Бейсов подход в Дълбокото Подсилено Обучение Бейсов подход в Дълбокото Машинно Обучение

Firstname Surname

A thesis presented for the degree of
Doctor of Philosophy

Supervised by:
Professor Louis Fage
Captain J. Y. Cousteau

University College London, UK
January 2015

I, AUTHORNAME confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Абстракт

Целта на настоящата работа е да създаде математически модел на самообучаващ се агент и да реализира агент, който напълно изследва дадена среда на Графичен Потребителски Интерфейс (ГПИ). Като входни данни за модела се използват изображения от ГПИ и информация за неговата сегментация. Агентът може да извършва действия, променяйки състоянието на ГПИ средата. Адекватността на извършените действия се оценява с награда, която средата предоставя. Наградата се определя от различни фактори, един от които е процентът покрит нов програмен код. Агентът успешно е изследвал средата, когато покритието на код на средата е пълно.

На базата на този модел ще създадем агент, който ще намира грешки в програмни продукти, ще синтезира пакети от автоматични тестове за качеството на софтуера и ще изпълнява посочени от потребителя задачи.

Благодарности

Съдържание

Списък на фигурите

List of tables

Речник

API	**A**pplication **P**rogramming **I**nterface
JSON	**J**ava**S**cript **O**bject **N**otation

Глава 1

Увод

1.1 Мотивация

Една от основните цели на Изкуствения Интелект (ИИ) е да създаде агенти, които разбират и взаимодействат със света около нас. Значителен прогрес в тази насока беше постигнат през последните години благодарение на развитието на изчислителната техника (графични ускорители), наличието на голямо количество данни, нови начини за събиране и съхранението им и нови алгоритми. Бързият напредък в сферата на подсиленото обучение доведе до разработката на интелигентни агенти, взаимодействащи с все по-сложни среди (Mnih et al. 2015), (Silver & Hassabis 2016), (Levine et al. 2016) и (Silver et al. 2017). Критични за това са обучаващите алгоритми, техники за скалирането им и симулационни среди, които предоставят начини за оценка и сравняване на различни агенти (Bellemare et al. 2013), (Todorov et al. 2012) и (Johansson et al. 2016).

Хората се справят лесно с редица задачи, които изискват комплексно разбиране на визуалния свят, разпознаване на различни обекти в него и взаимодействие с тях. Например (екран от ГПИ среда и разпознаване на обекти в него). Би било лесно за човек да изучи подобна визуална среда.

Агенти, действащи в симулирани среди, са фундаментално ограничени - те никога не се сблъскват със сложността на реалния свят, поради което не могат да използват семантично знание и достигнат интелигентност. В роботиката аген-

тите действат в реална среда, но процесът на обучение е бавен и скъп, дори и за тясно дефинирани задачи (Levine et al. 2016).

За справянето с тези проблеми могат да се използват среди, базирани на ГПИ приложения (Shi et al. 2017). Те предоставят разнообразни задачи, възможност за бързо итериране и обучение. Агентите получават същите сензорни данни, които получава човек, взаимодействащ с тези среди. Те предоставят възможност за изграждане на знание, невъзможно за придобиване в симулации.

Предизвикателства Тъй като подобни възможности изглеждат естествени за човек, може да забравим колко трудни са те за един агент. Изображенията са представени като голям масив от числа, които представят яркостта за всяка позиция. Едно такова изображение може да съдържа милиони такива *пиксели*, които агентът трябва да трансформира до семантични концепции на високо ниво, като например “текст” или “бутон”. При това, различни форми и цветове на даден бутон, също трябва да се класифицират като такъв, независимо от възможността за наличие на напълно различни шаблони (patterns) в яркостта на пикселите.

Концептуалното разбиране на дадено изображение е само първата стъпка за създаването на подобни агенти. Основна задача е създаване на модел на агент, който избира действия, които до доведат да постигането на поставена задача. Трудността тук се изразява в липсата на пълна информация (fully observed) за средата в която агента действа. Например, натискането на един и същи бутон в различни състояния на средата, може да доведе до наблюдаването на две напълно различни състояния на средата. Това означава, че е необходимо знание за конкретното състояние на средата.

Агентът няма предварителен модел на средата, която изучава. Той я “опознава”, чрез опити и грешки, като се опитва да приложи различни комбинации от действия в дадено състояние, за да постигне оптимална награда. На всяка стъпка даден агент трябва да избере дали да използва вече наученото или да избере действие, което не е изпълнил в конкретното състояние. Тази дилема се нарича: компромис на изучаване и използване на наученото (exploration exploitation tradeoff) и е основна задача за решаване от всеки агент. ГПИ средите могат да предоставят голям брой действия за дадено състояние (например меню системата на Microsoft Word), което прави пълното им изучаване неприложимо в

кратки интервали от време.

Обнадеждаващ прогрес Въпреки сложността на задачата, през последните години се наблюдава значителен прогрес в областта на подсиленото обучение. По конкретно, развитието на Изкуствените Невронни Мрежи (ИНМ) (Artificial Neural Networks) и методи за създаване на Бейсови модели с милиони параметри доведоха до значително разширение на областите в които подсиленото обучение е приложимо. Алгоритми като Дълбоко Q-обучение (Deep Q-learning) допринасят за създаване на агенти, които надхвърлят възможностите на хората в тясно дефинирани задачи (Mnih et al. 2015), както и по-широко приложими такива (Silver et al. 2017)

Неотговорени въпроси Основният подход, използван в много от тези приложения е създаване на модел, който работи в добре дефинирани среди или такива в които се наблюдава пълна информация. Допълнително, агентите взимат решения на базата на изчислени точкови оценки. Възможно е това да намаля ефективността на тези модели, както и обяснението на взетите решения. Открит остава въпросът дали добавянето на вероятностно разбиране за средата може да се справи с тези проблеми (Bellemare et al. 2017) (Anonymous 2018).

Принос В тази работа добавяме вероятностно разбиране за средата и разработваме модели и техники за ефективно Бейсово изучаване на ГПИ среди. Също така създаваме конкретна среда, която Агентът изучава. Например, агентът трябва да наблюдава дадено изображение и избере действие, което максимизира вероятността за постигане на поставена цел. Този модел ще опише процесът за взимане на решения използвайки вероятностни разпределения. С други думи, целта на работата е създаване на агент, който ефективно изучава визуални среди.

Дългосрочна мотивация Основен стремеж на работата е да направи принос към изграждането и развитието на мислещи машини, както и приложи създадените модели в конкретни приложения. Техниките предложени тук са стъпка към достигането на бъдеще, в което агентите могат ефективно да взаимодействат с реалния свят (или по-сложни виртуални среди) и изпълняват комплексни задачи.

Краткосрочна мотивация Създаване на автоматизирани тестове за оценка на

качеството на софтуерен продукт използвайки агент. Търсене за семантични и логически грешки в дадена програмна ГПИ среда. Възпроизвеждане на стъпки, необходими за възпроизвеждане на грешка.

Съществуващите методи не дават възможност за споделяне на наученото от друго приложение, намиране на аномалии във функционалността, бързо научаване на промени (премахване на старо знание) и оценка на несигурността при изпълнение на действие.

1.2 Цели на дисертацията

Целта на дисертацията е да дефинира политика π, определяща поведението на агента.

- Дефиниране на множество от възможни действия на агента (пространство на действията)
- Създаване на модел, който избира действията на агента
- Създаване на среда (за тестване и използване на агента) в която ще работи агента
  - Подбиране на приложения (applications)
  - Измерване на покритието на код
  - Създаване на изображение и сегментация от текущото състояние на средата
- Избор на метрики за оценка на действията на агента
- Предварително обучение (с учител) на агента с данни от хора, взаимодействащи със средата (imitation learning)
- Провеждане на експерименти и анализ на постигнатите резултати

Целта на настоящата дисертацаионна работа е да създаде система за автоматизирано тестване на ГПИ, която използва за входни данни само визуалния изход на тестваното приложение. За постигане на целта трябва да се изпълнят следните задачи:

- Избор на подходящи оценъчни функции и награди, които да мотивират максималното покритие на програмен код по време на тестване
- Създаване на структури, в които да се запазват поредиците от стъпки, необходими за повтаряне на тестови случай
- Създаване на модел, който генерира поредица от действия, използвани за изграждане на тестовите случаи
- Създаване на модел, който намира аномалии по време на изпълнение на програмата
- Автоматично именуване на отделни екрани и действия с цел улесняване на разбирането
- Създаване и провеждане на експерименти, които да сравнят преложения модел с вече съществуващи такива
- По подадено изображение, йерархия на изгледите и действия, да се определи кои действия са валидни върху кои елементи

1.3 Структура на дисертационния труд

Глава 2 дава познания върху Дълбокото подсилено обучение и Бейсовото моделиране. **Глава 3** поставя целите и задачите на текущата работа. В **Глава 4** се създава среда за тестване на Android мобилни приложения. **Глава 5** представя модел за генериране на входни данни за тестови случаи. В **Глава 6** се представя модел за намиране на аномалии в тестови случай по подадено изображение. Цялостната система е представена в **Глава 7** заедно с емпирични сравнения спрямо други решения. Нерешени проблеми, бъдещи подобрения и дискусия се намират в последната глава.

Глава 2

Литературен обзор

2.1 Подсилено обучение

Една от основните задачи в сферата на изкуствения интелект е взимане на поредица от решения в стохастична среда. Един конкретен пример за взимане на решения в стохастична среда е агент, който изучава ГПИ. Тази задача се състои в избор на редица от решения, които да максимизират броят на разгледаните състояния на текущото приложение. Това е по-сложно от задачи, в които трябва да се направи само едно решение. Оценката за представянето на агента може да се даде само след много извършени от него стъпки. Това означава, че той може да избере неправилно действие сега и да разбере за това много по-късно, т.е. имаме *забавяне на последствията*. Допълнително, не може да наблюдаваме точното състояние на средата, поради липсата на точен модел на средата, която се изучава.

Основният начин за моделиране на такива среди са Марковски вериги.

Марковските вериги за вземане на решения (MDP) моделират системи, които искаме да контролираме. Във всяка времева стъпка t, системата се намира в дадено състояние s. Например, описаният агент може да се намира на даден екран от приложението, след като е натиснал определен бутон. Системата преминава през различни състояния като резултат от действията, които сме избрали. Задачата ни е да избираме действия, които са добри и да минимизираме броя на

тези, които не са. Разнообразни проблеми са моделирани чрез Марковски вериги (MDP формализма). Някои примери за използване на марковски вериги са системи за препоръки (Joachims et al. 1997), рутиране на мрежи (Boyan et al. 1994), управление на асансьори (Crites & Barto 1996), навигация на роботи (Sutton & Barto 1998).

Подсиленото обучение (RL) (Sutton & Barto 1998) дава начини за решаване на задачи, дефинирани чрез MDP формализма. Самообучаващ се агент с подсилено обучение (RL) взаимодейства със средата за определено време. На всяка времева стъпка t, агентът получава състояние s_t от пространството на състоянията S и избира действие a_t от пространство с действия A, следвайки политика $\pi(a_t|s_t)$. Политиката π определя поведението на агента, т.е. в определено състояние s_t, какво действие агентът трябва да избере. Тя дава функция за преобразуване на състояние s_t до състояние s_{t+1} чрез действие a_t. Използвайки дадена политика, агентът получава скаларна награда r_t и преминава в следващо състояние s_{t+1}, което се определя от функцията за награди $R(s,a)$ и функцията, даваща вероятности за преминаване в друго състояние $P(s_{t+1}|s_t,a_t)$. Когато моделът, който моделира поведението на агента е дискретен, т.е. може да се разглежда като отделни епизоди, описаният процес продължава докато агентът не достигне до крайно състояние. Тогава агентът се рестартира за започване на ново обучение. Общата награда е дефинирана като:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

представлява обезценена стойност с фактор $\gamma \in (0,1]$. Агентът се опитва да максимизира очакваната награда във всяко състояние.

Функция на стойностите $Q^\pi(s,a)$ дава предсказана обща бъдеща награда, която измерва до колко са добри дадено състояние или двойка състояние-действие. Стойността на дадено действие $Q^\pi(s,a) = E[R_t|s_t = s, a_t = a]$ ни дава очакваната награда за избиране на дейсвие a в състояние s и следвайки фиксирана политика π. Оптимална стойностна функция $Q^*(s,a)$ предоставя действие a, което максимизира стойността на наградата за дадено състояние s. Може да дефинираме функция даваща стойност на състоянията $V^\pi(s)$, както и оптималната й версия $V^*(s)$ по сходен начин.

Извод: ние подсилваме обучението с въвеждане на награда.

2.1.1 Дълбоко обучение

Нека разгледаме един от най-простите статистически модели - линейната регресия (Gauss 1809; Legendre 1805). Нека е дадено множество от N входно-изходни двойки $\{(x_1, y_1), ..., (x_n, y_n)\}$. Например, нека x да е тегло в кг, а y - височина в см на N човека. Линейната регресия прави предположението, че съществува линейна функция, която преобразува всяко $x_i \in \mathbb{R}^Q$ към $y_i \in \mathbb{R}^D$. Тогава нашият модел е линейна трансформация на входните данни:

$$f(x) = xW + b$$

където W е $Q \times D$ матрица и b е вектор от D елемента. Тогава, задачата се свежда до намиране на такива параметри W и b, които минимизират средната квадратична грешка:

$$e = \frac{1}{N} \sum_i ||y_i - (x_i W + b)||^2$$

В общия случай, връзката между x и y може да не е линейна. Тогава искаме да дефинираме нелинейна функция $f(x)$, която преобразува входните данни до изходни. За тази цел може да приложим linear basis function regression (превод?) (Bishop 2007; Gergonne 1815), където входните данни x се подават на K фиксирани скаларни нелинейни трансформации $\phi_k(x)$ за създаване на свойствен вектор $\Phi(x) = [\phi_1(x), ..., \phi_k(x)]$. Трансформациите ϕ_k наричаме базисни функции. Върху така създадения вектор се прилага линейна регресия. LBFR може да се сведе до линейна регресия, когато $\phi_k(x) := x_k$ и $K = Q$. Този тип функции се смятат за фиксирани и взаимно ортогонални. Когато тези ограничения се пропуснат говорим за *параметризирани* базисни функции.

2.1.1.1 Изкуствени невронни мрежи

Когато подредим параметризирани базисни функции в йерархия, може да говорим за изкуствени невронни мрежи. Всеки свойствен вектор в тази йерархия ще наричаме слой. Композицията от подобни слоеве води до голямата гъвкавост на тези модели. Често те постигат високи резултати на различни задачи и могат да се приложат върху реални проблеми, работещи върху терабайти от данни.

Feed-forward neural networks. Нека разгледаме модел с един *скрит слой* (Rumelhart et al. 1985). Нека x е вектор с Q елемента, представящ входните данни. Трансформираме го с афинна трансформация до вектор с K елемента. Отбелязваме с W_1 линейната преобразуваща матрица (матрица на теглата) и с b транслацията използвана за трансформиране на x за да получим $xW_1 + b$. Върху всеки елемент на получената матрица се прилага нелинейна функция $\sigma(\cdot)$. Резултатът е т. нар. *скрит слой*, а всеки елемент се нарича *мрежова единица*. Върху резултата се прилага втора линейна трансформация с матрица на теглата W_2, която преобразува скрития слой до изходен вектор с D елемента. Имаме $Q \times K$ матрица W_1, $K \times D$ матрица W_2 и b - вектор от K елемента. Резултат от дадена невронна мрежа би бил:

$$\hat{y} = \sigma(xW_1 + b)W_2$$

при дадени входни данни x.

Когато използваме невронната мрежа за решаване на регресионна задача, може да минимизираме Евклидовата грешка:

$$e^{W_1,W_2,b}(X,Y) = \frac{1}{2N}\sum_{i=1}^{N}||y_i - \hat{y}_i||^2$$

където $\{y_1, \ldots, y_n\}$ са N наблюдавани изходни стойности, $\{\hat{y_1}, \ldots, \hat{y_n}\}$ са изходни данни от модела, а $\{x_1, \ldots, x_n\}$ са входните данни. Предполагаме, че минимизирайки тази грешка спрямо W_1, W_2, b ще получим модел, който генерализира добре при нови данни $X_{\text{test}}, Y_{\text{test}}$.

Когато задачата е да се предскаже класът, към който x принадлежи, от множес-

твото $\{1, \ldots, D\}$, използваме същия модел. Промяната се състой в това, че прилагаме softmax функция върху получения резултат. Тази функция ни дава нормализирани оценки за всеки клас:

$$\hat{p}_i = \frac{exp(\hat{y}_i)}{\sum d' exp(\hat{y}'_i)}$$

Когато вземем логаритъма от горната функция, получаваме softmax грешка:

$$e^{W_1, W_2, b}(X, Y) = -\frac{1}{N} \sum_{i=1}^{N} log(\hat{p}_{i,c_i})$$

където $c_i \in \{1, \ldots, D\}$ е наблюдаваният клас за вход i.

Описаният по-горе модел има проста структура, но може да бъде разширен за по-специализирани задачи. Този тип по-сложни модели се използват, когато задачите изискват обработка на поредици или изображения.

Convolutional Neural Networks CNN е архитектура (LeCun et al. 1989), която се използва при изображения. Задачи, които до скоро се смятаха за нерешими, имат решения посредством този тип модели (Hinton et al. 2012). Моделът е създаден чрез рекурсивно приложение на конволуции и обединяващи слоеве. Конволюционният слой е линейна трансформация, която запазва пространствена информация от входното изображение.

Recurrent neural networks (RNN) RNN е модел (Rumelhart et al. 1985; Werbos 1988), базиран на поредици от данни, който се използва за обработка на текст, обработка на видео и други (Kalchbrenner & Blunsom 2013; Sundermeyer et al. 2012). Входните данни за RNN са поредица от символи. За всяка времева стъпка t, проста невронна мрежа е приложена върху единствен символ, както и изходните данни от мрежата от предишната стъпка.

Конкретно, при дадена редица от входни данни $x = [x_1, \ldots, x_t]$ с дължина T, прост RNN модел е създаден чрез повтарящо се приложение на функция f_h. Така се генерира скрито състояние h_t за времева стъпка t:

$$h_t = f_h(x_t, h_{t-1}) = \sigma(x_t W_h + h_{t-1} U_h + b_h)$$

за някаква нелинейна функция σ. Изходните данни от модела може да бъдат дефенирани като:

$$\hat{y} = f_y(h_T) = h_T W_y + b_y$$

Съществуват и по-сложни RNN модели, като LSTM (Hochreiter & Schmidhuber 1997) и GRU (Cho et al. 2014).

2.1.2 Дълбоко подсилено обучение

Този тип методи се класифицират, когато използваме дълбоки невронни мрежи за апроксимиране на някой от компонентите на подсиленото обучение: функция на стойностите $V(s;\theta)$, политика $\pi(a|s;\theta)$ или модела за промяна на състояние и награди. Параметрите θ представляват тегла в дълбоки невронни мрежи. Когато използваме "плитки" модели, като например линейна регресия, дървета за вземане на решения и др. като апроксиматори на функция, имаме "плитко" подсилено обучение с параметри θ за съответния модел. Основната разлика между дълбокото и плиткото подсилено обучение се състой в апроксиматора на функцията, която използват. Когато се използва извън политикова апроксимация - например на нелинейни функции, може да се наблюдават нестабилност и разходимост (Tsitsiklis et al. 1997). Въпреки това, скорошната работа върху дълбоки Q-мрежи (Mnih et al. 2015) и *AlphaGo* (Silver & Hassabis 2016) стабилизират процеса на обучение и постигат много добри резултати.

Дълбокото подсилено обучение започна рязкото си развитие с работата на (Mnih et al. 2015). Преди това, RL даваше нестабилни резултати, когато се използваха нелинейни апроксиматори като невронни мрежи. Дълбоките Q мрежи (DQN) направиха няколко важни приноса: 1) стабилизиране на обучението, използвайки дълбоки невронни мрежи (Lin 1992) 2) подход за цялостно обучение без почти никакво познание за областта 3) обучаване на гъвкава невронна мрежа с еднакъв алгоритъм за изпълняване на различни задачи, например 49 Atari игри (Bellemare et al. 2013), на които се представят по-добре от всеки известен алгоритъм до момента.

2.1.2.1 Double DQN

(Van Hasselt et al. 2016) предложиха Double DQN (D-DQN) за справяне с проблема на прекалена увереност (overestimate?) на Q-learning алгоритъма. В базовият алгоритъм (както и в DQN), параметрите се обновяват според:

$$\theta_{t+1} = \theta_t + \alpha(y_t^{\theta} - Q(s_t, a_t; \theta_t))\Delta_{\theta_t} Q(s_t, a_t; \theta_t)$$

където

$$y_t^Q = r_{t+1} + \gamma \max_{\alpha} Q(s_{t+1}, a; \theta_t)$$

така че оператора $\max$ използва еднакви стойности, за да избере и оцени дадено действие. Като следствие от това, е по-вероятно да избере недостатъчно добри стойности. Double DQN предлага да оцени алчната политика спрямо невронна мрежа, но използва друга, за да оцени стойността й. Това може да се постигне с малка промяна на DQN алгоритъма, заменяме y_t^Q с:

$$y_t^{D-DQN} = r_{t+1} + \gamma Q(s_{t+1}, \max_{\alpha} Q(s_{t+1}, a_t; \theta_t); \theta_{\bar{t}})$$

където θ_t е параметър за първата невронна мрежа, а $\theta_{\bar{t}}$ е параметър за целевата мрежа.

2.1.2.2 Асинхронни методи

(Mnih et al. 2016) предложи асинхронни методи за четири RL алгоритъма: Q-learning, SARSA, n-step Q-learning and advantage actor-critic и asynchronous advantage actor-critic (A3C). Този подход използва паралелни агенти, които използват различни политики за изучаване на средата. Асинхронните методи могат да се изпълняват върху многоядрени процесори. Те се изпълняват много по-бързо и предоставят по-бързо обучение от други известни методи.

2.2 Бейсова статистика

Избиране на следващо действие по време на създаване на тестов случай пряко зависи от увереността във взимането на правилното решение. Несигурността от избиране на действие може да бъде моделирана посредством Бейсов подход.

Нека θ е неизвестна стойност, която може да е скаларна, векторна или матрица. Методите за статистически извод (inference) могат да ни помогнат да я намерим. Класическият статистически подход третира θ като фиксирана стойност. Единствената информация, която използваме за намиране на неизвестната стойност, идва от данните, с които разполагаме. Изводът се базира на резултат, получен от фунцкията на правдоподобие на θ, която свързва стойности от $p(y|\theta)$ с всяка възможност на θ, където $y = (y_1, ..., y_n)$ е вектор с наблюдавани стойности.

Бейсовият подход третира θ като случайна стойност. За достигане на извод се използва разпределението на параметри при дадени данни $p(\theta|y)$. Това разпределение се нарича апостериорно. Освен функцията на правдоподобие, Бейсовият подход включва априорно разпределение $p(\theta)$, което представя вярванията ни за θ преди да се разгледат данните.

Теоремата на Бейс дава връзка между фунцкията на правдоподобие и априорното разпределение:

$$p(\theta|y) = \frac{p(\theta|y)p(\theta)}{p(y)}$$

където:

$$p(y) = \int p(y|\theta)p(\theta)d\theta$$

Формулата на Бейс може да бъде пренаписана по следния начин:

(1)
$$p(\theta|y) \propto p(\theta|y)p(\theta)$$

тъй като $p(y)$ не зависи от θ

Когато θ е многомерна величина може да напишем уравнение (1) използвайки маргиналните апостериорни разпределения като например:

$$p(\theta_1|y) = \int p(\theta|y)d\theta_2$$

където $\theta = (\theta_1, \theta_2)$. В много случаи резултатите са многомерни и точни изводи може да бъдат направени само аналитично. Поради тази причина често се използват приближения.

2.2.1 Монте Карло алгоритми за Марковски Вериги (MCMC)

MCMC алгоритмите правят неявно интегриране като взимат извадки от апостериорното разпределение. По този начин се намират приближения на стойностите, от които се интересуваме.

В съществото си тези методи създават Марковска верига с апостериорното разпределение на параметрите като стационарно разпределение. Когато веригата е крайна и повтаряща се, стойността на θ може да бъде оценена от извадки на средни пътища. Генерираните извадки $\theta^{(t)}, t = 1, \ldots, N$ от това разпределение дават представа за целевото разпределение.

2.2.1.1 Метрополис-Хастингс алгоритъм

Този алгоритъм е предложен от Metropolis (Metropolis et al. 1953) и по-късно генерализиран от Hastings (Hastings 1970). Методът създава Марковска верига с желаното стационарно разпределение. Алгоритъмът избира кандидат стойност θ' от предварително избрано разпределение $q(\theta, \theta')$, където $\theta' \neq \theta$. Избраната стойност θ' се проверява чрез приеми-откажи метод (accept-reject step), за да се подсигури, че принадлежи на целевото разпределение.

2.2.1.2 Извадки на Гибс

Този метод, предложен от Geman и Geman (Geman & Geman 1984), често се представя като специален случай на Метрополис-Хастингс алгоритъма.

2.2.2 Извод със свободни вариационни параметри

Variational Inference (VI) методите обикновено предлагат по-добри резултати спрямо MCMC, когато времето за изпълнение е ограничено. Допълнително предимство на тези подходи е, че те са детерминирани. Систематичната грешка и дисперсията се приближават до 0 при MCMC методите, за колкото повече време бъдат оставени да се изпълняват те. Тези свойства правят MCMC алгоритмите много ефективни на теория. В практиката обаче, времето за изпълнение и изчислителната мощ са ограничени. Това налага търсенето на по-бързо методи дори когато това намаля точността на получените резултати.

Този тип методи дефинират приближено вариационно разпределение $q_\omega(\theta)$, параметризирано от ω, с лесна за оценяване структура. Искаме приближеното разпределение да е максимално близко до това на апостериорното. За целта свеждаме задачата до оптимизационна и минимизираме Kullback-Leibler (KL) (Kullback & Leibler 1951) отклонението спрямо ω. Интуитивно, KL измерва приликата между две разпределения:

$$KL(q\omega(\theta) \,||\, p(\theta|x,y)) = \int q\omega(\theta) log \frac{q\omega(\theta)}{p(\omega|x,y)} d\omega$$

(Define x,y - dataset)

Този интеграл е дефиниран, когато $q\omega(\theta)$ е непрекъсната спрямо $p(\theta|x,y)$. Нека $q^*_\omega(\theta)$ е минимизираща точка (може да е локален минимум). Тогава KL може да ни даде приближение на апостериорното разпределение:

$$p(y^*|x^*,x,y) \approx \int p(y^*|x^*,\theta) q^*_\omega(\theta) d\theta =: q^*_\omega(y^*|x^*)$$

VI методите заменят изчисляването на интеграли с такова на производни. То-

ва е много подобно на оптимизационните методи използвани в DL. Основната разлика се състой в това, че оптимизацията е върху разпределения, а не точкови оценки. Този подход запазва много от предимствата на Бейсовото моделиране и представя вероятностни модели, които дават оценка на несигурността в изводите си.

2.2.3 Бейсови Невронни Мрежи

Един от големите недостатъци на съществуващите архитектури на невронни мрежи е, че изводите, които получаваме, са оценки на точки. Моделите не казват до колко са сигурни в предложените резултати. Когато например един лекар получи резултат от даден модел, той трябва да знае защо и как моделът е стигнал до него. Бейсовата статистика може да даде отговор на тези въпроси (Gal & Ghahramani 2015). Дори при модели използващи RNN, Бейсова интерпретация на задачата дава по-добри резултати от съществуващи такива (Gal & Ghahramani 2016).

Бейсови невронни мрежи, предложени в края на 80-те години (Kononenko 1989) и задълбочено изучавани по-късно (MacKay 1992; Neal 2012), предлагат вероятностна интерпретация на моделите за дълбоко обучение, като представят теглата им като вероятностни разпределения. Този тип модели са устойчиви на пренастройване (overfitting), предлагат оценки на несигурността и могат да се тренират върху малко на брой данни.

2.3 Автоматизирано тестване на ГПИ

Проверката за правилно поведение на софтуер продукт е неизменна част от създаването му. Откриване и поправяне на всички потенциални проблеми преди той да бъде доставен до крайния потребител може да се сметне за най-добър случай.

2.3.1 Автоматизирано тестване на Android приложения

Мобилните приложения също имат нужда от проверка на качеството. Поради тази причина, в последните години засилено се разглеждат начини за автоматизацията на подобен вид тестове. Много голяма част от извършената работа до момента се състои в създаване на входни данни за приложения за мобилната операционна система Android. Подходите използвани до момента, се различават по начина, по който създават входнни данни и изучават и използват евристики за приложението.

2.3.1.1 Съществуващи системи

Dynodroid (Machiry et al. 2013) е инструмент, който се базира на случайно изучаване. Предлага се и ръчен начин за въвеждане на входнни данни, когато системата е заседнала.

MobiGUITAR (Amalfitano et al. 2015) строи модел на приложението по време на тестване. За всяко ново състояние се поддържа списък с възможни действия, които се изпълняват използвайки DFS (depth first search) стратегия.

SwiftHand (Choi et al. 2013) се опитва да максимизира покритието на код за тестваното приложение. Допълнително, инструментът се старае да минимизира броя рестартирания на приложението. SwiftHand генерира единствено докосвания и скролвания.

PUMA (Hao et al. 2014) предлага генерална среда за автоматизиране на ГПИ. Инструментът предлага рамка за програмиране, в която могат да бъдат имплементирани различни стратегии за изучаване на тестваното приложение.

2.3.1.2 Покритие на код

Една от основните цели на системите за автоматизирано тестване на софтуер е да постигнат максимално покритие на програмния код. Няколко решения се опитват да постигнат това и за операционната система на Android.

BBoxTester (Zhauniarovich et al. 2015) е рамка за изготвяне на доклади относно

покритието на програмния код, без той да бъде наличен. За разлика от други подобни системи, BBoxTester предлага детайлни метрики за покритието на отделни класове, методи и т.н. В основата на системата се използва друг софтуерен продукт - Emma (Roubtsov & others 2005). BBoxTester е система с отворен код, намираща се на https://github.com/zyrikby/BBoxTester. За съжаление системата е неподдържана (от 2015г.) и несъвместима с нови версии на Android.

CovDroid (Yeh & Huang 2015) е друга система за тестване посредством подход на черната кутия (black-box testing). Програмният код на продукта не е наличен. CovDroid изчислява покритието на код като инструментира кода на приложението и използва Android Debug Bridge (adb) за да наблюдава изхода от изпълнение на програмата.

ABCA (Huang et al. 2015) използва подход, много близък до този на CovDroid. Софтуерният пакет може да бъде намерен на http://cc.ee.ntu.edu.tw/~farn/tools/abca/. По време на този обзор, страницата на инструмента не беше активна. Авторите на статията не отговориха на запитването за активен адрес за изтегляне на ABCA.

GUITracer (Molnar 2015) представя иновативен подход за визуализация на покритието на код, когато приложението е базирано на ГПИ. Основен недостатък на системата е ограничението за работа върху Java AWT, SWING или SWT рамки за изграждане на ГПИ.

GroddDroid (Abraham et al. 2015) предлага автоматично намиране и изпълнение на зловреден софтуер (malware). Системата предлага и измерване на покрит код. Софтуерът може да бъде намерен на http://kharon.gforge.inria.fr/. Програмният код е ясно документиран и лесен за употреба. Един недостатък е използването на Logcat монитора за извличане на метрики за покритие на кода. Повечето от горепосочените системи използват този подход.

2.3.1.3 Текущо състояние (State of the art?)

(Choudhary et al. 2015)

2.3.2 Проверка на качеството

- Достатъчно бързо ли е? (Model should monitor for speed exec anomalies or report just slow parts)
- Как да повторя грешката? (Provide/execute steps for reproduction)
- Има ли разлики в изходните данни? (Change in hierarchy/image screenshot)

Глава 3

Избор на действие

Проверката за качество на софтуерни продукти често се извършва посредством автоматизирани, полуавтоматизирани или ръчно изпълняващи се тестове. Основна цел при създаване и изпълнение на тези тестове е постигане на високо или пълно покритие на програмен код (Zhu et al. 1997). От своя страна, това покритие повишава възможността програмата да не достига непредвидени състояния и притежава желаната функционалност (Ohba 1982).

Създаването на тестове, които проверяват цялостната функционалност на системата често се извършва от специалисти по проверка на качеството (QA). Те създават автоматизирани или ръчно изпълняват тестове, спрямо предварително създадени спецификации. Част от тестовете, обхващащи целият софтуерен продукт се извършват спрямо графичната потребителски интерфейс (ГПИ), който той предоставя. Тези тестове (наречени ГПИ тестове (GUI tests) симулират взаимодействието на потребител с програмата.

Създаването на автоматизирани ГПИ тестове е обвързано с трудности, като често променящи се визуални елементи, забавено изпълнение, достигане на непредвидени състояния на средата и др. (Memon 2002). Често, поради тази причини подобен вид тестове се изпълняват изцяло ръчно или полуавтоматизирано, което изисква взаимодействие с експерт.

QA експертът взаимодейства с ГПИ чрез поредица от действия (извършени чрез мишка, клавиатура, докосвания върху екран и/или др.), които променят ГПИ и

водят до друго състояние (в частност, нов екран). Когато това състояние не е наблюдавано до момента, покритието на програмен код се увеличава, поради нуждата от изпълнение му за създаване на самото състояние.

Тогава, целта при създаване на ГПИ тестове може да се определи като посещаване на всяко състояние на визуалната среда поне веднъж. Повторно наблюдение на дадено състояние може да е необходимо поради допълнителни възможни действия. Действията, които се избират, определят последователността на наблюдаваните състояния, както и бързодействието на текущия тест (минимален брой на взети действия за постигане на целта).

В тази част от работата ще създадем модел, който избира следващо действие, когато средата се намира в определено състояние. Това действие трябва да бъде избрано, така че да максимизира увеличението на покритие на програмен код и минимизира нуждата възможността за попадане във вече наблюдавано състояние. Състоянието на средата ще бъде закодирано чрез матрица, отговаряща на елементите в нея. Това е опростен подход към решаване на поставената задача, като той ще бъде разширен в следващата глава.

Решаването на задачата използва т.нар подход базиран на наличните данни (data-driven). Конкретно, създаваме БИНМ, която приема състоянието на средата като входни параметри и изчислява апостериорно разпределение за оценката на (до колко добро е) всяко от възможните действия. Обучението на модела се извършва чрез предварително събрани данни.

3.1 Литературен обзор

Съществуват различни подходи за автоматизирано създаване на ГПИ тестове за мобилни и уеб приложения: (Amalfitano et al. 2015), (Choi et al. 2013), (Moreira & Paiva 2014), (Salvesen et al. 2015), (Moreira et al. 2017) (Memon 2002), но тяхната практическа употреба и ефективност са незадоволителни (Choudhary et al. 2015).

Предложеният подход е вдъхновен от: - (Mnih et al. 2015) - използват се ИНМ за обучение на агент, който играе игри, надвишавайки възможностите на човек на някои от тях - (Shi et al. 2017) - среда, предоставяща възможност за създаване

и обучение на агенти, изпълняващи задачи в уеб среди (напр. закупуване на самолетен билет) - (Chang et al. 2010) - визуален скриптов език за създаване на тестове, който използва изображения за определяне на следващо действие

Текущият подход се различава по, това че:

- автоматизира напълно (или в голяма степен) създаването на ГПИ тестове
- предоставя среда, даващата информация за новото покритие на код при взимане на действие
- оценя несигурността за избиране на действие, което може да е полезно за:
  - Състояния в които е необходима допълнителна информация за да бъде продължено изучаването на средата (напр. екран изискваш потребителско име и парола)
  - достигнато е неочаквано състояние (аномалия), което може да е свързано с грешка в програмният код

3.2 Модел

Нека имаме среда E, намираща се в състояние $s \in \mathbb{S}$, върху което могат да бъдат изпълнени действия от множеството от действия $\mathbb{A}$. При избор на действие $a \in \mathbb{A}$, средата E предоставя награда r, която приема стойности в интервала $[0; 1]$ и преминава в ново състояние s' (в частност, $s' = s$, т.е. средата може да не премине в ново състояние). Множеството $\mathbb{A}$ е ненаредено и всяко $a \in \mathbb{A}$ може да се обозначи с единствено цяло число, като по този начин въвеждаме наредба в $\mathbb{A}$. Всяко състояние на средата S позволява изпълнението на действия $\mathbb{A}$, които са предварително дефинирани. Множеството от всички възможни състояния на средата $\mathbb{S}$ е неизвестно.

Нека след първоначално обучение от специалист имаме матрица на преходите D с размерност $n \times 3$, където n е броя на преходите. Всеки ред от D дефинира наредена тройка $(, ,)$, която описва получените награди при изпълнение на съответното действие за даденото състояние.

Нека имаме състояние s', за което D не съдържа информация. В този случай, целта е да намерим подмножеството от действията, така че изпълнението им да води до получаване на оптимална награда от средата E.

Задачата е решена, когато D съдържа информация за всички състояния $s \in \mathbb{S}$.

3.3 Пример

Експертът е обучил … като се стреми да посети всяко състояние поне по веднъж и минимизира избраните действия.

Нека имаме визуална среда с 5 възможни действия:

- a_1 - клик горе в ляво
- a_2 - клик горе в дясно
- a_3 - клик долу в ляво
- a_4 - клик долу в дясно
- a_5 - връщане назад

Нека след първоначално обучение от специалист качество на софтуер (QA expert) имаме матрицата на преходите D, дефинирана като:

s_{x_1}	s_{x_2}	s_{x_3}	s_{x_4}	action	reward
b	w	g	g	a_3	0.25
b	w	w	b	a_5	0.00
b	w	g	g	a_4	0.25

където:

- $s = (s_{x_1}, s_{x_2}, s_{x_3}, s_{x_4})$ е вектор от характеристиките на състоянието
- w - бял цвят, върху който не могат да бъдат предприемани действия
- b - син цвят, който представя текстова информация
- g - зелен цвят, който представя бутон

Първото състояние (първият ред от матрицата D) може да се визуализира като:

Избрано е действие a_3 за което е получена награда 0.25. Средата преминава в ново състояние (втори ред от D):

Тук е избрано действие a_5 - връщане назад и получаваме награда 0. Това ни връща в първоначалното състояние:

Тук сме избрали a_4 и сме получили награда от 0.25. Получаваме ново състояние:

Получихме ново състояние, което не е описано в D. Свеждаме задачата до избор на действие, което ще ни даде максимална награда за текущото състояние.

3.4 Обучение на модела

Задачата се свежда до намиране на действие, което предоставя най-висока награда за текущото състояние. Допълнително, вероятностното разпределение над всички възможни действия би позволило оценяване на несигурността при избор на действие. С тази информация може да решим кога да използваме знанията за средата и кога да я изучаваме []. Когато вероятността е по-голяма ще е по-вероятно да изберем конкретното действие.

3.4.1 Бейсова невронна мрежа (БНМ)

3.5 Експерименти

3.6 Заключение

Глава 4

Среда за изучаване (RL exploration) на ГПИ приложения

Много от съществуващите системи за автоматизирано тестване на Android приложения се опитват да изградят решения, които взимат предвид недостатъците при тестване на приложения. Някои от трудностите повече не съществуват благодарение на напредъка на модерния компютърен хардуер, а други могат да бъдат решени много по-ефективно благодарение на новъведени инструменти за разработка за Android. Например, **SwiftHand** (Choi et al. 2013) се опитва да намали нуждата от преинсталиране на приложението върху устройството. В по-новите си версии, adb, предлага способ за изчистване на състоянието на дадено приложение, без нужда от преинсталирането му.

От особена важност за изграждане на алгоритъм в среда за подсиленото обучение е наличието на награда. Повечето от изградените системи се опитват да максимизират покритието на код. В практиката тази метрика е важна, но и недостатъчна. Фактът, че дадена част от програмния код се е изпълнила и не е предизвикала грешка в програмата не означава, че поведението на програмата е правилно (или не се е променило без това бъде желания ефект от разработчиците).

За нуждите на текущата работа и всеки желаещ да използва се предлага обща среда за изучване/тестване на мобилни приложения. Поради липсата на други свободни инструменти (или такива, които са използваеми). Системата е свобод-

на за използване, с отворен код и може да бъде намерена на https://github.com/appgym/appgym.

4.1 Android специфична среда

Средата се състои от два основни компонента - клиент и сървър. Сървърът работи върху Android устройството и предоставя данни за постигнатото покритие на код, изпълнение на действията, генерирани от модела и изображение за текущото състояние. Клиентът предоставя възможните действия на модела, както и комуникира със сървъра за да представи неговата функционалност.

Клиентът предоставя интерфейс към средата подобен на този на OpenAI gym (Brockman et al. 2016). Двата основни метода, които реализира са `reset()` и `step(action)`. `reset()` предоставя възможност на средата да се върне до първоначално състояние. Това се постига чрез спиране на приложението (ако то е стартирано), изтриване на данните поддържащи състоянието му, стартирането му и предоставяне на образ от екрана, както и възможните действия за състоянието. Изброената функционалност се реализира посредством adb команди и библиотеката `uiautomator` https://github.com/xiaocong/uiautomator. Методът `step(action)` изпълнява избраното действие и предоставя новото състояние на средата, заедно с получената награда и новите възможни действия. Тук също се взима решение дали текущия епизод от обучението е приключил.

Множеството от възможните действия за текущото състояние се базират на броя и видовете графични елементи в него. Всеки елемент върху който може да се извърши докосване, задържане, скролиране, влачене и т.н. се превръща в действие. Множеството от графични елементи се извлича посредством библиотеката `uiautomator`.

Наградата за всяка избрана стъпка пряко се базира на покритието на код за текущия епизод. Стойността се изменя в интервала $[0; 1.0]$ и е нарастваща. Получаването на наградата след всяка стъпка е необходимо за обучение на модела. Скоростта на изпълнение пряко влияе на общото бързодействие на системата. За намиране на текущото покритие на код и изграждане на доклад се използва `JaCoCo` (Hoffmann et al. 2009). `JaCoCo` е интегриран в инструментите за

разработка на Android и се използва основно, когато е нужно покритие на код базирано на преминали тестове.

За целта на текущата работата бяха направени някои промени, които предоставят възможност за извличане на необходимите данни, докато програмата се изпълнява и не е в тестова среда. Няколко подхода бяха изпробвани за изграждане на крайните доклади. Първоначално бяха използвани adb команди и генериране на доклад посредством gradle задача. Бързодействието не беше задоволително - необходими бяха около 2 секунди на съвременен мобилен компютър. Около повината от времето се губеше в генериране на доклад, който предоставя повече от необходимата информация.

Крайното решение използва комбинация от `HTTP` сървър на устройството, клиент, специализиран начин за записване на данните за покритие на код и специализиран генератор за доклади. Допълнително бързодействие се постига чрез Nailgun https://github.com/martylamb/nailgun сървър, който изпълнява генератора за доклади. Така описаните оптимизации извършват необходимата работа за около 20 милисекунди (или 0.02 секунди) на същия компютър.

Взимането на текущото състояние се състой в направата на изображение на текущия екран на устройството. Тази задача отново се извършва от библиотеката `uiautomator`. Изображението може да бъде намалено до желани размери в зависимост от изискванията на задачата. Така полученото изображение се представя за текущо състояние под формата на тензор.

4.2 Web специфична среда

Web средата предоставя възможност за изучаване на мобилни web приложения. Тя използва библиотеката `pyppeteer`, която предоставя автоматизиран достъп до Web Browser.

Системата AppGym е структурирана така че да може да се използва с други мобилни и настолни операционни системи. Интерфейсът е много сходен до този на OpenAI gym. Възможно е рамката да бъде променена, така че да бъде използвана за други задачи и други цели.

Глава 5

Изучаване на ГПИ среди

В тази глава изграждаме подход, който ни позволява да изучаваме среда, чието състояние се базира на изображения. По зададено изображение и множество от действия трябва да изберем действие, което максимизира изучената част от средата. По-конкретно, текущото състояние се определя от изображението на приложението и възможните действия с графичните елементи, а изучената част е моментното покритие на код.

Подходът, който използваме се базира изцяло на данните, които средата предоставя. В частност, разработваме модели, които представляват дълбока невронна мрежа, която приема изображения като входни данни и избира действие. Така създадения модел се тренира върху мобилни приложения за Android. Експерименталните резултати показват…

5.1 Related Work

Извличане на характеристики (features) от сензорна информация намалява нуждата от ръчното им закодиране и увеличава скоростта на изграждане на модели. Скорошните открития в областта на дълбокото самообучение доведоха до големи открития в компютърното зрение (Krizhevsky et al. 2012; Mnih 2013; Sermanet et al. 2013) и гласовото разпознаване (Dahl et al. 2012; Graves et al. 2013). Те използват различни архитектури за дълбоки невронни мрежи, вклю-

чително конволуции, многослойни персептрони (multilayer perceptrons) и рекурентни невронни мрежи. Използвани са в обучение с учител, без учител. Дълбока конволюционна невронна мрежа беше използвана за създаване на агент, който играе Atari 2600 компютърни игри (Mnih et al. 2013). За входни данни е използван видео вход с размери 84x84 пиксела и възможните действия. Работата използва повторение на преживяното (experience replay) (Lin 1992) и надгражда върху Neural Fitted Q-Learning (NFQ) (Riedmiller 2005). Още по-добри резултати бяха постигнати като се използва Монте Карло дървета за планиране, които бавно достигат извод, за обучение на дълбоки невронни мрежи, които са многократно по-бързи (Guo et al. 2014).

5.2 Дадено на агента

Предполагаме, че на стъпка t получаваме изображение и множество от възможни действия, определящи текущото състояние на средата. Искаме да създадем модел, който при подадени така описаните входни данни ни дава апостериорно разпределение, описващо вероятностите всяко от възможните действия да ни доведе до състояние с оптимално увеличение на покритието на програмен код.

Предизвикателството в така поставената задача се състой във факта, че изображенията са сложни многомерни обекти, а оптималното действие във всяко състояние може да е различно от това в което и да е друго. Нашият модел трябва да изгради вътрешно представяне на средата (напр. да научи какво е бутон, граници на отделните елементи и кои действия да използва върху тях) и да научи оптималните действия за всяко състояние.

1. Изображение на ГПИ средата - пример ГПИ. примери + изображения
2. Изучи ГПИ средата - да достигне всички възможни състояние на средата
3. Възможни действия -

Агентът щракване върху екрана (пример визуален) Агентът натиска и задържа (пример визуален) Агентът натиска и влачи курсора (пример визуален) Агентът scroll-ва нагоре и надолу

5.3 Задачи

- Да постигне 100% покритие на програмния код на даден софтуерен продукт (СП)
- Да генерира поредица от действия с които преминава през всички клонове на дървото, описващо възможните състояния на СП

5.4 Модел на агента

Нека всяко състояние s е представено като изображение и имаме множество от действия A, които отговарят на различни графични елементи на екрана. Искаме да изберем действие a, което максимизира покритието на код. Предизвикателството се състой в намирането на възможно най-кратките поредици от действия, когато A може да е различно за всяко състояние s.

Подобно на (Mnih et al. 2013) създаваме дълбока конволюционна невронна мрежа, която използва изображения за входни данни.Директна работа с реалните размери на изображение взето от устройството може да изисква прекалено много изчисления (computationally demanding). Съвременните мобилни устройства достигат до 3840x2160 разделителна способност (Syny Xperia Z5 Premium). Въпреки това, физическият размер на екраните на смартфоните достигат до 5.5“. Това ги прави далеч по-малки от екраните на настолните и преносимите компютри. Основното взаимодействие с мобилните устройства се извършва чрез различни жестове (натискане, задържане, приплъзване и т.н.) с екрана. Предвид физическите размери на пръстите на човек, екраните не могат да съдържат голямо множество от елементи с които може да се взаимодейства.

5.4.1 Пространство на състоянието (State space)

Дадено състояние s на средата съдържа цветно изображение I и информация за сегментацията DOM модел D, т.е. $s = (I, D)$. Изображението има размер $W \times H \times 3$, където W е широчината на изображението в пиксели, H височината на изображението в пиксели и 3 - броя на цветовете в палитрата (черве-

но, зелено и синьо (rgb)). DOM моделът е представен като списък от текстови елементи, а информацията за сегментацията на изображението е дадена от наредената четворка (x, y, w, h) x абцисната координата, y ординатната ос, w - широчината на сегмента, h - височина на сегмента.

5.4.2 Пространство на действията (Action space)

Позицията на курсора $m = (m_x, m_y) \in [0, W) \times [0, H)$ се моделира чрез мултиномиално разпределение върху възможните позиции. ГПИ средата не изисква наличие на клавиатура, защото . Възможните действия са: click, drag, scroll-up, scroll-down.

5.4.3 Представяне на изображенията

Векторното представянето на изображения се базира на напредъка постигнат в компютърното зрение, където Конволюционните Невронни Мрежи (КНМ) са показали, че могат да превръщат сурови изображения в мощни представяния [cite], които позволяват създаването на модели, представящи се на човешко ниво в състезания като ImageNet classification challenge [cite]. КНМ може да се разглежда като функция $CNN_{\theta_c}(I)$, която извлича характеристики за изображение I и има параметри θ_c

5.4.4 Модел за избор на действия

Агентът избира действие a във време t, когато се намира в състояние s. Решението на агента се взима благодарение на Дълбока Бейсова Невронна Мрежа (Deep Baysian Neural Network)

5.4.5 Определяне на награди

Наградата r_t за всяка стъпка t се дефинира като:

$$r = -1 + C_a$$

където C_a е новият процент покритие на код след избора на действие a. "Наказанието", количествено оценено с -1 за всяко следващо взето действие "мотивира" агента да се стреми изучи средата максимално бързо. Това спомага за намаляване на възможността за разглеждане на две съседни състояния в цикъл.

5.4.6 Архитектура на модела

- Входен слой - броят на невроните е равен на броят на пикселите в изображението, което представя средата в текущото време (обикновено 6400)
- 1-ви скрит слой - 2-мерен конволюционен слой с 3 входни канала и 16 изходни. Размерността на ядрото (kernel) е 5.
- 2-ри скрит слой - нормализиращ слой за 16-те изходни канала от предходния слой. … Повтаряме предходните 2 слоя още 2 пъти
- 7-ми скрит слой - напълно свързан (fully-connected) слой, който сплесква (flatten?) броя на измеренията до 1.
- 8-ми скрит слой - отпадащ (dropout) слой, който изключва *20%* от невроните в мрежата на всяка стъпка.
- Изходен слой - редуцира броя на невроните от предходния слой до броя на възможните действия на Агента

5.4.7 Цел/Оптимизация/Тренировка/Обучение

Целта на създадения модел е да научи апостериорното разпределение на действията в дадено състояние на средата s - формула.

Определение 1 (функция на загубата на Хубер). Казваме, че

Обучението ще се състой в минимизиране на т.нар функция на загубата на Хубер. Тя има свойствата на средна квадратична грешка, когато грешката е малка и тези на средна абсолютна грешка, когато грешката е голяма - това прави модела ни издръжлив (robust) на екстремни стойности (outliers) [cite].

5.4.8 Памет

Ще запазваме преходите от състояние s до s' при избрано действие a и получената награда r, които ще използваме за допълнително обучение на модела. Избирането на случайно подмножество от така запазените преходи

5.4.9 Оценка на модела за избор на действия

Адекватността на агента, т.е. адекватността на всички избрани от агента действия се измерва чрез т.нар мярка за “съжаление” (regret) - разликата между оптималната обща награда и получената обща награда. Оптималната награда може да се постигне, когато на всяка стъпка t, агентът избира оптимално действие a^*.

5.4.10 Намиране на апостериорно вероятностно разпределение на действията

Политиката, която използват агентите в [cite] е епсилон-лакома с намаляща стойност. Тя се отличава с това, че избира случайни действия, за да събере данни в началната фаза на обучението по-късно намаля вероятността за избиране на случайно действие. Обученият агент избира действие с максимална награда. Вместо това, може да опитаме да минимизираме несигурността (uncertainty) на нашата НМ. Това се оказва сравнително лесно ползвайки Thompson Sampling.

5.4.10.1 Thompson Sampling

Thompson sampling е политика, която насърчава агента да изследва средата в която действа, като избира действие с максимална награда, използвайки текущото си познание за средата. В нашият случай, това може да направим като симулираме стохастичен преход през НМ и изберем действието с най-висока очаквана награда.

5.5 Експерименти

Глава 6

Заключение

6.1 Нерешени проблеми

6.2 Бъдеща работа

6.3 Дискусия

Приложение 1: Някои важни вероятностни разпределения

Приложение 2: Фигури

Литература

Abraham, A. et al., 2015. GroddDroid: A gorilla for triggering malicious behaviors. In *Malicious and unwanted software (malware), 2015 10th international conference on*. IEEE, pp. 119–127.

Amalfitano, D. et al., 2015. MobiGUITAR: Automated model-based testing of mobile apps. *IEEE Software*, 32(5), pp.53–59.

Anonymous, 2018. Efficient exploration through bayesian deep q-networks. *International Conference on Learning Representations*. Available at: https://openreview.net/forum?id=Bk6qQGWRb.

Bellemare, M.G., Dabney, W. & Munos, R., 2017. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*.

Bellemare, M.G. et al., 2013. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47, pp.253–279.

Bishop, C., 2007. Pattern recognition and machine learning (information science and statistics), 1st edn. 2006. Corr. 2nd printing edn. *Springer, New York*.

Boyan, J.A., Littman, M.L. & others, 1994. Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in neural information processing systems*, pp.671–671.

Brockman, G. et al., 2016. OpenAI gym. *arXiv preprint arXiv:1606.01540*.

Chang, T.-H., Yeh, T. & Miller, R.C., 2010. GUI testing using computer vision. In *Proceedings of the sigchi conference on human factors in computing systems*. ACM, pp. 1535–1544.

Cho, K. et al., 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Choi, W., Necula, G. & Sen, K., 2013. Guided gui testing of android apps with minimal restart and approximate learning. In *ACM sigplan notices*. ACM, pp. 623–640.

Choudhary, S.R., Gorla, A. & Orso, A., 2015. Automated test input generation for android: Are we there yet?(E). In *Automated software engineering (ase), 2015 30th ieee/acm international*

conference on. IEEE, pp. 429–440.

Crites, R.H. & Barto, A.G., 1996. Improving elevator performance using reinforcement learning. *Advances in neural information processing systems*, 8.

Dahl, G.E. et al., 2012. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1), pp.30–42.

Gal, Y. & Ghahramani, Z., 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*. pp. 1019–1027.

Gal, Y. & Ghahramani, Z., 2015. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2.

Gauss, C.F., 1809. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium auctore carolo friderico gauss*, sumtibus Frid. Perthes et IH Besser.

Geman, S. & Geman, D., 1984. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6), pp.721–741.

Gergonne, J., 1815. Application de la méthode des moindres quarrésa l'interpolation des suites. *Annales de Math. Pures et Appl*, 6, pp.242–252.

Graves, A., Mohamed, A.-r. & Hinton, G., 2013. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, pp. 6645–6649.

Guo, X. et al., 2014. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in neural information processing systems*. pp. 3338–3346.

Hao, S. et al., 2014. Puma: Programmable ui-automation for large-scale dynamic analysis of mobile apps. In *Proceedings of the 12th annual international conference on mobile systems, applications, and services*. ACM, pp. 204–217.

Hastings, W.K., 1970. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1), pp.97–109.

Hinton, G.E. et al., 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hochreiter, S. & Schmidhuber, J., 1997. Long short-term memory. *Neural computation*, 9(8), pp.1735–1780.

Hoffmann, M. et al., 2009. Jacoco code coverage tool. Online, 2009.

Huang, S.-Y. et al., 2015. ABCA: Android black-box coverage analyzer of mobile app without source code. In *Progress in informatics and computing (pic), 2015 ieee international conference on*. IEEE,

pp. 399–403.

Joachims, T. et al., 1997. Webwatcher: A tour guide for the world wide web. In *IJCAI (1)*. Citeseer, pp. 770–777.

Johansson, F., Shalit, U. & Sontag, D., 2016. Learning representations for counterfactual inference. In *International conference on machine learning*. pp. 3020–3029.

Kalchbrenner, N. & Blunsom, P., 2013. Recurrent continuous translation models. In *EMNLP*. p. 413.

Kononenko, I., 1989. Bayesian neural networks. *Biological Cybernetics*, 61(5), pp.361–370.

Krizhevsky, A., Sutskever, I. & Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. pp. 1097–1105.

Kullback, S. & Leibler, R.A., 1951. On information and sufficiency. *The annals of mathematical statistics*, 22(1), pp.79–86.

LeCun, Y. et al., 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4), pp.541–551.

Legendre, A.M., 1805. *Nouvelles méthodes pour la détermination des orbites des comètes*, F. Didot.

Levine, S. et al., 2016. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39), pp.1–40.

Lin, L.-J., 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4), pp.293–321.

Machiry, A., Tahiliani, R. & Naik, M., 2013. Dynodroid: An input generation system for android apps. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering*. ACM, pp. 224–234.

MacKay, D.J., 1992. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3), pp.448–472.

Memon, A.M., 2002. GUI testing: Pitfalls and process. *Computer*, 35(8), pp.87–88.

Metropolis, N. et al., 1953. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6), pp.1087–1092.

Mnih, V., 2013. *Machine learning for aerial image labeling*. PhD thesis. University of Toronto.

Mnih, V. et al., 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*.

Mnih, V. et al., 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V. et al., 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540),

pp.529–533.

Molnar, A.-J., 2015. Live visualization of gui application code coverage with guitracer. In *Software visualization (vissoft), 2015 ieee 3rd working conference on*. IEEE, pp. 185–189.

Moreira, R.M. & Paiva, A.C., 2014. A gui modeling dsl for pattern-based gui testing paradigm. In *Evaluation of novel approaches to software engineering (enase), 2014 international conference on*. IEEE, pp. 1–10.

Moreira, R. et al., 2017. Pattern-based gui testing: Bridging the gap between design and quality assurance.

Neal, R.M., 2012. *Bayesian learning for neural networks*, Springer Science & Business Media.

Ohba, M., 1982. Software quality &equil; test accuracy× test coverage. In *Proceedings of the 6th international conference on software engineering*. IEEE Computer Society Press, pp. 287–293.

Riedmiller, M., 2005. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European conference on machine learning*. Springer, pp. 317–328.

Roubtsov, V. & others, 2005. Emma: A free java code coverage tool.

Rumelhart, D.E., Hinton, G.E. & Williams, R.J., 1985. *Learning internal representations by error propagation*, DTIC Document.

Salvesen, K. et al., 2015. Using dynamic symbolic execution to generate inputs in search-based gui testing. In *Proceedings of the eighth international workshop on search-based software testing*. IEEE Press, pp. 32–35.

Sermanet, P. et al., 2013. Pedestrian detection with unsupervised multi-stage feature learning. In *Proceedings of the ieee conference on computer vision and pattern recognition*. pp. 3626–3633.

Shi, T. et al., 2017. World of bits: An open-domain platform for web-based agents. In D. Precup & Y. W. Teh, eds. *Proceedings of the 34th international conference on machine learning*. Proceedings of machine learning research. International Convention Centre, Sydney, Australia: PMLR, pp. 3135–3144. Available at: http://proceedings.mlr.press/v70/shi17a.html.

Silver, D. & Hassabis, D., 2016. AlphaGo: Mastering the ancient game of go with machine learning. *Research Blog*.

Silver, D. et al., 2017. Mastering the game of go without human knowledge. *Nature*, 550(7676), pp.354–359.

Sundermeyer, M., Schlüter, R. & Ney, H., 2012. LSTM neural networks for language modeling. In *Interspeech*. pp. 194–197.

Sutton, R.S. & Barto, A.G., 1998. *Reinforcement learning: An introduction*, MIT press Cambridge.

Todorov, E., Erez, T. & Tassa, Y., 2012. MuJoCo: A physics engine for model-based control. In *Intelligent robots and systems (iros), 2012 ieee/rsj international conference on*. IEEE, pp. 5026–5033.

Tsitsiklis, J.N., Van Roy, B. & others, 1997. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5), pp.674–690.

Van Hasselt, H., Guez, A. & Silver, D., 2016. Deep reinforcement learning with double q-learning. In *AAAI*. pp. 2094–2100.

Werbos, P.J., 1988. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4), pp.339–356.

Yeh, C.-C. & Huang, S.-K., 2015. CovDroid: A black-box testing coverage system for android. In *Computer software and applications conference (compsac), 2015 ieee 39th annual*. IEEE, pp. 447–452.

Zhauniarovich, Y. et al., 2015. Towards black box testing of android apps. In *Availability, reliability and security (ares), 2015 10th international conference on*. IEEE, pp. 501–510.

Zhu, H., Hall, P.A. & May, J.H., 1997. Software unit test coverage and adequacy. *Acm computing surveys (csur)*, 29(4), pp.366–427.