTsKoDABGcf2Re80Lv_6YA

19. [Электронный источник] https://en.wikipedia.org/wiki/Reinforcement_learning
20. [Электронный источник] http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MDP.pdf
21. [Электронный источник] http://nvdla.org/
22. [Электронный источник] https://www.nvidia.com/en-us/data-center/tensorcore/
23. [Электронный источник] http://tube.sfu-kras.ru/video/1965
24. [Электронный источник] https://www.google.com/url?q=https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf&sa=D&ust=1530217411698000&usg=AFQjCNEKdsIuU7DGvgdwIgS0B2EJI-mMIQ
25. [Электронный источник] https://www.google.com/url?q=http://www.image-net.org/&sa=D&ust=1530217489064000&usg=AFQjCNG4Idgs7cj3pvZ1pmkNrnEIKsl16A
26. [Электронный источник] https://www.google.com/url?q=https://neuroscience.stanford.edu/events/mbc-symposium-deep-learning-fundamental-progress-brain-representations-and-semantic-learning&sa=D&ust=1530217990257000&usg=AFQjCNFPfsOWaqbySjaD0H7BPowYDqT-zQ
27. [Электронный источник] https://habr.com/post/282277
28. [Электронный источник] https://habr.com/post/282826
29. [Электронный источник] https://gym.openai.com/
30. [Электронный источник] https://gym.openai.com/envs/#atari
31. Х.Брингс. Машинное обучение 2017г.
32. [Электронный источник] http://karpathy.github.io/
33. [Электронный источник] https://gym.openai.com/docs
34. П.Норвиг. Искусственный интеллект, современный подход. 2-е изд. 2007г.

Приложение 1. Код эмулятора механического робота «Crawler».

```
    """
    The Crawler robot from cs188 course from Berkeley University.
    """

    import os
    import logging

    from math import cos, sin, atan2, pi, sqrt

    import pygame
    import numpy as np

    from pygame import (QUIT, KEYDOWN,
                        K_ESCAPE, K_LEFT, K_RIGHT, K_UP, K_DOWN, K_LSHIFT)


    SCREEN_RECT = pygame.Rect(0, 0, 640, 480)
    MAIN_DIR = os.path.split(os.path.abspath(__file__))[0]


    def rotate(X, Y, x, y, angle):
        """Rotate vector `(X, Y)` around point `(x, y)` by `angle`."""
        new_x = x + (X - x) * cos(angle) + (Y - y) * sin(angle)
        new_y = y - (X - x) * sin(angle) + (Y - y) * cos(angle)
        return new_x, new_y


    def clip(x, min_x, max_x):
        return min(max_x, max(x, min_x))


    class Board:
        def __init__(self, min_p, max_p, Np, min_q, max_q, Nq):
            self.min_p = min_p
            self.max_p = max_p
            self.min_q = min_q
            self.max_q = max_q
            self.Np = Np
            self.Nq = Nq
            self.dp = (max_p - min_p)/Np
            self.dq = (max_q - min_q)/Nq

            self.up = np.zeros((Np + 1, Nq + 1), np.float)
            self.down = np.zeros((Np + 1, Nq + 1), np.float)
            self.left = np.zeros((Np + 1, Nq + 1), np.float)
            self.right = np.zeros((Np + 1, Nq + 1), np.float)

            self.i = Np//2
```

```
            self.j = Nq//2

        def draw(self, screen, xmin, xmax, ymin, ymax):
            ampl = max(0.01,
                       np.max(np.max(np.abs(self.up))),
                       np.max(np.max(np.abs(self.down))),
                       np.max(np.max(np.abs(self.left))),
                       np.max(np.max(np.abs(self.right))))

            def draw(val, poly):
                     color = (val < 0) and [-255*val/ampl, 0, 0] or [0,
255*val/ampl, 0]
                pygame.draw.polygon(screen, color, poly)

            dx = (xmax - xmin)/self.Np
            dy = (ymax - ymin)/self.Nq
            for i in range(self.Np + 1):
                x = xmin + (i + 0.5)*(dx + 1)
                xl, xr = x - dx/2, x + dx/2
                for j in range(self.Nq + 1):
                    y = ymin + (j + 0.5)*(dy + 1)
                    yt, yb = y + dy/2, y - dy/2
                    draw(self.left[i, j],  [[x, y], [xl, yb], [xl, yt]])
                    draw(self.right[i, j], [[x, y], [xr, yt], [xr, yb]])
                    draw(self.up[i, j],    [[x, y], [xl, yb], [xr, yb]])
                    draw(self.down[i, j],  [[x, y], [xl, yt], [xr, yt]])

        def go_up(self):
            """Returns `(array_to_update, i0, j0)`."""
            old_j, self.j = self.j, clip(self.j - 1, 0, self.Nq)
            return self.up, (self.i, old_j)

        def go_down(self):
            """Returns `(array_to_update, i0, j0)`."""
            old_j, self.j = self.j, clip(self.j + 1, 0, self.Nq)
            return self.down, (self.i, old_j)

        def go_left(self):
            """Returns `(array_to_update, i0, j0)`."""
            old_i, self.i = self.i, clip(self.i - 1, 0, self.Np)
            return self.left, (old_i, self.j)

        def go_right(self):
            """Returns `(array_to_update, i0, j0)`."""
            old_i, self.i = self.i, clip(self.i + 1, 0, self.Np)
            return self.right, (old_i, self.j)

        def coordinates(self):
            return self.i*self.dp + self.min_p, \
                   self.j*self.dq + self.min_q
```

```
        class State:
            """
            Const State of the Crawler (all angles, geometry, collision points).

            Global constants [https://stackoverflow.com/a/23274028].
            """
            __need_warning = True

            def __init__(self, theta, phi, crawler):
                self.theta = theta
                self.phi = phi
                self.crawler = crawler

                crawler, phi, theta = self.crawler, self.phi, self.theta

                x0, y0 = crawler.width, crawler.height
                x1 = x0 + crawler.L0*cos(phi)
                y1 = y0 + crawler.L0*sin(phi)
                x2 = x1 + crawler.L1*cos(phi + theta)
                y2 = y1 + crawler.L1*sin(phi + theta)

                # TODO: `atan2` is monotonic, `atan2(y, x)==0 <=> y==0` => I can
use
                #       `y/x` instead of `atan2(y, x)` to save CPU.
                angles = [0, atan2(y1, x1), atan2(y2, x2)]
                x = [0, x1, x2]
                y = [0, y1, y2]
                self.min_angle = min(angles)
                self.joint = angles.index(self.min_angle)
                self.distances = [sqrt(x ** 2 + y ** 2) for x, y in zip(x, y)]

            def compute_geometry(self):
                bot, phi, theta = self.crawler, self.phi, self.theta
                x0, y0 = bot.width, bot.height
                x1 = x0 + bot.L0*cos(phi)
                y1 = y0 + bot.L0*sin(phi)
                x2 = x1 + bot.L1*cos(phi + theta)
                y2 = y1 + bot.L1*sin(phi + theta)
                 return bot.x, bot.y, bot.width, bot.height, x0, y0, x1, y1, x2,
y2

            def T(self, dTheta, dPhi):
                """Pure transition from the State to another State."""
                theta = clip(self.theta + dTheta, -0.9*pi, 0.9*pi)
                phi = clip(self.phi + dPhi, -0.4*pi, 0.9*pi)
                if State.__need_warning:
                    State.__need_warning = False
                      logging.warning('do arm-body collisions handling here (or
treat it '
```

```
                                        'as the ground using another `State` as
well)')
                return State(theta, phi, self.crawler)

            def __setattr__(self, key, value):
                if key in self.__dict__:
                    raise TypeError('State is read-only')
                        assert key in "phi theta crawler joint min_angle
distances".split(), \
                    "unknown field: " + key
                self.__dict__[key] = value

            def __str__(self):
                # This is the state of the Crawler, after all.
                return "Crawler(theta=%.3f, phi=%.3f)" % (self.theta, self.phi)

            __unicode__ = __repr__ = __str__


        class MrClaw:
            """
                    The "Crawler" from Berkeley CS188x AI course
[https://goo.gl/UK6YaQ].

             Helper class State manages collisions of the arm with the ground
(arm can hit the ground
             in the middle point of the update and I want to keep the physical
model accuracy).
            """

            def __init__(self, phi=0, theta=0, x=0, y=0, width=100, height=50,
                         L0=50, L1=70):
                self.x = x
                self.y = y
                self.width = width
                self.height = height
                self.L0 = L0
                self.L1 = L1
                self.state = State(theta, phi, self)

            def update(self, new_theta, new_phi):
                """Compute new state and return the reward."""
                dTheta = new_theta - self.state.theta
                dPhi = new_phi - self.state.phi

                L = [self.state, self.state.T(dTheta, dPhi)]

                def partition(L, indx=0):
                    """
                    Recursive search of special points.
```

```
                First pass - binary subdivision to make the step smaller
than 1/8th of the sin-period.
                Second pass - find roots of any of three equations: α₁(θ,ϕ)
= 0 || α₂(θ,ϕ) = 0 || α₁(θ,ϕ) = α₂(θ,ϕ),
                where θ = θ₀ + t·Δθ,
                      ϕ = ϕ₀ + t·Δϕ.
                """
                S0, S1 = L[indx:indx+2]
                if abs(S0.theta - S1.theta) + abs(S0.phi - S1.phi) > 0.1:
                          S = State((S0.theta + S1.theta)/2.0, (S0.phi +
S1.phi)/2.0, S0.crawler)
                    L.insert(indx+1, S)
                      partition(L, indx+1)     # After partitioning L might
grow, thus I do
                    partition(L, indx)      # the upper half first.
            partition(L)

            reward = 0
            for A, B in zip(L[:-1], L[1:]):
                middle = State((A.theta + B.theta)/2, (A.phi + B.phi)/2,
A.crawler)
                                    reward   -=   B.distances[middle.joint]   -
A.distances[middle.joint]
            self.x += reward

            self.state = L[-1]
            return reward

        def draw(self, screen):
            """Drawing has an inner state (the animation)."""
                        x,   y,   w,   h,   x0,   y0,   x1,   y1,   x2,   y2   =
self.state.compute_geometry()
            lines = (
                (x,      y,      x + w,  y,      (0, 255, 0)),
                (x + w,  y,      x + w,  y + h,  (0, 255, 0)),
                (x + w,  y + h,  x,      y + h,  (0, 255, 0)),
                (x,      y + h,  x,      y,      (0, 255, 0)),
                (x + x0, y + y0, x + x1, y + y1, (0, 0, 255)),
                (x + x1, y + y1, x + x2, y + y2, (0, 0, 255)),
            )

            angle = self.state.min_angle

                        def  line(x0,  y0,  x1,  y1,  color,  angle=angle,
line=pygame.draw.line,
                     ymax=SCREEN_RECT.height):
                x0, y0 = rotate(x0, y0, x, y, angle)
                x1, y1 = rotate(x1, y1, x, y, angle)
                # Change direction of Y-axis to "upward" instead of the SDL
one.
                line(screen, color, (x0, ymax - y0), (x1, ymax - y1), 2)
```

```
            for l in lines:
                line(*l)

        def move(self, dx, dy):
            self.x += dx
            self.y += dy

        def turn(self, dphi, dtheta):
            self.update(dtheta, dphi)


    def load_image(file):
        """Load an image and prepare it for play."""
        file = os.path.join(MAIN_DIR, 'data', file)
        try:
            surface = pygame.image.load(file)
        except pygame.error:
            raise SystemExit('Could not load image "%s" %s' % (file,
                                                               pygame.get_er
ror()))
        return surface.convert()


    def main():
        pygame.init()

        gamma = 0.9
        Q = Board(-pi, pi, 30,
                  -pi, pi, 30)
        R = Board(-pi, pi, 30,
                  -pi, pi, 30)

        # Set the display mode (uncomment to get a fullscreen).
        win_style = 0  # | pygame.FULLSCREEN
        best_depth = pygame.display.mode_ok(SCREEN_RECT.size, win_style, 32)
            screen  =  pygame.display.set_mode(SCREEN_RECT.size,  win_style,
best_depth)

        # Create the background by tiling the `background.gif` image.
        bg_tile = load_image('background.gif')
        background = pygame.Surface(SCREEN_RECT.size)
        for x in range(0, SCREEN_RECT.width, bg_tile.get_width()):
            background.blit(bg_tile, (x, 0))
        screen.blit(background, (0, 0))
        pygame.display.flip()

        crawler = MrClaw(x=100, y=100, L0=90, L1=110)
        clock = pygame.time.Clock()

        is_alive = True
```

```
        while is_alive:
            # Get input.
            for e in pygame.event.get():
                    if e.type == QUIT or (e.type == KEYDOWN and e.key ==
K_ESCAPE):
                    is_alive = False

            screen.blit(background, (0, 0))

            Q.draw(screen, 400, 600, 10, 400)
            R.draw(screen, 100, 300, 10, 400)
            crawler.draw(screen)

            pygame.display.flip()

            # Handle player input.
            is_pressed = pygame.key.get_pressed()
            if is_pressed[K_LSHIFT]:
                if is_pressed[K_RIGHT]:   crawler.move(+5, 0)
                if is_pressed[K_LEFT]:    crawler.move(-5, 0)
                if is_pressed[K_UP]:      crawler.move(0, +5)
                if is_pressed[K_DOWN]:    crawler.move(0, -5)
            else:
                direction = None
                if is_pressed[K_RIGHT]:   direction = "right"
                if is_pressed[K_LEFT]:    direction = "left"
                if is_pressed[K_UP]:      direction = "up"
                if is_pressed[K_DOWN]:    direction = "down"
                if direction is not None:
                    arr, old_pos = getattr(Q, "go_" + direction)()
                    new_pos = Q.i, Q.j
                    # TODO: improve that array boundary clipping (a kludge).
                    if old_pos != new_pos:
                        reward = crawler.update(*Q.coordinates())
                        # arr[old_pos] += reward
                        arr[old_pos] = reward + gamma*max(Q.up[new_pos],
                                                          Q.down[new_pos],
                                                          Q.left[new_pos],
                                                          Q.right[new_pos])
                        getattr(R, direction)[old_pos] = reward

            # Limit the framerate to 60 FPS.
            clock.tick(60)

        pygame.quit()


    if __name__ == '__main__':
        main()
```