Testing Mobile Software with Machine Learning: An Introduction

Aaron Briel
PinkLion
aaron@pinklion.ai

Chris Navrides
test.ai
chris@test.ai

Jennifer Bonine
PinkLion
jennifer@pinklion.ai

Abstract

User Interface testing has changed very little since its inception. It is still completely reliant upon knowing the state and actions available on the screen. However, this way of testing is incompatible with current development methodologies such as A/B flighting and dynamic data, as well as variations by test platform. These new methodologies present a significant challenge to the current paradigm in that the current state of the application may not be known. Platform variations contribute to automation framework bloat and test cycle time. This paper demonstrates how Machine Learning can assist in closing the gap by removing reliance on the state of the application and its underlying DOM.

Biography

Aaron Briel is a seasoned engineer with over 15 years of experience in development, QA automation, DevOps, cybersecurity, AI and Machine Learning. Applications he built for a Fortune 100 company produced data presented in Congressional hearings to demonstrate post-breach compliance progress. His work in AI started at the University of Minnesota in the early 2000's, where he developed a genetic algorithm capable of generating hip-hop drum sequences. Aaron is currently Chief AI Architect for PinkLion and a graduate student in ML/AI at Georgia Tech.

Jennifer Bonine has been a strategic C-suite advisor for some of the most respected Fortune 100 companies in the areas of technology strategic roadmap, Quality Assurance, organizational development, and change management. She is a well-known speaker, teacher and trainer, keynoting at both national and international technology conferences. Jennifer has held executive level positions at leading development and quality engineering teams for Fortune 100 companies in several Industries. She is currently the CEO of PinkLion.

Chris Navrides has worked in the software automation and testing industries for 10 years at large tech companies such as Google, Microsoft and Dropbox. He holds patents for temporal content selection and entity-based summarization from his work at Google. Chris is currently a Software Engineer at test.ai.

1 Introduction

User Interface (UI) testing was originally developed when applications were far more static and development cycles were on the order of quarters to years. Tests that made assumptions regarding the state of the UI and underlying application were logical at the time. Webpages and applications today, however, are dynamically changing based upon Machine Learning (ML) models determining the optimal items to place on them. An example of this is content customized for users on Amazon. New versions of native applications are released every few weeks filled with A/B experiments.

Test authors and framework designers are having an increasingly hard time writing reliable tests while using technology and techniques developed 15 to 20 years ago. While the introduction of automated testing into the development lifecycle promised to more closely accommodate increasingly agile methodologies, the holy grail of Continuous Delivery has rarely been realized much less sustained.

A traditional automated testing solution relying on the DOM is likely to break when faced with modified content or elements, whereas an AI/ML driven testing solution can demonstrate flexibility. For example, an image-based element classification approach has no dependency on the DOM. Renamed elements with the same images would have no effect on this class of tests, and even visually modified elements might only require a few training iterations.

What if minor details in the flow of a specific validation change? A traditional automated test would likely fail, even with bloated and fragile conditionals. However, an ML-based approach to navigation would intelligently find the correct path to the pertinent location required for the test.

Automated testing must adapt and become smarter in order to meet the business and quality challenges of the modern software development lifecycle. Development of traditional automation solutions is often so time consuming that the final product, once completed, is no longer compatible with the application it was initially meant to test. Maintenance of these frameworks consistently exceeds the release cycle timeline itself, rendering the approach infeasible in the current fast-paced reality of software development. Hard coding all of the possible scenarios to make “intelligent tests” is one approach but is inflexible and doesn’t scale. Machine Learning is designed specifically for challenges such as these by offering solutions more compatible with rapid release cycles.

2 What is Machine Learning?

Machine Learning (ML) is a software development technique that allows a program, often called an agent, to generalize about data it has not yet seen. Instead of programming a set of predefined commands, a model or computer brain is trained by presenting the agent with sets of examples or input to learn from. The model essentially provides the agent with confidence percentages for behaviors or actions to take given specific states it is presented with. Using this model allows it to handle both previously seen, and unseen challenges in the future.

2.1 Supervised Learning

The approach of training by providing feedback based on existing examples is known as supervised learning, or function approximation. A simple example might include an image classifier. The supervised aspect of learning consists of the fact that a human provides feedback as to the correctness of the agent's predictions. This feedback could consist of well-labeled training data. The image classification example is relevant to test automation not only for validations but also for element identification. An example of image classification is an agent that can classify number images. In training, it is presented with thousands of examples of hand written numbers with correct labels. Depending on the number of samples, training sessions, and other factors, this can produce an agent that is able to correctly classify well written test samples. (Briel, 2018).

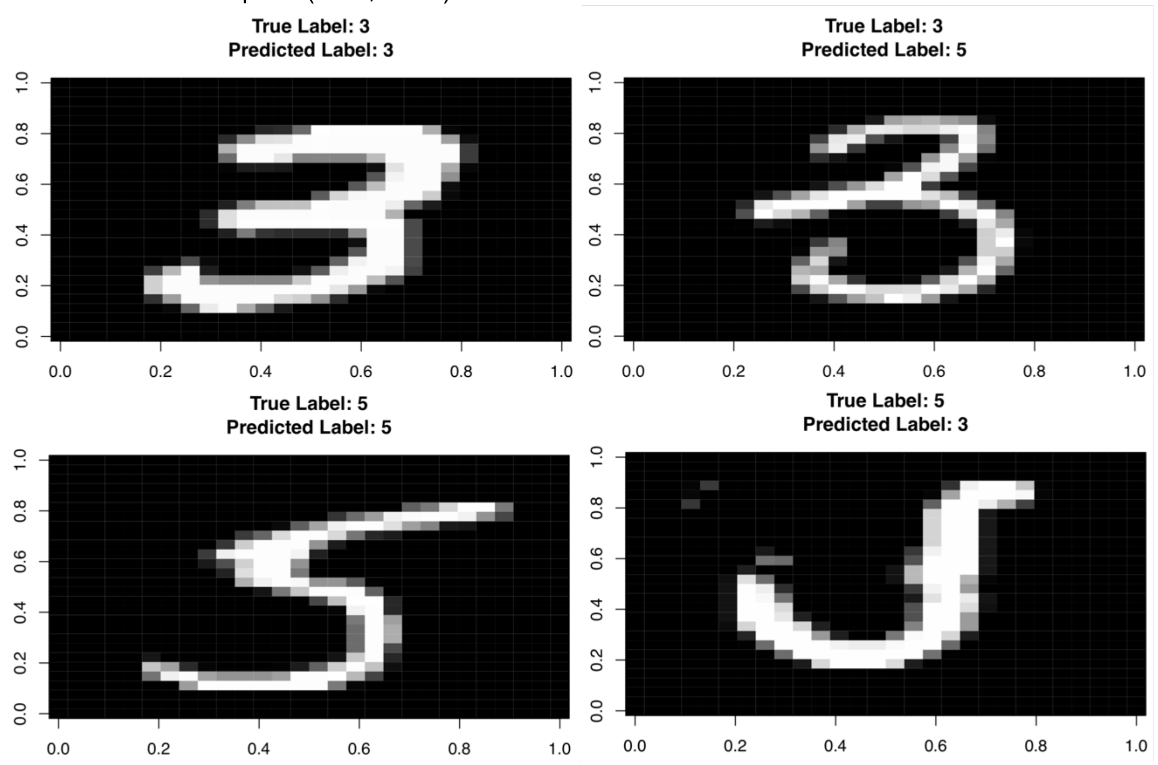

Figure 1: Test samples and predictions given by a number classification agent

2.2 Unsupervised Learning

In unsupervised learning the agent only has whatever data is presented to it without any behavioral directives. One popular unsupervised learning technique is called Clustering. Clustering is an exploratory approach to data analysis that allows for the discovery of previously unknown relationships in data. The EM Algorithm is often used as the basis for clustering algorithms, where parameters are estimated to describe an underlying probability distribution when only part of the data produced by the said distribution is observable. Dissimilarity in data objects becomes apparent in Clustering, making it an optimal technique for anomaly detection. This is highly relevant in the testing context, as anomaly detection could be used to discover defects in applications.

Deep Learning has provided many opportunities to explore unsupervised learning. DeepMind, for example, introduced its AlphaZero system that was able to teach itself how to play chess, shogi, and Go, defeating previous world-champion AI applications in the process (Silver, 2018).

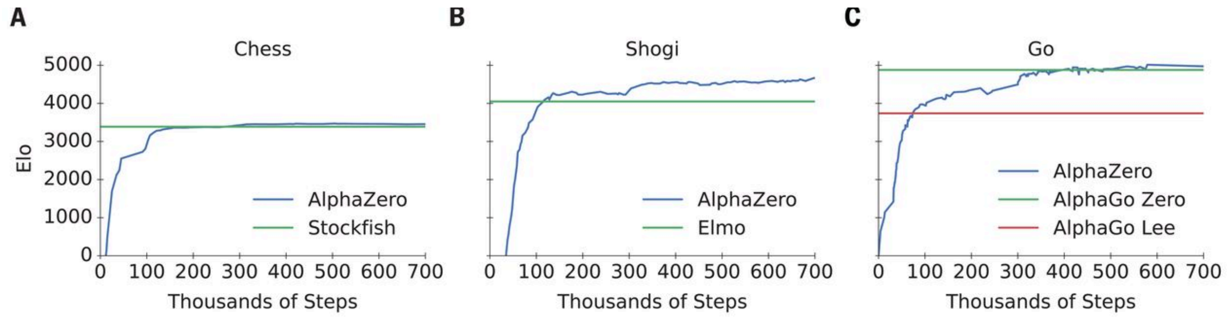

Figure 2: DeepMind's AlphaZero system training for 700,000 steps, showing exponential performance increases after learning occurs

2.3 Reinforcement Learning

Reinforcement Learning (RL) is a hybrid ML approach that can leverage aspects of supervised and unsupervised learning. Constructing a model and optimal behavioral in RL becomes a balancing process of exploring the environment and exploiting rewards. The Q-table contains values for different states, and is updated based on the classic Bellman Equation (Littman, 1996):

$$Q_i(s, \vec{a}) = (1 - \alpha)Q_i(s, \vec{a}) + \alpha[R_i + \gamma V_i(s')] \tag{1}$$

The maximum Q-values for actions and states represent the optimal policy of the agent, or the best action to take given a specific state. A fundamental property of the Q-update function is that the values are updated recursively as the agent progresses through its tasks. A discount factor (γ) is applied to future rewards, allowing flexibility with respect to how much an agent values immediate versus delayed actions. Introduction of the learning rate (α) ensures eventual convergence of the Q-updates.

An example of an agent that uses RL might be a robot vacuum cleaner. As the robot moves a specific distance without hitting an obstacle it might be presented with a positive reward, while hitting an obstacle or staying in place might present it with a negative reward. A reward system in this context might be set up to emulate walking on a hot beach. The agent wants to keep moving around in order to prevent its feet from burning, and stepping on broken glass would teach it to avoid such obstacles in the future.

3 Element Classification

The easiest way to leverage ML models in testing is through element classification. Similar to how a human can look at an app they have never seen before and determine what is a menu button or search button, the same can be achieved by a well-trained ML model. This is something that traditional test automation is simply incapable of achieving because there is no standard naming or app design convention for commonly occurring elements such as search icons and menus. For example, search icons aren't always going to have an ID of "search_icon". In the ML scenario, an image of the page might be broken up into sub-images which constitute element contenders. The human tester may go in and provide feedback on the labels provided to the elements by the agent. Rewards are allocated based on these adjustments and the model is updated appropriately. Over time, the model is able to correctly predict the label to assign to elements it has never seen.

3.1 Shopping cart classifier

To illustrate element classification, we will present an example of the steps that would be taken in building a shopping cart classifier. One would first obtain a significant number of shopping cart images. These will be used to train a model on how to classify the element icon.

A large number of samples is necessary in the data selection process in order to cover the wide range of what constitutes the shopping cart element. An example of the variance encountered includes an empty cart compared to one with items in it, possibly denoted with a number over the top. Other examples needed to increase the diversity of items presented include carts enclosed in a circle and ones with varying background colors. The reason such a wide range of examples is needed is so that the agent is able to generalize and classify cart examples that have not yet been seen. Fig. 1 illustrates the extent of diverse cart items that might be considered (Frost, 2019).

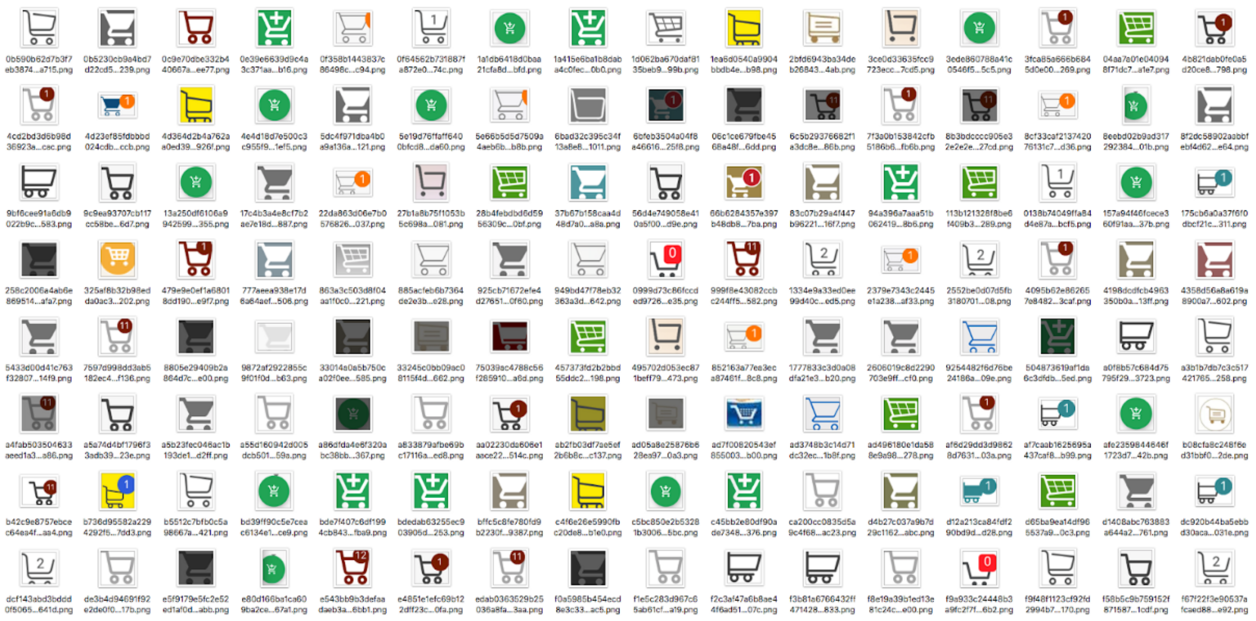

Figure 3: A sample of cart images that might be used to train the model

4 Reinforcement Learning for Exploratory Testing

Another area where Machine Learning can augment the testing effort is with exploratory testing. RL would be an appropriate contender for this, as it involves a process discussed previously of exploration of the environment. When an RL agent using Q-learning crawls a webpage, it will initially execute random actions leading to exploratory application behaviors. It's Q-table in this case could be populated with page element and action combinations, Actions like 'click' executed on certain elements such as buttons may have a higher reward than a 'swipe' action on a text field, and its Q-table will be updated accordingly.

With respect to element labelling, when an agent crawls an app it may apply some correct and some incorrect labels. When the human tester provides feedback to the agent as to the correctness of its labeling, a strong positive reward is assigned in the cases of elements labeled correctly, while a negative reward is allocated to incorrect labels. Over a few iterations of this training the agent improves its labeling process for subsequent crawls.

This approach can also apply to paths taken by the agent within the application. A strong practical benefit of such an RL approach in test development is that the exploratory phase may lead to testing paths that may have never been considered. Positive rewards can be distributed in the case where the agent discovers novel screens within the application, while negative rewards might be given in situations where progress in discovery has stalled or the agent has browsed to an external application. Situations such as these are analogous to the beach walking example earlier, where the stalling corresponds to standing on hot sand, which browsing away from the app is similar to stepping on broken glass. The fact that the approach provides value in automating a previously expensive human task is a strong ancillary benefit.

5 ML for Testcase Creation

We saw that it is indeed possible for RL-driven exploratory testing to produce novel test flows. It would thus by no means be unrealistic to postulate the feasibility of an agent capable of generating test cases. Along with element classification, another model could contain more abstract information related to common flows of application behavior. Such a system that combines these areas of functionality has indeed been proposed (Santiago, 2018). In this proposal, test flows are modeled as "sequences", where specific phrases or words can be processed as application behaviors and others constitute elements.

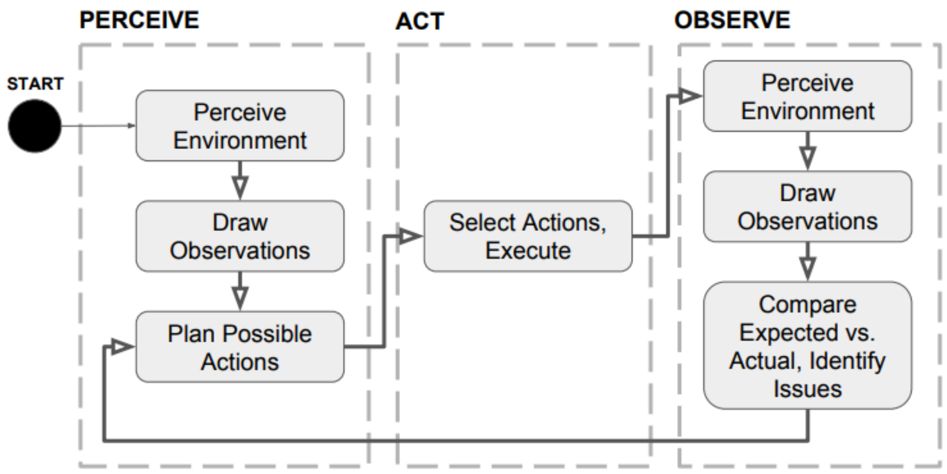

Figure 4: Modeling testing as a sequence or test flow

The elements training discussed previously could occur within a "web classifier" sub-system, with additional human feedback provided to a test flow generator. A grammar module could then convert said flows to a "test generator" that produces executable tests.

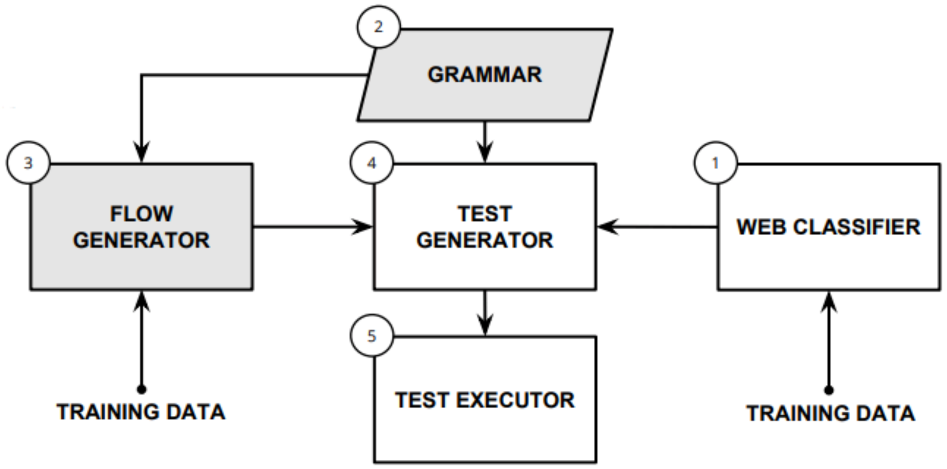

Figure 5: AI driven test generation system

Test cases themselves could be simplified to only include validations instead of sequences of prior steps. When a human engineer is asked to test that a guest login page loads, we are not concerned with how she got there. ML presents us with the ability to replicate this human-like behavior, where navigation to a specific page for validation is possible without having to explicitly code all steps it may take to get there –

steps which are likely to change and break a traditional automation test. This is illustrated in Fig. 2 and Fig. 3. A traditional automation approach would require conditional logic to deal with the environment loading page and the android-specific popup, while an ML approach would not.

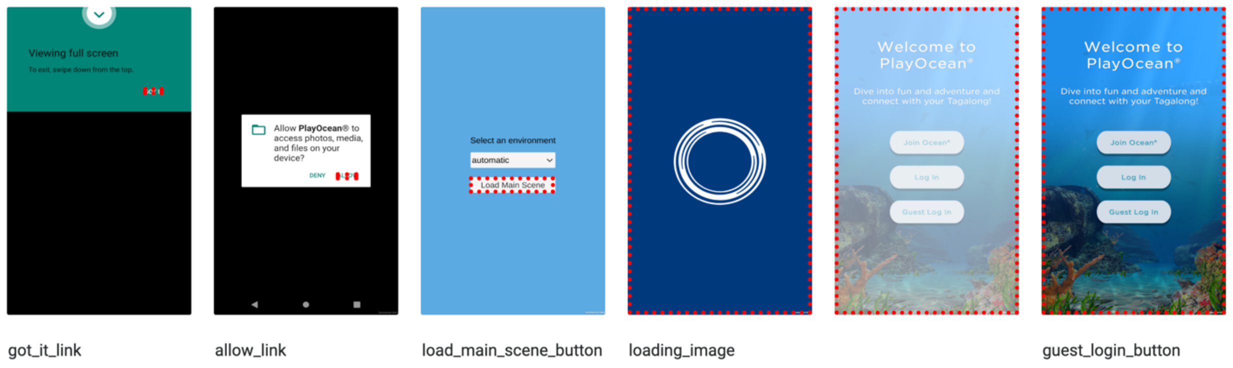

Figure 6: Running a test to validate that the welcome page loads on a certain android device

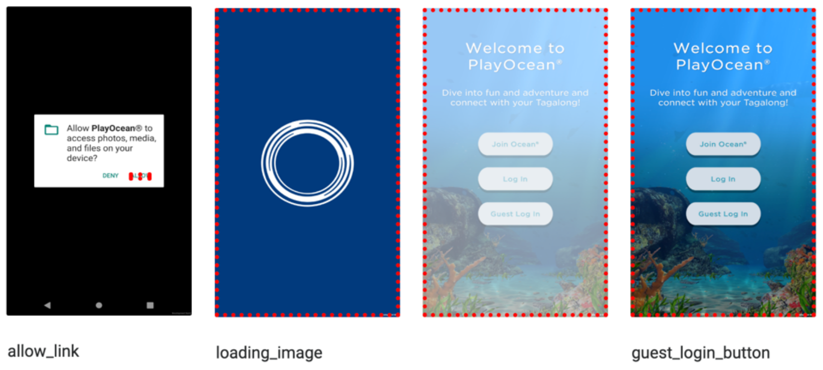
Allow PlayOcean® to access photos, media, and files on your device?
DENY
Welcome to PlayOcean®
Dive into fun and adventure and connect with your Tagalong!
Join Ocean®
Log In
Guest Log In
allow_link
loading_image
guest_login_button

Figure 7: The same test run on a different device and environment

6 ML Solutions in Practice

PinkLion and test.ai have applied the techniques above to thousands of applications. Reinforcement Learning in particular is a technique frequently leveraged, wherein a process called “crawling” is executed, consisting of exploratory testing and element extraction. This is analogous to the initial phase of exploration in RL, where the Q-table is populated with values for actions and corresponding states. Many defects are often discovered in this initial phase, and release dates have frequently been pushed forward as a result of this process, even before any customized test flows have been developed.

7 Challenges and Limitations

As with any technology there are limitations, problems and challenges. One limitation is that complex use cases still require customized automation flow development. Although test case creation was briefly touched, it is nowhere near the point of being able to capture complete business requirements. A paradigm shift in application testing focused on more granular validations combined with more simplified test case creation may result in a convergent satisfaction of this limitation.

Maintenance and bias can also introduce challenges. If elements change to a point where the model no longer recognizes them, additional training may be necessary. Although this may initially take longer than a minor code modification in a traditional automation script, it could provide an added benefit of making the model more generalizable and adaptable to future element changes.

Finally, it is possible that a human may introduce bias into a model. This bias could be the result of an error in classification. In this case repeated training may be necessary in order for the model to balance itself to the correct classification. This is a small price to pay for the benefits provided by novel element classification.

8 The Future

Industries are disrupted by technological advancements, forcing a change in the way humans are needed. QA and automation are no different, and the rapid introduction of AI and ML into this space will require a paradigm shift in the approaches taken. As applications are ultimately consumed by humans, human input will be needed in their development and testing. ML will first replace the need for automation of menial and repetitive tasks but will eventually evolve to higher level test creation. Human feedback will be needed to guide these systems, although even this might eventually be provided through beta or crowd-sourced behavior analysis.

References

Briel, Aaron. 2018. “Image Classification using Logistic Regression with Stochastic Gradient Descent.” https://github.com/aaronbriel/logistic-regression/blob/master/logistic_regression.pdf (accessed July 31, 2019).

Frost, Nick. 2019. “Training Data for Our Open-Sourced AI Classifier for Appium.” https://www.test.ai/blog/training-data-for-app-classifier/ (accessed August 1, 2019).

Littman, Michael. 1994. “Markov games as a framework for multi- agent reinforcement learning.” In Proceedings of the Eleventh International Conference on Machine Learning (July 1994), pages 157–163.

Santiago, Dionny. 2018. “AI Driven Test Generation. Machines Learning from Human Testers.” In 36th Annual Pacific NW Software Quality Conference. http://uploads.pnsqc.org/2018/papers/119-Santiago-AI%20Driven%20Test%20Generation.pdf
(accessed August 1, 2019).

Silver, David. 2018. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play.” Science: Vol. 362, Issue 6419 (Dec 2018), pages 1140-1144