



UNIVERSIDAD NACIONAL DE COLOMBIA

Intelligent Adaptive Testing Using Machine Learning Techniques

Arcesio José Cáliz Viñas

Universidad Nacional de Colombia
Faculty of Science, Department of Statistics, Bogotá, Colombia
2022

Intelligent Adaptive Testing Using Machine Learning Techniques

Arcesio José Cáliz Viñas

Degree work presented as a partial requirement to qualify for the Title of :
Master in Statistics

Director(a):
Álvaro Mauricio Montenegro Díaz Ph.D

Line of research:
Reinforcement learning and Multidimensional Item Response Theory
Universidad Nacional de Colombia
Faculty of Science, Department of Statistics, Bogotá, Colombia
2022

Resumen

Test Adaptativos Inteligente Usando Técnicas De Aprendizaje De Máquina.

Los test adaptativos inteligentes permiten realizar evaluaciones que reducen el número de preguntas, mejoran la estimación y se adaptan a las respuestas de la persona. Una decisión a tomar en el diseño de estas pruebas, es el método para escoger la siguiente pregunta. No existe un método que demuestre ser el mejor desde la perspectiva de las mejoras implicadas en un test adaptativo. En este trabajo exploramos el uso de algoritmos de aprendizaje por refuerzo en conjunto con algoritmos de aprendizaje profundo para la escogencia de las preguntas. Los resultados muestran que bajo ciertas condiciones y dependiendo del algoritmo utilizado, estos nuevos métodos logran resultados competentes en términos del número de preguntas hechas y la exactitud de la estimación comparándolos con los métodos estadísticos tradicionales.

Palabras clave: Aprendizaje por refuerzo, Aprendizaje de Máquina, Redes Neuronales, Test Adaptativos, Teoría De Resupuesta al Ítem..

Abstract

Intelligent Adaptive Testing Using Machine Learning Techniques.

Intelligent adaptive tests allow to perform evaluations that reduce the number of questions, improve the estimation, and adapt to the person's answers. A decision to make in the design of these tests is the method for choosing the next question. There is no method that proves to be the best from the perspective of the improvements implied by an adaptive test. In this work, we explore the use of reinforcement learning algorithms in conjunction with deep learning algorithms for question choice. The results show that under certain conditions and depending on the algorithm used, these new methods achieve competent results in terms of the number of questions asked and the accuracy of the estimation compared to traditional statistical methods.

Keywords: Reinforcement Learning, Machine Learning, Neural Networks, Adaptive Tests, Item Response Theory

List of Figures

3-1. Neural Network Model Used For Q-learning	24
3-2. Average Test Return During Training process	25
3-3. Kernel Density Estimator of <code>number_of_items</code> Asked Per Model For Test Set	26
3-4. Kernel Density Estimator of <code>number_of_items</code> Per Model For Test Set . . .	27
3-5. Count Of Best Distance Comparing Multiple Strategies	28
3-6. Bi-variate Histogram Of Models With Most Accurate Estimation	30

List of Tables

3-1. Count Of Cases When Strategy Got The Lowest Euclidean Distance	28
3-2. Count Of Cases When Strategy Got The Lowest Number Of Items Asked . .	29
3-3. Best Models According To The Sum Of score	30
C-1. Reinforcement Learning Hyper-parameters	39

1. Item Response Theory

1.1. Introduction

Item response theory (IRT) is a paradigm from psychometrics that defines a general framework that studies the design and analysis of the interactions between persons and test items using statistical models. IRT seeks to separate item parameters and population sampled so both can be studied separately [Chalmers, 2012]. Classical test theory (CTT) is the early procedure for test analysis where the goal was either to identify an item difficulty or a test-taker ability to answer correctly. Unlike CTT, IRT focuses on the items while CTT focus is on the test. CTT can be seen as the rudiments of IRT, where some assumptions are still used. The idea of having a mathematical function that models the probability of correctness of a question answered by a person is a basis of the IRT. Early development of IRT began on the 1950s and 1960s by Fredrick Lord, Georg Rasch, and Paul Lazardsfeld but IRT popularity raised during the 1980s when the first models and estimation methods were proposed and personal computers were accessible.

Items, according to [Reckase, 2009] can be separated as stimulus material and a form of answering, the former asks a specific question and the latter yields a response. Items are categorized according to the form of its parts, for example open-ended items like writing an essay, multiple choice items popular in standardized test for college admissions, mathematical demonstrations, etc. Items sharing the same type of answer should have different stimuli; a key aspect of IRT is the idea of items having different difficulties, thus items must have complex parameters to differentiate each other, and the interaction between items parameters and the person capabilities reflects the correctness of the answer.

People, like items, differentiate each other in numerous ways. In the context of IRT people have cognitive skills, knowledge, interests, and capabilities that are supposed to be estimated. These parameters allow ordering the people in a continuum, and the number of dimensions to consider, depends on the test and the population sample. Usually, it is impossible to directly observe this capabilities so they must be inferred from the responses of each item, due to this phenomenon, the list of aptitudes is known as latent traits. Multidimensional Item Response Theory (MIRT) uses the concepts of IRT and generalizes models to handle latent traits in a multidimensional space.

Items parameters characterize their difficulty, discrimination power, and the probability of guessing the correct answer. Some models include additional parameters depending on the item. On the other hand, latent traits are meant to describe an ability required to answer correctly, but some assumptions are needed to reduce the number of dimensions, for example the assumption that people fully understand the language and specialized vocabulary of the stimulus material. Both items parameters and latent traits are the inputs to calculate the probability of getting the right answer.

This chapter describes models used in IRT, then the explanation of procedures to estimate the latent traits, and finally the concepts of computerized adaptive testing (CAT).

1.2. Models

Every test is different: skills to be evaluated, the number of items, difficult of the items, time to accomplish the test, and the test-takers are some examples of the visible differences. To model every item of every test is an arduous task, so the need to implement assumptions is more notorious. Due to the large number of cognitive skills, we must assume a finite number of latent traits. Second, the persons latent traits remain constant over the evaluation, i.e. the answer of the items relies exclusively on the person abilities, which is highly restrictive as people might learn something by interacting with previous items or by cheating. The probability increases monotonically depending on the underlying latent trait. Finally, each individual has a true location on the continua.

Models depend on the type of test and the assumptions on the latent traits. The existence of correlations between cognitive skills leads to compensatory models in which the latent traits are not seen entirely separated, but on the contrary, having a high value on one skill increases the chances of answering correctly a question that is focus on the second skill. Either polytomous or dichotomous question, the most widely used model is the 3-parameter model logistic model (3PL). This model was initially proposed by [Lord, 1980] for the uni-dimensional case. We will focus on dichotomous items coded as 0 or 1 for incorrect and correct answers, respectively.

Let $i = 1, 2, \dots, N$ represent each person, $j = 1, 2, \dots, n$ the test items, X_{ij} represent the score of the i -th person on the j -th item, suppose m latent traits $\boldsymbol{\theta}_i = (\theta_{i1}, \theta_{i2}, \dots, \theta_{im})$ associated to m item slopes $\boldsymbol{\alpha}_j = (\alpha_{j1}, \alpha_{j2}, \dots, \alpha_{jm})$, d_j the intercept and dividing it by an element of the items slopes gives the relative difficulty of the corresponding coordinate, $\tau_j \in (0, 1)$ is the guessing parameter and D is a scaling parameter which according to [Reckase, 2009] a value of 1,702 is used to make the model closely to the traditional ogive metric, the probability of correctly answer a dichotomous items is given by:

$$P(X_{ij} = 1 | \boldsymbol{\theta}_i, \boldsymbol{\alpha}_j, d_j, \tau_j) = \tau_j + \frac{1 - \tau_j}{1 + \exp[-D(\boldsymbol{\alpha}_j^T \boldsymbol{\theta}_i + d_j)]} \quad (1-1)$$

For the uni-dimensional case, most of the parameters keep the same definition although d_j is known as difficulty. Modifying 3PL by setting $\tau_j = 0$, yields the multidimensional generalization of the 2-parameters model proposed by [Birnbbaum, 1968]. A different model is explained in [Kelderman and Rijkens, 1994], which is used to handle items that are divided into sub-items, each providing a partial credit score. Partially compensatory models proposed by [Simpson, 1978], where latent traits are not correlated, thus lacking one specific skill entirely reduces the probability of correctly answering an item. The focus of this document is on the 3PL.

1.3. Latent Traits Estimation

MIRT procedures are divided in exploratory and confirmatory item analysis. Exploratory seeks to model the probabilities as a function of the item parameter and the latent traits, while in confirmatory the number of dimensions is unknown and are estimated empirically.

Regarding person's parameters, assuming that all item parameters are calibrated and a set of items were administered to an examinee whose responses were collected, three methods are used to estimate the location of the latent traits: maximum likelihood (ML), maximum a posteriori likelihood (MAP), and least squared criterion. The following document will not cover least squared; additional information can be found in [Reckase, 2009].

Summarizing all items parameters as $\boldsymbol{\Psi}$, let P be the probability function of correctly answering the option k on item j , and $\mathbf{X}_i = (X_{i1}, X_{i2}, \dots, X_{in})$ represent the scores of the $j = 1, 2, \dots, n$ items for the i -th person, the likelihood function is:

$$L(\boldsymbol{\theta}_i | \boldsymbol{\Psi}, \mathbf{X}_i) = \prod_{j=1}^n \prod_{k=0}^1 P(X_{ij} = k | \boldsymbol{\theta}_i, \boldsymbol{\Psi})^{\chi_{jk}} \quad (1-2)$$

Where χ_{jk} is an indicator function to filter the probability of the chosen k :

$$\chi_{jk} = \begin{cases} 1 & \text{if } X_{ij} = k \\ 0 & \text{otherwise} \end{cases} \quad (1-3)$$

Since we are considering the dichotomous case, (1-2) results in:

$$L(\boldsymbol{\theta}_i | \boldsymbol{\Psi}, \mathbf{X}_i) = \prod_{j=1}^n P(X_{ij} = 0 | \boldsymbol{\theta}_i, \boldsymbol{\Psi})^{x_{j0}} * P(X_{ij} = 1 | \boldsymbol{\theta}_i, \boldsymbol{\Psi})^{x_{j1}} \quad (1-4)$$

Maximizing likelihood can be done analytically or using computational approximations. To differentiate a product of functions is computationally expensive, the most common solution as [Casella and Berger, 2002] mention is to use the natural logarithm of the likelihood functions since logarithm is a strictly increasing function in $(0, \infty)$, implying:

$$\arg \max_{\boldsymbol{\theta}_i} L(\boldsymbol{\theta}_i | \boldsymbol{\Psi}, \mathbf{X}_i) = \arg \max_{\boldsymbol{\theta}_i} \log(L(\boldsymbol{\theta}_i | \boldsymbol{\Psi}, \mathbf{X}_i)) = \arg \max_{\boldsymbol{\theta}_i} l(\boldsymbol{\theta}_i | \boldsymbol{\Psi}, \mathbf{X}_i) \quad (1-5)$$

If there is no variability in the response vector, i.e $SD(\mathbf{x}) = 0$, the maximum-likelihood estimator will tend to ∞ , which is the case when few items have been answered. ML estimation considers $\boldsymbol{\theta}$ a quantity unknown but fixed and looks for the vector of parameters $\boldsymbol{\theta}$ that yields the highest probability, given the observed scores. MAP estimation uses ML concepts but treats $\boldsymbol{\theta}$ as the realization of a random variable Θ whose distribution function is known and referred as *prior distribution* $g(\boldsymbol{\theta})$. Under Bayesian approach let X_1, X_2, \dots, X_n be a random sample indexed by $\boldsymbol{\theta}$ and the vector of constant items parameters $\boldsymbol{\Psi}$, the *posterior distribution* is:

$$P(\boldsymbol{\theta} | \mathbf{X}, \boldsymbol{\Psi}) = \frac{P(\mathbf{X} | \boldsymbol{\theta}, \boldsymbol{\Psi})g(\boldsymbol{\theta})}{\int_{\Theta} P(\mathbf{X} | \mathbf{v}, \boldsymbol{\Psi})g(\mathbf{v})d\mathbf{v}} \quad (1-6)$$

And the MAP estimator is found by solving:

$$\hat{\boldsymbol{\theta}}_{MAP}(\mathbf{x}) = \arg \max_{\boldsymbol{\theta}} P(\boldsymbol{\theta} | \mathbf{x}, \boldsymbol{\Psi}) \quad (1-7)$$

$$= \arg \max_{\boldsymbol{\theta}} \frac{P(\mathbf{x} | \boldsymbol{\theta}, \boldsymbol{\Psi})g(\boldsymbol{\theta})}{\int_{\Theta} P(\mathbf{x} | \mathbf{v}, \boldsymbol{\Psi})g(\mathbf{v})d\mathbf{v}} \quad (1-8)$$

$$= \arg \max_{\boldsymbol{\theta}} P(\mathbf{x} | \boldsymbol{\theta}, \boldsymbol{\Psi})g(\boldsymbol{\theta}) \quad (1-9)$$

From [Fisher, 1925], we know that the quadratic approximation of the likelihood function is an acceptable measure of precision of the estimates. Using the result of [Chalmers, 2016], the computation of the variance of $\boldsymbol{\theta}$ is estimated using the Hessian matrix:

$$\text{var}(\hat{\boldsymbol{\theta}}) = - \left(\frac{\partial^2 l(\hat{\boldsymbol{\theta}} | \mathbf{x}, \boldsymbol{\Psi})}{\partial \boldsymbol{\theta} \boldsymbol{\theta}^T} \right)^{-1} \quad (1-10)$$

The standard errors of each component of $\hat{\theta}$ are obtained by taking the square-root of the diagonal values of the covariance matrix. In MIRT context, the prior density function is usually assumed to be a multivariate normal distribution with mean vector μ_{θ} and covariance matrix Σ_{θ} . This optimization problem is solved using computational resources and algorithms like BFGS, for more information see [Fletcher, 1987].

1.4. Computerized Adaptive Testing

Computerized adaptive testing (CAT) is a methodology to construct tests that quickly adapt item selection based on the examinee's latent traits. At a high level, CAT is a cycle of (1) estimating the location, (2) selecting an item from an item bank that maximizes some criteria, (3) administer the item and re-estimates the location, and (4) check whether the test should be finalized based on stopping criteria. The cycle begins again if the stopping criterion is not met. The goal is to effectively select items that provide information about the person location [Reckase, 2009]. Even though the theory and practice of item selection and estimations are still evolving, according to [Frey et al., 2016] the efficiency of CAT is substantially higher than non-adaptive sequential if correlations between latent traits are considered in item selection for ability estimation.

The counterpart of CAT is sequential testing or standardized linear test in which the number of items is fixed and the examinee must answer every item to finish the test; it is widely used in college admission tests such as Scholastic Assessment Test (SAT) and other certifications exams. CAT environment is a bit more complex than traditional linear testing as both estimation and item selection happen at the same time the person is being evaluated, therefore requiring computational resources and specific calculations in real time.

Each component the CAT procedure must be carefully decided when designing, we begin by covering some ideas found in different sources:

- *Stopping rules.* We start by this component because is more intuitive than other parts. The most clear rule is to stop the CAT when a fixed length of items has been scored. Variable length on the other hand, is a possible approach by establishing a metric to monitor after administering each item, for example the accuracy of the estimation using (1-10) or the convergence of the estimates. Mixing both approaches is widely used; terminating the CAT when any criterion is reached.
- *Item bank.* The design of the item bank for a CAT is critical for its proper functioning [Reckase, 2009]. Items must be properly calibrated beforehand (can be done using IRT). Even though there is no clear formalization for the bank design, some arbitrary rules

used: avoid including items with high usage rates in previous tests, the pool should have a large set of items; a couple of times the highest possible number of items to be asked seems a fair number, and items must have enough variability in every direction, i.e. different levels of relative difficulty per ability should be reflected.

- *Item selection.* Probably the most complicated part of a CAT, most methods rely on maximizing some criteria using the Fisher information matrix. The following chapter will discuss the state-of-the-art.
- *Estimation.* Typically, estimation methods are maximum likelihood, weighted likelihood, Bayesian estimation including maximum a posteriori (MAP) and expected a posteriori (EAP). Bayesian methods have shown better performance when having few scores. To increase the variability of the responses and dwindle the possibility of estimated an infinite value, people may consider using a pre-CAT.
- *Pre-cat (optional).* Refers to a stage before starting the CAT where a small selection of items is used to give a starting point to the estimation. It is up to the designer the method for selecting items in the pre-cat.

1.4.1. Next Item Selection

Computerized adaptive testing heavily relies on item selection, having the best estimation procedure makes little impact if the person scores do not truly reflect the true location of the latent traits. For the uni-dimensional case, selecting a question based on the scores and estimates is a simpler task as all items are easily ordered according to their difficulty, contrary to the multidimensional case where there is no obvious way to select the next items.

Mostly all selection rules are based on the maximization of some criteria for example the maximization information in a direction with minimum information developed by [Reckase, 2009], Kullback-Leibler information [Veldkamp and Linden, 2002], methods that use the Fisher information matrix like D-Rule [Segall, 1996], A-rule, and T-rule. There is no one-size-fits-all as [Mulder and van der Linden, 2008] mentions, each of these might perform better than other in different situations.

D-Rule

D-Rule, initially proposed by [Segall, 1996], uses the determinant of Fisher information matrix to find the item that minimizes a confidence interval. The Fisher information matrix is defined as:

$$\mathcal{F}(\boldsymbol{\theta}) = -\mathbb{E} \left(\frac{\partial^2 l(\boldsymbol{\theta} | \mathbf{x}, \boldsymbol{\Psi})}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T} \right) \quad (1-11)$$

Due to local independence the information provided by new items can be calculated by adding the information matrix of that specific item:

$$\mathcal{F}_{J+m}(\hat{\boldsymbol{\theta}}) = \mathcal{F}_J(\hat{\boldsymbol{\theta}}) + \mathcal{F}_m(\hat{\boldsymbol{\theta}}) \quad (1-12)$$

And the variant when using a prior distribution information [Segall, 1996]

$$\mathcal{F}_{J+m}(\hat{\boldsymbol{\theta}}) = \mathcal{F}_J(\hat{\boldsymbol{\theta}}) + \mathcal{F}_m(\hat{\boldsymbol{\theta}}) - (\partial^2 \log(\phi(\boldsymbol{\theta})))^{-1} \quad (1-13)$$

D-rule declares that the items selected is the one that maximizes the determinant of the Fisher information matrix. The result comes from a relationship between the matrix and the confidence region around the estimates of $\boldsymbol{\theta}$ by [Anderson, 1984]. When the determinant is maximized, the volume of the confidence region is minimized.

$$\text{D-rule} = \max \left(\left| \mathcal{F}_{J+1}(\hat{\boldsymbol{\theta}}) \right|, \left| \mathcal{F}_{J+2}(\hat{\boldsymbol{\theta}}) \right|, \dots, \left| \mathcal{F}_{J+M}(\hat{\boldsymbol{\theta}}) \right| \right) \quad (1-14)$$

A-Rule, T-Rule, E-Rule

T-rule, A-rule, and E-rule also rely on the Fisher information matrix. T-rule select items that increase the average unweighted information about latent traits [Chalmers, 2016], which is the same as selecting the item that maximizes the trace of the Fisher information matrix. A-rule minimizes the sum of the asymptotic sampling variance of the estimates by selecting the item that minimizes the trace of the inverse of the Fisher information matrix [Mulder and van der Linden, 2008]. E-rule maximizes the smallest eigenvalue of the information matrix; minimizes generalized variance of the ability estimator along their largest dimension [Mulder and van der Linden, 2008]

$$\text{T-rule} = \max \left(\text{tr} \left(\mathcal{F}_{J+1}(\hat{\boldsymbol{\theta}}) \right), \text{tr} \left(\mathcal{F}_{J+2}(\hat{\boldsymbol{\theta}}) \right), \dots, \text{tr} \left(\mathcal{F}_{J+M}(\hat{\boldsymbol{\theta}}) \right) \right) \quad (1-15)$$

$$\text{A-rule} = \max \left(\text{tr} \left(\Sigma_{J+1}(\hat{\boldsymbol{\theta}}) \right), \text{tr} \left(\Sigma_{J+2}(\hat{\boldsymbol{\theta}}) \right), \dots, \text{tr} \left(\Sigma_{J+M}(\hat{\boldsymbol{\theta}}) \right) \right) \quad (1-16)$$

Kullback-Leibler Information

This alternate method was proposed by [Chang and Ying, 1996] for the uni-dimensional case, and generalized by [Veldkamp and Linden, 2002]. Let $\boldsymbol{\theta}_0$ be the true location of the person, and $\boldsymbol{\theta}_{k-1}$ another possible location after $k - 1$ items, the Kullback-Leibler information is defined as:

$$K_{ik}(\hat{\boldsymbol{\theta}}_{k-1}, \boldsymbol{\theta}_0) = \mathbb{E} \left[\log \frac{L(\boldsymbol{\theta}_0 | \boldsymbol{\Psi}, \mathbf{X}_i)}{L(\hat{\boldsymbol{\theta}}_{k-1} | \boldsymbol{\Psi}, \mathbf{X}_i)} \right] \quad (1-17)$$

It is not possible to know the true location, instead the current estimation after $k - 1$ items is used. The suggested rule is to select the item that maximizes the posterior expected Kullback-Leibler information defined as:

$$K_i^B(\hat{\boldsymbol{\theta}}^{k-1}) = \int_{\boldsymbol{\Theta}} K_i(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}^{k-1}) f(\boldsymbol{\theta} | X_{i1}, X_{i2}, \dots, X_{ik-1}, \boldsymbol{\Psi}) \partial \boldsymbol{\theta} \quad (1-18)$$

where $f(\boldsymbol{\theta} | \mathbf{X}_i, \boldsymbol{\Psi})$ is the posterior density given the $k - 1$ items. This method is computationally expensive because of the evaluation of the integrals.

2. Reinforcement Learning

2.1. Introduction

When talking about traditional computational paradigms, three variables define the procedure involved in the resolution of a problem: data, rules, and answers [Moroney, 2020]. The intention is to define the rules that, for a set of data, consistently calculate an answer, for example: rendering each pixel of an image, find the inverse of a matrix, develop an app that monitors heart beat, etc. With recent developments in computational power, machine learning algorithms became feasible and new computational paradigms involving data, rules, and answers, were introduced. The three most common machine learning task are: supervised learning, unsupervised learning, and reinforcement learning.

Supervised learning uses labeled data to discover the rules that maps data to answers. For every supervised problem, there is a “right” answer, and the trained model should be able to identify this answer for each data. Two problems are solved using this paradigm: classification and regression, in the former the right answer is categorical, whereas the later, the answer is continuous. Some examples of supervised algorithms are: Logistic Regression, support vector machines, decision trees, etc. On the other hand, unsupervised learning is used to discover patterns, clusters, and groups in unlabeled data; for example grouping customers according to their shopping list. This type of paradigm is useful to find insights and hidden relationships between variables.

Reinforcement learning is a sub-field of machine learning concerned to make decisions by interacting with some environment. In it, there is a ML (machine learning) model to be trained to take actions, it is called the *agent* and it learns to maximize some reward signal. Tasks to be resolved are categorized according to their nature in episodic and continuous. In episodic tasks there is a clear ending point, for example the resolution of a sudoku game, in contrast, continuous tasks can run indefinitely, for example, the process of controlling an elevator or the development of a personal assistance robot.

To understand the reinforcement learning paradigm, first we introduce an overview of Markov Decision Processes, the key components of a reinforcement learning problem, and the algorithm used in this document.

2.2. Markov Decision Process

Markov Decision Process (MDP) is the mathematical formalization of the process defining sequential decision making, it can be understood as the idealized form of the reinforcement learning problem [Sutton and Barto, 2018]. MDP proposes that any goal-directed learning task depends on three signals: (1) actions taken by the agent, (2) the state or basis of the action, and (3) the rewards received.

For a discrete time-steps $t = 1, 2, 3, \dots$, consider a representation of the environment's states $S_t \in \mathcal{S}$, an action $A_t \in \mathcal{A}$ and a reward $R_{t+1} \in \mathcal{R}$. MDP formulates that an agent perceives state S_t , chooses an A_t and as a consequence of the action it gets a reward R_{t+1} and new state S_{t+1} is reached. The sequence of these events is:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2 \dots \quad (2-1)$$

The pattern repeats until a terminal state is reached for episodic tasks; for continuous tasks, the pattern goes forever. If $\mathcal{S}, \mathcal{A}, \mathcal{R}$ have all finite values, the process is called *finite* Markov Decision Process, but in general, problems can have continuous spaces for both actions and states. For now, we assume R_t and S_t are finite random variables depending only on the preceding state and action to define the probability density function:

$$p(s', r|s, a) \doteq P(S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a) \quad (2-2)$$

The function p defines the environment model and defines the *dynamics* of the MDP. From p we can define the state-transition probabilities:

$$p(s'|s, a) \doteq \sum_{r \in \mathcal{R}} P(s', r|s, a) \quad (2-3)$$

And also calculate the expected reward given the preceding state and action:

$$\mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r|s, a) \quad (2-4)$$

In case p is known, we can use Dynamic Programming (DP), which is a collection of algorithms introduced by Richard Bellman to find the behavior that yields the best rewards by bootstrapping previous learned estimates, in practice, having all the assumptions for DP is rarely the case. Monte Carlo Methods (MCM) are useful when not assuming complete knowledge of the environment, they use approximations and the experience of previous interactions to find an optimal behavior. Finally, Temporal difference (TD) methods are a mix between DP and MCM, is a model-free approach like MCM, but uses other learned estimates like DP.

2.2.1. Policies and Rewards

The goal of reinforcement learning is to find the policy that maximizes cumulative reward over the long run. Given a time T to represent the terminal state, we can define the expected return following [Sutton and Barto, 2018] notation as:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \quad (2-5)$$

If task is continuous, then $T = \infty$ and G_t might diverge depending on the values of the rewards, to tackle this problem, the parameter *discount rate* (γ) where $0 < \gamma < 1$ is introduced to give more impact to immediate actions:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{t-T} R_T = \sum_{k=t+1}^T \gamma^{k-t-1} R_k \quad (2-6)$$

A policy π is the strategy that the agent uses to maps states to rewarding actions. It will select the corresponding action based on the current state and some knowledge on how each action will impact G_t . To understand which action is better, we need to define the *action-value* under the policy π :

$$v_\pi(s) \doteq \mathbb{E}_\pi [G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^T \gamma^k R_{t+k+1} | S_t = s \right] \quad (2-7)$$

And similarly, is the *action-value* function under policy π :

$$q_\pi(s) \doteq \mathbb{E}_\pi [G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^T \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \quad (2-8)$$

From then, we can say that a policy π is better than π' if $v_\pi(s) \geq v_{\pi'} \forall s \in \mathcal{S}$. The optimal policy by definition is the one that yields the largest expected return for all states. There can be multiple optimal policies, all of them are denoted as π_* . Optimal policies shares the same state-value function v_* and same action-value function q_* .

$$v_* = \max_{\pi} v_\pi(s, a) \quad (2-9)$$

$$q_* = \max_{\pi} q_\pi(s, a) \quad (2-10)$$

From (2-6) and (2-7) we can establish a recursive relationship between the current action-state function and the action-state function in the next state:

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi [G_t | S_t = s] \\
&= \mathbb{E}_\pi [R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) [r + \gamma \mathbb{E} [G_{t+1} | S_{t+1} = s']] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')] \tag{2-11}
\end{aligned}$$

And since $v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$, it is possible to rewrite (2-11) in terms of q :

$$q_\pi(s, a) = \sum_{s'} p(s', r | s, a) \left[r + \gamma \sum_{a'} \pi(a' | s') q_\pi(s', a') \right] \tag{2-12}$$

These are the Bellman equations for v and q respectively, and explicitly solving any of these provides the route to solve a reinforcement learning problem. However, depending on the problem, due to absence of p , or the highly computer resources, the exact solutions might not be feasible.

To give an example, consider the Chinese game GO that uses a board of 19x19 locations where each can be in three states: black piece, white piece, or empty; the number of possible states is 3^{361} if all combinations were legal. There is no computer at the moment that is able to process this amount of information, nor simulate every possible combination to find the optimal policy.

A common trade off in the reinforcement learning paradigm is exploitation vs exploration. The agent needs to decide whether to try a new action that has not been done, or to exploit the action that, by previous experiences produces the best return. Some methods like MCM and TD use a *ϵ -greedy policy*, where ϵ is the probability of selecting a new action instead of use a previous know action. To check whether the algorithm is approximating to a better policy, ϵ is reduced after each iteration using a *decay parameter* so the agent takes more decisions by itself and not just randomly tries new configurations.

2.3. Temporal Difference

The following example is borrowed from [Sutton and Barto, 2018]:

Suppose you wish to predict the weather for Saturday, and you have some model that predicts Saturday's weather, given the weather of each day in the week. In the standard case, you would wait until Saturday and then adjust all your

models. However, when it is, for example, Friday, you should have a pretty good idea of what the weather would be on Saturday – and thus be able to change, say, Saturday’s model before Saturday arrives.

Temporal Difference (TD) is a mixture of MCM and DP, in the sense that is a model-free method where the agent has no prior knowledge about the environment and it learns by bootstrapping the estimated. TD problems are divided in policy evaluation or *prediction*, which is the problem of estimate the action-value function given a policy, and the *control* problems that seeks to find the optimal policy.

TD predictions takes ideas from MCM to update the estimates V of v_π using a different approach. The target to update is $R_{t+1} + \gamma V(S_{t+1})$ and it gets done in an incremental way after each step:

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \quad (2-13)$$

This process is called TD(0) and was invented by Richard Sutton, α is the learning rate tuning parameter in the reinforcement learning context and it is usually between 0 and 1. Since the estimate of $V(S_t)$ uses the estimate of $V(S_{t+1})$, TD is a *bootstrapping* method.

Other benefits from TD against MCM are that TD is implemented in an incremental fashion while MCM has to wait for the episode to finish to update $V(S_t)$. TD also do not need the environment model and they have been proved to converge to v_π for any fixed policy π .

Control problems faces the exploration vs exploitation trade off. Almost any TD-control algorithm uses a greedy policy with a decay parameter to reduce the amount of exploration. The target is to directly approximate the learned action-value function Q to q_* . The main algorithms found in the literature are: SARSA, Expected SARSA, and Q-learning.

2.3.1. Q-Learning

Q-learning, proposed by [Watkins, 1989] is a TD method that incrementally updates the action-value function using the current reward and the expected action-value in the next state; the next action is influenced by the next state maximum future reward. The minimal requirement to guarantee the convergence to q^* is that all state-action pairs are visited, emphasizing the importance of the ϵ -greedy policy. For every state visited, Q is updated based the weighted average of the current value of the state-action pair and the max future action-value.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \quad (2-14)$$

Deep Q Learning is the process of implementing a Q-learning algorithm when a neural network plays the role of the action-value function. Any machine learning algorithm suitable for regression can be used instead of a neural network, however, neural networks have desirable properties like partial training, multi output generalization, fully customization of the architecture, and can handle non structured data like images.

If facing continuous state or continuous action spaces, [Wiering and van Otterlo, 2012] suggests some methods like: model approximation, policy approximation, and value approximation, the later uses a function approximation, e.g a neural network, to estimate the values of the action-value function. Function approximation helps solving problem with large problem with continuous state and action spaces generalizing earlier experiences to unseen states.

2.3.2. Stability and Experienced Replays

The neural network is just the model chosen, the performance depends on the quality of the data and the hyper-parameters set during the training phase. Since the agent is constantly learning, one of the hyper-parameters is the model training frequency.

There are no recipes when doing machine learning but some options are not feasible, let us say that we train the model after every action, this may be seem reasonable as the recent experience is automatically reflected in the next action, but, this could lead to some previous knowledge to be overwritten with new bad decisions.

Experience replay prevents the model to forget rewarding actions, reduces the variance of updates as the training data are randomly chosen, the model learns more efficiently as mistakes and hits are continuously updated, and stables the model as by training with a larger dataset.

It is worth to notice that training the neural network after every action, even using experience replay, leads to instability due to the fact that updating $Q(S_t, A_t)$ also updates $Q(S_{t+1}, a)$, thus increasing the objective variable. [Mnih et al., 2015] resolved this issue introducing a second network called “target network”, the notion is to grab the experience replay, use the target network to estimate the y_i following equation (2-14), and then train the original network after some episodes. The weights of the target network are periodically updated to match the weights of the original network.

2.3.3. Illegal Actions

There are two types of actions: legal and illegal, while the former can include actions that increases or decrease cumulative reward, the later simply cannot be executed, for example, consider a tic-tac-toe game where the agent goes second, if the opponent takes the center

position, the agent simply cannot overwrite this location.

Illegal actions should be carefully managed, depending on the range of the action-value function used. One idea is just to ignore them and pick the legal action that yields the highest reward, in general this can work but if there is no prior information provided by the environment about the legal action, then the model by experience troubles to learn how to map actions. Another option is to penalize illegal actions and propagate the bad reward until the episode is done; depending on the problem, this approach takes longer to converge as all state-illegal actions should be learned by the algorithm. Last, is to ignore and adjust the action-value result of that action no lower number, that way, the algorithm not only learns how to map illegal actions to low Q , none of this actions are taken by the agent.

Independently of the approach to solve illegal actions, the observation should contain the relevant information regarding illegal actions.

2.3.4. Deep Q-Learning Algorithm

DQL considers multiple hyper-parameters like learning rate, batch size of training set, network architecture and number of hidden units, replay buffer maximum capacity, among others. The following algorithm describes the step by step procedure to implement a DQL algorithm for an episodic task

Algorithm 1 Deep-Q-Learning with Experience Replay

- 1: Initialize main network and target network with random weights
 - 2: Initialize Replay Buffer RB with batch size BN and capacity N
 - 3: **for** episode = 1 ... M **do**
 - 4: Reset environment and initialize $s = s_0$
 - 5: **while** s is not terminal **do**
 - 6: Choose action a using main network ϵ -greedy policy
 - 7: Take action a and observe r', s'
 - 8: Store trajectory (s, a, r, s) in RB
 - 9: Sample batch of size BN
 - 10: Train target network
 - 11: Every C steps update, copy weights for target network to main network
-

2.4. Policy Gradient Methods

Another family of methods to train for a reinforcement learning task, updates the policy directly instead of using the value-function to take actions, these methods seek to maximize

a performance metric related to the reward. These types of methods are considered when expressing a policy function is easier than approximating an action-state function for all possible combinations, for example, consider the Atari Breakout game where establishing a policy of not letting the ball fall is conceptually easier than estimating a value function for every position of the ball and brick.

We start by defining the policy $\pi(a|s, \boldsymbol{\rho}) = \Pr\{A_t = a|S_t = s, \boldsymbol{\rho}_t = \boldsymbol{\rho}\}$ as a parametrized function of $\boldsymbol{\rho}$ that yields the probability that action a is taken at time t given that the environment is at state s ; some learning rate parameter α , and the performance metric $J(\boldsymbol{\rho})$ we seek to maximize using stochastic gradient ascent:

$$\boldsymbol{\rho}_{t+1} = \boldsymbol{\rho}_t + \alpha \widehat{\nabla_{\boldsymbol{\rho}} J(\boldsymbol{\rho})} \quad (2-15)$$

where $\widehat{\nabla_{\boldsymbol{\rho}} J(\boldsymbol{\rho})} \in \mathbb{R}^d$ is a stochastic estimate whose expectation approximates the true gradient of the performance metric with respect to the policy parameters.

2.4.1. Policy Gradient Theorem

The most natural metric to improve, is the expected discounted cumulative reward of initial state. The problem reduces to establish a procedure to update the policy $\pi(a|s, \boldsymbol{\rho})$ maximizing the metric, by establish a mathematical expression to get the gradient of the value function with respect to $\boldsymbol{\rho}$.

$$\begin{aligned} \nabla J(\boldsymbol{\rho}) &= \nabla v_{\pi}(s_0) \\ &= \nabla \mathbb{E}[G_t | S_t = s_0] \\ &= \nabla \left[\sum_a \pi(a|s) q_{\pi}(s, a) \right] \end{aligned} \quad (2-16)$$

The policy gradient theorem derives this gradient into an expression we can use to sample and implement stochastic gradient ascent (See appendix for a formal proof):

$$\begin{aligned} \nabla J(\boldsymbol{\rho}) &\propto \sum_s \mu(s) \sum_a q_{\pi}(s, a) \nabla \pi(a|s, \boldsymbol{\rho}) \\ &= \sum_s \mu(s) \sum_a \pi(a|s) q_{\pi}(s, a) \frac{\nabla \pi(a|s, \boldsymbol{\rho})}{\pi(a|s, \boldsymbol{\rho})} \\ &= \mathbb{E}_{\pi} [q_{\pi}(S_t, A_t) \nabla \ln \pi(A_t | S_t, \boldsymbol{\rho})] \end{aligned} \quad (2-17)$$

$$= \mathbb{E}_{\pi} [G_t \nabla \ln \pi(A_t | S_t, \boldsymbol{\rho})] \quad (2-18)$$

2.4.2. REINFORCE

Also known as Monte-Carlo policy gradient methods, it is an online method that updates the policy parameters by episode using stochastic gradient ascent algorithm, this requires samples whose expectation is proportional to the actual gradient of the performance metric. These samples are easily obtained from the policy gradient method. The algorithm is defined as:

$$\boldsymbol{\rho}_{t+1} = \boldsymbol{\rho}_t + \alpha G_t \nabla \ln (A_t | S_t, \boldsymbol{\rho}_t) \quad (2-19)$$

REINFORCE is widely used with a baseline to normalize differences in size, the notion behind its operation is to try to maintain the same scale whether the gradient is high in comparison to the value function. The baseline is any function as long as it does not depend on the action, otherwise the gradient is affected by this quantity. A natural election is the value function.

$$\boldsymbol{\rho}_{t+1} = \boldsymbol{\rho}_t + \alpha (G_t - \hat{v}_\pi(S_t, \boldsymbol{w})) \nabla \ln (A_t | S_t, \boldsymbol{\rho}_t) \quad (2-20)$$

REINFORCE requires entire episodes as it uses the return at every time step. See Algorithm 2 for the pseudo code of the reinforce with baseline algorithm.

Algorithm 2 REINFORCE with Baseline Algorithm

- 1: Initialize a differentiable policy $\pi(a|s, \boldsymbol{\rho})$
 - 2: Initialize a differentiable value function $\hat{v}(s, \boldsymbol{w})$
 - 3: **for** episode = 1 ... M **do**
 - 4: Reset environment and initialize $s = s_0$
 - 5: collect $S_0, A_0, R_1, S_1, A_1, \dots, S_{T-1}, A_{T-1}, R_T$
 - 6: **for** $t = 0, 1, \dots, T$ **do**
 - 7: $G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k$
 - 8: $A \leftarrow G - \hat{v}(S_t, \boldsymbol{w})$
 - 9: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^w A \nabla \hat{v}(S_t, \boldsymbol{w})$
 - 10: $\boldsymbol{\rho} \leftarrow \boldsymbol{\rho} + \alpha^\rho A \nabla \ln \pi(A_t | S_t, \boldsymbol{\rho})$
-

2.4.3. Actor-Critic

Another family of methods uses the ideas of Q-learning to implement an approximation of the value-function that criticizes the decision taken by the REINFORCE algorithm, in other words, whether the policy indicates that some state has a high probability of maximizing cumulative reward, this decision is contrasted with the estimated action-state function that

leads to the new state.

Actor-critic and REINFORCE methods are considered online policy optimization as the policy parameter are updated using only the most recent episodes without the need of implementing or having an buffer for experienced replays.

2.4.4. TRPO: Trust Region Policy Optimization

This on-line procedure to optimize policies was proposed by [Schulman et al., 2015] and used the previous work of natural gradient policy to establish a surrogate loss function with little tuning of hyper-parameters that show monotonic improvement.

Prior work from [Kakade and Langford, 2002] proposed a policy update scheme for mixture policies called conservative policy iteration (CPI) using a local approximation of the expected discounted reward $L_\pi(\tilde{\pi})$ for a policy $\tilde{\pi}$, in terms of the policy $\tilde{\pi}(a|s)$, the current policy $\pi(a|s)$ and the advantage function of the current policy $A_\pi(s, a)$. Later, they derived a lower bound of the expected discounted reward in terms of the local approximation.

TRPO extends the results of CPI to general policies, and, by making some adjustments to the lower bound, derived a constrained optimization:

$$\max_{\boldsymbol{\rho}} \sum_s \beta_{\boldsymbol{\rho}_{\text{old}}}(s) \sum_a \pi_{\boldsymbol{\rho}}(a|s) A_{\boldsymbol{\rho}_{\text{old}}}(s, a) \quad (2-21)$$

$$\text{Subject to } \overline{D}_{KL}^{\boldsymbol{\rho}_{\text{old}}}(\boldsymbol{\rho}_{\text{old}}, \boldsymbol{\rho}) \leq \delta$$

By expressing (2-21) in terms of expected values and the importance sampling estimator, one might find an equivalent expression to maximize:

$$\max_{\boldsymbol{\rho}} \mathbb{E}_{s \sim \beta_{\boldsymbol{\rho}_{\text{old}}}, a \sim \boldsymbol{\rho}_{\text{old}}} \left[\frac{\pi_{\boldsymbol{\rho}}(a|s)}{\pi_{\boldsymbol{\rho}_{\text{old}}}(a|s)} A_{\boldsymbol{\rho}_{\text{old}}}(s, a) \right] \quad (2-22)$$

$$\text{subject to } \mathbb{E}_{s \sim \boldsymbol{\rho}_{\text{old}}} [D_{KL}(\pi_{\boldsymbol{\rho}_{\text{old}}}(\cdot|s) \parallel \pi_{\boldsymbol{\rho}}(\cdot|s))] \leq \delta$$

2.4.5. PPO: Proximal Policy Optimization

PPO is another actor-critic whose idea is similar TRPO in the sense of increasing the step to reduce convergence time but not so large so they are stable after each training. PPO uses the same surrogate objective function but modifies the constraint imposed in the KL divergence to a much simpler criteria that does not involve the second order derivative. There are two versions of PPO one that imposes a penalty on the KL divergence, and one that clips the probability ratio between the old policy and the new policy. Given the popular use of the clipping strategy, the following subsection will give some context and how to implement it.

Given the surrogate objective function for the conservative policy iteration:

$$L^{\text{CPI}}(\boldsymbol{\rho}) = \mathbb{E}_{\pi_{\boldsymbol{\rho}_{\text{old}}}} \left[\frac{\pi_{\boldsymbol{\rho}}(a|s)}{\pi_{\boldsymbol{\rho}_{\text{old}}}(a|s)} A_{\boldsymbol{\rho}_{\text{old}}}(s, a) \right] \quad (2-23)$$

PPO proposes to clip whenever the probability ratio $r_t(\boldsymbol{\rho}) = \frac{\pi_{\boldsymbol{\rho}}(a_t|s_t)}{\pi_{\boldsymbol{\rho}_{\text{old}}}(a_t|s_t)}$ is away from 1:

$$L^{\text{CLIP}}(\boldsymbol{\rho}) = \mathbb{E}_{\pi_{\boldsymbol{\rho}_{\text{old}}}} [\min(r_t(\boldsymbol{\rho})A_t(s, a), \text{clip}(r_t(\boldsymbol{\rho}), 1 - \epsilon, 1 + \epsilon)A_t(s, a))] \quad (2-24)$$

Using this scheme, L^{CLIP} will take the minimum value between the original L^{CPI} and the clipped version when the probability ration is not in $[1 - \epsilon, 1 + \epsilon]$, therefore changes the new policy will not largely differ than the current policy.

Algorithm 3 PPO Clip Algorithm

- 1: Initialize current policy with parameters $\boldsymbol{\rho}_k$
 - 2: **for** episode = 1 . . . M **do**
 - 3: Collect $N = \{\tau_i\}$ trajectories using policy $\pi_{\boldsymbol{\rho}_k}$
 - 4: Compute all return G_t
 - 5: Compute the advantage function A_t using an estimator of the value function (e.g. a neural network)
 - 6: Update the policy parameter by maximizing the objective function $\boldsymbol{\rho}_{k+1} = \arg \max_{\boldsymbol{\rho}} L^{\text{CLIP}}$
 - 7: Update value function parameters minimizing mean squared error
-

3. Simulation

Given the item selection problem in a CAT procedure we can train a model following the rules of reinforcement learning paradigm. The model in this context, is an agent that selects the next question given the observation. The following section will focus on the simulation process for the CAT environment, the hyper-parameters of the RL training, and the evaluation of the model against other proposed methodologies.

As seen before, RL demands a precise interpretation of episodes, state, rewards and actions. Linking ideas from IRT, we conclude that an episode is the process of taking a test, the state is a collection of variables to describe the current estimation, the standard error of the latent traits and the items provided; and the actions refer to the choice of non-asked remaining items. Reward definition highly depends on the purpose of the algorithm; we begin by training an agent that quickly reduces the number of questions by reducing the standard error of the latent traits, and later propose a different methodology that incorporates the exactness of the estimation.

3.1. Simulation

Suppose a number $N = 1000$ of examinees whose latent traits θ are located in a 3-dimensional and follow a multivariate normal distribution: $\theta \sim MN(\mu_\theta, \Sigma_\theta)$ with means $\mu_\theta = \mathbf{0}$ and covariance matrix Σ_θ . Assuming no correlation between latent traits, then:

$$\theta_{ij} \sim N(0, 1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \quad \forall i \in \{0, 1, 2\} \quad (3-1)$$

Considering a multivariate normal is a decision based on the traditional uses of this distribution to handle continuous data for population. Another argument is the result [Ioffe and Szegedy, 2015] proposed, where having standardized data normalized stabilize and speed up the training of neural networks.

The item bank is composed of $M = 210$ items. The items slopes α_j are randomly chosen following a multivariate normal distribution using the following method, and the intercept parameter d is a sample from a univariate uniform. The steps are:

- There are $P = 3$ sections, each consists of M/P items.

- The items of each section discriminates one ability, specifically, the items of the first section \mathcal{S}_1 discriminates θ_1 , \mathcal{S}_2 discriminates θ_2 , and \mathcal{S}_3 discriminates θ_3 .
- Each $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$, follow a log multivariate normal distribution with $\boldsymbol{\mu}_{\mathcal{S}_1} = (\mu_{1\mathcal{S}_1}, \mu_{2\mathcal{S}_1}, \mu_{3\mathcal{S}_1})$, $\boldsymbol{\mu}_{\mathcal{S}_2}, \boldsymbol{\mu}_{\mathcal{S}_3}$ and $\Sigma_{\mathcal{S}_1}, \Sigma_{\mathcal{S}_2}, \Sigma_{\mathcal{S}_3}$ respectively. Log-multivariate normal benefits of having a density function always positive, otherwise if using the 3PL, lacking an ability might increase the chances of answer correctly. For \mathcal{S}_1 , the mean $\mu_{1\mathcal{S}_1}$ should be greater than $\mu_{2\mathcal{S}_1}$ and $\mu_{3\mathcal{S}_1}$, to reiterate that section 1 puts more importance on θ_1 .

To check the performance of the model the test set is constructed with the same assumptions and consists of $U = 200$ examinees that are not part of the initial $N = 1000$ people. The training phase is not aware of the existence of these people. This test group will be used to compare the results of the models against other classical proposed method.

Refer to algorithm 4 for more information about the items bank generation process.

Algorithm 4 Items Bank Generation

- 1: **procedure** GENERATE_ITEMS
 - 2: $M \leftarrow 210$
 - 3: $P \leftarrow 3$
 - 4: **for** i **do** $1 \dots P$
 - 5: Initialize $\boldsymbol{\mu}_{\mathcal{S}_i}$ using a random uniform distribution
 - 6: Initialize $\Sigma_{\mathcal{S}_i}$
 - 7: Generate slopes $\alpha_1, \alpha_2, \dots, \alpha_p$ for M/P items by sampling from log-multivariate normal distribution
 - 8: Generate intersects \boldsymbol{d} for M/P items by sampling from an uniform distribution
-

3.2. IRT components

The process of collecting answers for all items, for each person, is done before starting the training procedure, due to the fact that environments should be as reproducible as possible. The exploration might not be possible if the answer for one person differs every time the test is taken.

The method chosen for estimation is the maximum a posteriori estimation (MAE). Considering this methods relies on knowing how the simulation was performed, to keep the estimation as exactly as possible, and to accelerate the convergence.

Termination criteria were defined empirically by setting a max number of questions to 60, and early termination when the standard error of the estimation for each latent trait es below 0,35.

3.3. Reinforcement Learning Hyper-parameters

Whether we use DQN, REINFORCE or PPO strategies, the environment, actions, and rewards must be equal for all three algorithms. As described before, the environment is a test/exam taken by a random person, and the observations are composed of:

- Current estimation of latent traits (initialized to be the mean).
- Current standard error of estimations (initialized to be a vector of ones).
- Dichotomous variables for each question indicating if the question was already asked.

The action is the choice of an integer number between 1 and 210 that selects the corresponding question from the item bank. Actions can be illegal, i.e. the model can select a question that was already asked for the same test, in this case and independently of the algorithm used, illegal actions are ignored, i.e the action is omitted and the next legal action with the highest prediction is selected.

For the reward signal, two schemes are implemented:

1. To give a constant reward of -1 for each question asked until a terminal state is reached, and if the convergence criteria of the standard errors are met, a positive reward of +30 is given at the terminal state.
2. The second strategy gives reward equal to the negative Euclidean distance between the current estimation and the true abilities of the examinee, and a positive reward of +30 if the convergence criteria are met.

The first option forces the algorithm to find the optimal route to reach convergence as soon as possible, and the second allows the agent to learn a route that minimizes the estimation error and minimize the number of questions asked.

Multiple discount values are used, from experience and nature of the reward signal a large discount factor should be used to give more importance to future rewards.

3.4. Models

Reinforcement learning benefits from a model that: allows partial training with randomly generated samples from the experience replays or recent episode data for the REINFORCE case; can generate multiple outputs like action-values for each action or action distributions; and finally, can predict a continuous value. The most natural selection to address this problem are neural networks.

The model architecture is the same for the DQN, REINFORCE and PPO, with the little notation that DQN output is the estimate of the action-value function, REINFORCE uses one model to estimate the policy and another model to estimate the action-value function to be used as a baseline, and PPO uses the actor-critic architecture with two models. Figure 3-1 shows a high level diagram of the model implemented. All models use Adam optimizer, constant learning rate, RMSE loss function and constant batch size.

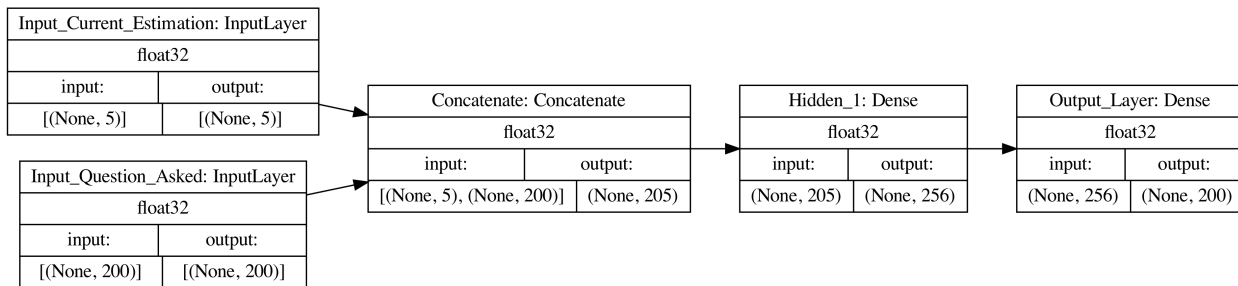


Figure 3-1.: Neural Network Model Used For Q-learning

Each algorithm carries its own hyper-parameters, like replay buffer capacity, the number of steps to collect before training the network, checkpoint intervals, etc. Algorithms were trained using tensorflow-agent library [Guadarrama et al., 2018] and using Weight & Biases [Biewald, 2020] machine learning platform for experiment tracking. Information and graphics about each model can be found in <https://wandb.ai/jcaliz/TDG> and every code used is stored on the personal github <https://github.com/JoseCaliz/TDG>

3.5. Results

Unless explicitly specified these results are measures of the performance of Reinforcement Learning (RL) algorithm and traditional statistical strategies on the test set. For the following part, only the two best models for each algorithm were selected. Dashed line is used for the reward signal that depends on the parameter estimation. Please refer to C-1 for the respective hyper-parameters. The number after the model name, refers to the rewards strategy used to train the model.

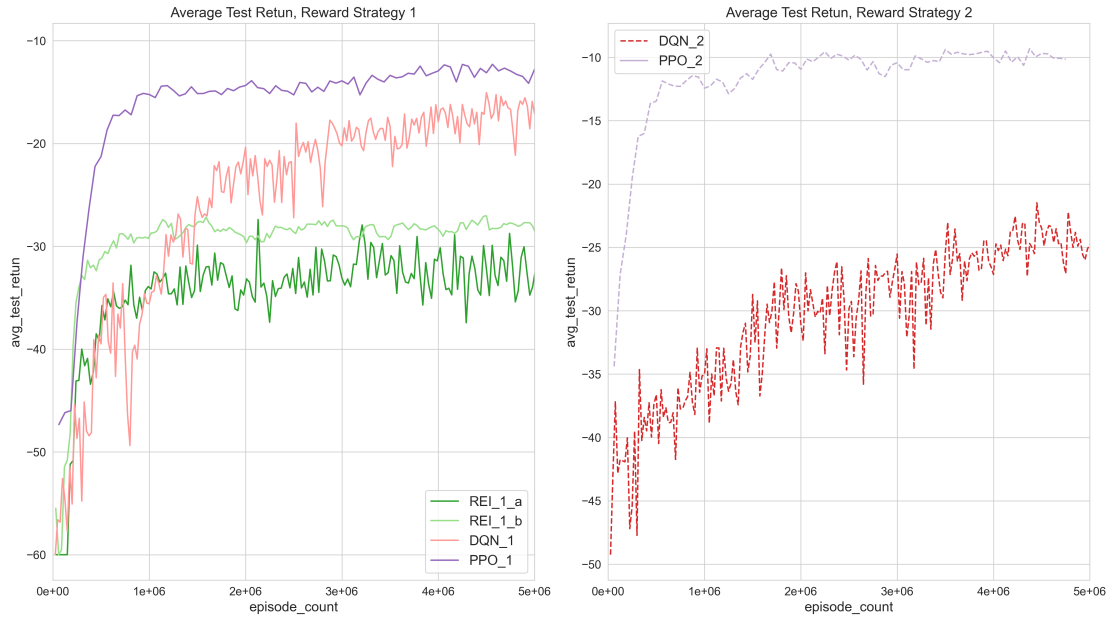


Figure 3-2.: Average Test Return During Training process

It would not make sense to use an algorithm that did not converge, and it is an interesting comparison to check graphically which models converged in fewer steps. Models trained with the second reward scheme are plotted in a different figure as returns for these models have different meanings. Figure 3-2 shows the average test return after supplying many of episodes, it can be seen that policy gradient models tend to become deterministic after 1 million episodes, while DQN methods have higher variance and slower convergence, which is explained by the exploration parameter being set to a constant. Some variations of DQN incorporate a *decay* parameter to reduce the exploration and exploit knowledge for later episodes; however, it was not considered part of this implementation. Both PPO algorithms converged to best policy given the respective reward signal provided, in contrast to REINFORCE methods that stuck in local optima, a sufficient reason to not consider this algorithm for further experimentation in the second approach of rewards.

Another analysis is to check the performance of strategies using the convergence criteria, for which, figure 3-3 show the kernel density estimation of the number of questions asked per strategy. At first glance both PPO models and the first DQN model are the ones that converge faster thus reducing the number of questions, a direct consequence of finding a better policy. REINFORCE models are less aggressive in reducing the standard error as fast as possible and both have similar behavior for this metric, with the *wordly-pond-8* model having a slight better performance. As for the traditional statistical methods, Drule and Arule have the best performance with distributions spread across the possible values, while KL and Trule distributions are skewed to the right.

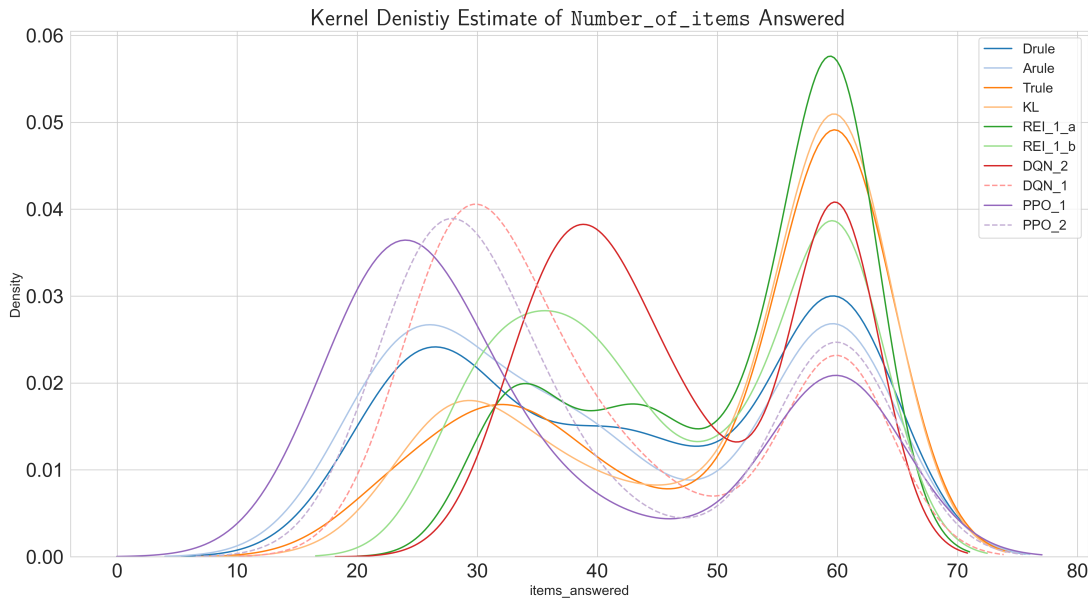


Figure 3-3.: Kernel Density Estimator of `number_of_items` Asked Per Model For Test Set

Reducing the number of items does not imply that estimation is improved. While CAT provides a framework to compensate for estimation vs number of items, it mostly depends on the items provided. We can simulate the possible results of each item selection strategy for a given person and the answers for the 200 items in the bank, allowing us to understand in which cases the methods have a better compromise between the number of items and prediction estimation, under the assumption that applying more items increases the accuracy of prediction. This process is repeated for all test users resulting in 10 estimations per person (one for each statistical method and one for each RL models). We begin by comparing the exactness of the estimation based on the items provided using the box plots of the Euclidean distance between the estimation of the method and the true value of the person.

Figure 3-4 shows that, apart from PPO trained with the second reward scheme, none of the RL models have competitive estimations against traditional statistical methods, highlighting the importance of the definition of the reward signal, which also explains why the DQN_2 reduces the estimation error compared to other models. Even though models PPO_daily-field-68 and DQN_1 show outstanding performance in reducing the number of items and reach convergence criteria the cost is that the estimate is inaccurate.

Strategies can be compared per person, i.e., for a given person, which strategy used the least number of items and the most accurate estimation, a full table of the 200 people is given at the appendix. Two additionally summary tables are provided: 3-1 and 3-2. The first table

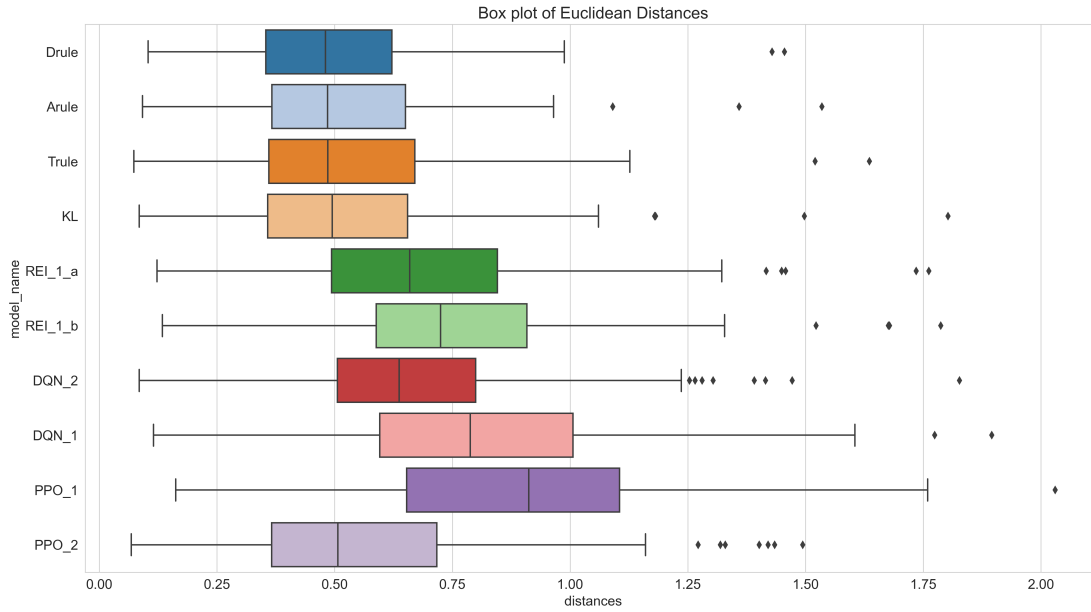


Figure 3-4.: Kernel Density Estimator of `number_of_items` Per Model For Test Set

summarizes the cases where the respective strategy had the lowest result in distance, and the second, the lowest items suministered. For the distance case, while it seems PPO_2 outperforms other models in the free-for-all contest, this is not the case when doing one-on-one contest where the 200 results are divided into just two strategies, as shown in figure 3-5 where in the first two subplots, Arule and Drule find better estimations for most users, and in the latter subplot it can be seen that when comparing the three strategies PPO is the one that yields the higher count.

The reciprocate analysis of the total number of items is done by 3-2 and but this table is already filtered to include only the RL models that show a good compromise between estimation and convergence. Results are not surprising and they confirm insights we got from figure 3-3. Some rows have more than one strategy as ties are possible when the strategies used the same number of items for the same person.

So far, both variables `number_of_items` and `distance` have been discussed separately and independent, for a more complete analysis we should give an importance metric to each variable and check which strategies perform better. A natural approach is to plot the bivariate histogram that is shown in figure 3-6, where traditional methods, in particular Drule and Arule, tend to keep a balance between both variables and few outliers on the distance, whereas RL methods show cases where estimation is inaccurate and the number of items asked is around the center of each graph. Some useful insight is that the distribution Drule, Arule, KL, and PPO are denser when the number of questions is either low or high, and the

Strategy	Cases (Distance)
Drule	24
Arule	39
Trule	22
KL	23
REI_1.a	16
REI_1.b	8
DQN_1	3
DQN_2	10
PPO_1	1
PPO_2	54

Table 3-1.: Count Of Cases When Strategy Got The Lowest Euclidean Distance

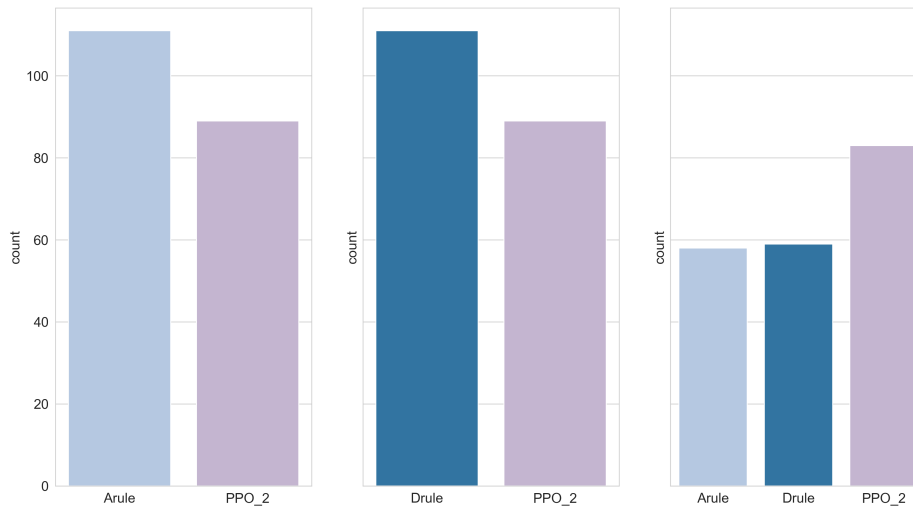


Figure 3-5.: Count Of Best Distance Comparing Multiple Strategies

	Cases (Items)
PPO_2	71
Arule, DQN_2, Drule, KL, PPO_2, Trule	58
Arule	39
Arule, Drule	7
Drule	6
DQN_2	4
Arule, Drule, Trule	3
Arule, PPO_2	3
Drule, PPO_2	2
KL	2
Trule	2
Arule, Drule, KL, Trule	1
Arule, KL, PPO_2	1
Arule, Drule, PPO_2	1

Table 3-2.: Count Of Cases When Strategy Got The Lowest Number Of Items Asked

distance is low.

Generally, there is no clear nor correct way to do to perform a multi-output evaluation. The methodology used here is based on scale variables to remove units, give importance to each and use the concept of loss function in the machine learning paradigm. The steps are:

- Select the most promising strategies.
- Scale `number_of_items` and `distance` variables using a min-max scaler.
- Create a new variable `CAT_score` as the sum of the scaled variables.
- Group by strategy and aggregate the sum of `CAT_score`.

This methodology allows us to weight each variable by factors, say for example we want strategy that penalizes large distances, it is sufficient to use the updated formula `score = distances × 2 + items`. The variable `score` and table **3-3** provides information about the best model using different scores definition. PPO model has a good compromise when both variables have the same importance but Arule strategy is better when giving more importance to any variables.

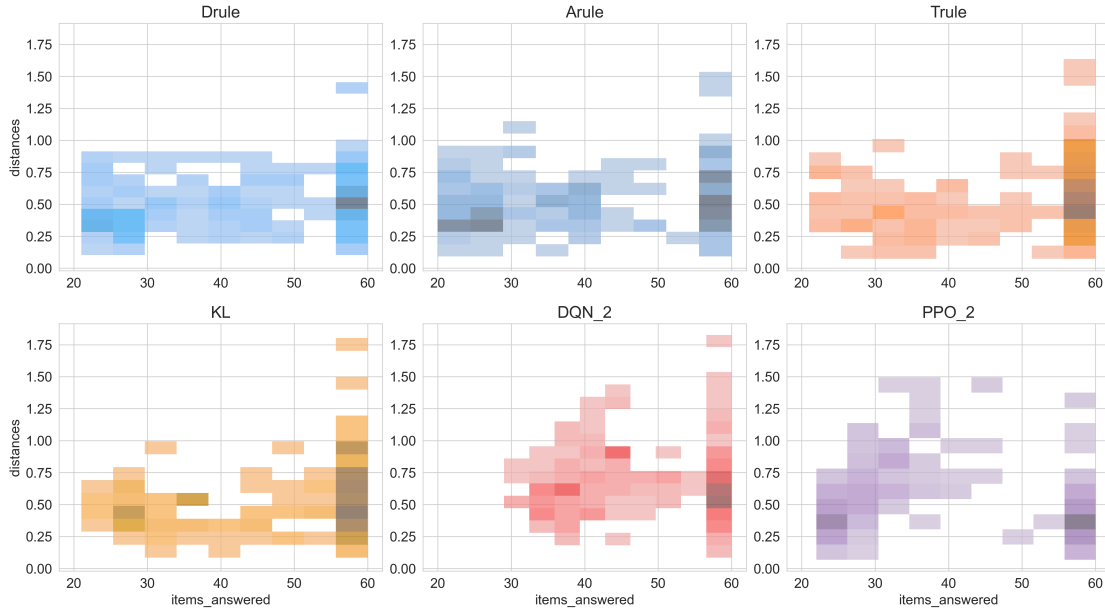


Figure 3-6.: Bi-variate Histogram Of Models With Most Accurate Estimation

Score Formula	Lowest Score	Second Lowest Score
$\text{distance} \times 0 + \text{items_answered}$	PPO_2	Arule
$\text{distance} + \text{items_answered} \times 0$	PPO_2	Arule
$\text{distance} + \text{items_answered}$	PPO_2	Arule
$\text{distance} \times 2 + \text{items_answered}$	Arule	PPO_2
$\text{distance} + \text{items_answered} \times 2$	Arule	PPO_2
$\text{distance} \times 3 + \text{items_answered}$	Arule	Drule
$\text{distance} + \text{items_answered} \times 3$	Arule	Drule

Table 3-3.: Best Models According To The Sum Of score

4. Conclusions

4.1. Conclusions

We have explored different attempts to solve the problem of selecting the next items one at a time in a Computerized Adaptive Test (CAT) matched to examinee ability, keeping the number of questions as low as possible and improving the accuracy of the estimation. While it seems that most of the statistical methods have similar results, RL paradigms allowed us to find an optimum model to fit specific requirements as the reward hypothesis is a powerful framework for training custom algorithms. It is not trivial to find an optimum model, the assumptions we have used here are the typical IRT assumptions and the models we have found not necessarily will work on other cases. There is no strategy that can solve this problem from all the considered perspectives for comparison.

For the case presented and under the metric proposed, the PPO model (PPO_2) has the best performance when giving the same importance to the number of questions asked and the accuracy of the estimation, but the results changed when giving importance to any metrics. Arule and Drule show great performance in the latter case, due to having fewer outliers on the metrics as shown in bi-variate histograms.

The reward signal defines the performance of the RL algorithm. We cannot compare internally two algorithms that were trained under different reward scheme; however, in general, REINFORCE methods got stuck in local optima, and for this particular task performance is below in comparison to PPO and DQN. On the other hand, DQN had slow convergence, implying that additional hyper-parameter tweak is necessary.

4.2. Recommendations

Multiple possible improvements can be developed for this project. We did not explore the case where questions have a correlation between them, which is probably the most common case for tests in the real world. The procedure would be to re generate another bank of items, retrain the RL models and check the results. Following the same path, another extension is to consider a different distribution for the latent traits as using a multivariate

normal distribution is a strong assumption that affects the estimation procedure.

Double Deep Q Network is another popular machine learning model that was not part of this analysis, it seeks to reduce the known overestimation of the DQN by decomposing the max operation in the target into action selection and action evaluation using the target network to estimate the values [van Hasselt et al., 2015]. The performance of this model is superior compared to DQN for the Atari games. The penalty PPO strategy was not tested for this work either.

Considering a larger sample and scalability when latent trait dimensions are greater than three. Traditional statistic strategies do not involve a training procedure at that fact is a potential downer when using RL models. For this case, timing profiles and resource management should be taken into account.

Finally, to set up a decay parameter for DQN and increase batch size to explore whether the model converges faster with respect to the number of iterations to the optimal action-value function, as DQN is proven to converge when all actions-states pairs are visited with a non-zero probability.

A. Proof of Bellman Equation

The following is a formal and in-depth proof of the Bellman equation (2-11). We begin by definition of the value-state function:

$$\begin{aligned}
 v_\pi &= \mathbb{E}[G_t | S_t = s] \\
 &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
 &= \mathbb{E}[R_{t+1} | S_t = s] + \mathbb{E}[G_{t+1} | S_t = s]
 \end{aligned} \tag{A-1}$$

The first expected is expressed value using (2-2) and the definition of the policy $\pi(a|s) = p(a|s)$

$$\begin{aligned}
 \mathbb{E}[R_{t+1} | S_t = s] &= \sum_{s'} \sum_r r p(r, s' | s, a) \\
 &= \sum_a \sum_{s'} \sum_r r p(r, s' | s) \pi(a|s)
 \end{aligned} \tag{A-2}$$

As for the second expected value, using conditional probabilities relationships:

$$\begin{aligned}
 \mathbb{E}[G_{t+1} | S_t = s] &= \sum_{g'} g' p(g' | s) \\
 &= \sum_{g'} g' \sum_r \sum_{s'} p(g', s', r | s) \\
 &= \sum_{g'} g' \sum_r \sum_{s'} p(g' | s', r, s) p(s', r | s)
 \end{aligned} \tag{A-3}$$

$$= \sum_{g'} g' \sum_r \sum_{s'} p(g' | s') p(s', r | s) \tag{A-4}$$

$$= \sum_r \sum_{s'} \sum_{g'} g' p(g' | s') p(s', r | s)$$

$$= \sum_r \sum_{s'} v_\pi(s') p(s', r | s)$$

$$= \sum_a \sum_r \sum_{s'} v_\pi(s') p(s', r | s, a) \pi(a|s) \tag{A-5}$$

The Markov property, an assumption of the RL framework, states that the reward of the current state depends only on the current state, thus $p(g'|s', r, s, a) = p(g', |s')$ which explains the transition from (A-3) to (A-4). Putting together (A-1), (A-2), (A-5):

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}[R_{t+1}|S_t = s] + \mathbb{E}[G_{t+1}|S_t = s] \\
&= \sum_a \sum_{s'} \sum_r r p(r, s'|s) \pi(a|s) + \sum_a \sum_r \sum_{s'} v_\pi(s') p(s', r|s, a) \pi(a|s) \\
&= \sum_a \pi(a|s) \sum_{r, s'} p(r, s'|s) (r + \gamma v_\pi(s'))
\end{aligned} \tag{A-6}$$

□

B. Proof of the Policy Gradient Theorem

We begin by establishing a recursive relationship between the action-state function of the current state and the value function of the next state, assuming no discount ($\gamma = 1$)

$$\begin{aligned}
 q_\pi(s, a) &= \mathbb{E}[G_t | S_t = s, A_t = a] \\
 &= \mathbb{E}[R_{t+1} + G_{t+1} | S_t = s, A_t = a] \\
 &= \mathbb{E}[R_{t+1} | S_t = s, A_t = a] + \mathbb{E}[G_{t+1} | S_t = s, A_t = a] \\
 &= \sum_{r, s'} r p(s', r | s, a) + \sum_{r, s'} v_\pi(s') p(s', r | s, a) \quad \text{Using results from (A-2) and (A-5)} \\
 &= \sum_{r, s'} p(s', r | s, a) (r + v_\pi(s')) \tag{B-1}
 \end{aligned}$$

The policy gradient theorem expresses the gradient of the performance metric with respect to the policy parameters. To keep notation simple, all gradients are respect to $\boldsymbol{\rho}$ and the policy π is a function of $\boldsymbol{\rho}$:

$$\begin{aligned}
 \nabla v_\pi(s) &= \nabla \left[\sum_a \pi(a|s) q_\pi(s, a) \right] \\
 &= \sum_a [\nabla \pi(a|s) q_\pi(s, a) + \pi(a|s) \nabla q_\pi(s, a)] \quad \text{Product rule} \\
 &= \sum_a \left[\nabla \pi(a|s) q_\pi(s, a) + \pi(a|s) \nabla \sum_{s', r} p(s', r | s, a) (r + v_\pi(s')) \right] \\
 &\quad \text{Using result from (B-1)} \\
 &= \sum_a \left[\nabla \pi(a|s) q_\pi(s, a) + \pi(a|s) \sum_{s', r} p(s', r | s, a) (\nabla r + \nabla v_\pi(s')) \right] \\
 &= \sum_a \left[\nabla \pi(a|s) q_\pi(s, a) + \pi(a|s) \sum_{s'} p(s' | s, a) \nabla v_\pi(s') \right] \tag{B-2}
 \end{aligned}$$

This last equation has a recursive form where $v_\pi(s')$ is unrolled in terms of a', s', s'' , and $v_\pi(s'')$ following the same procedure, and so on, for example:

$$\begin{aligned} \nabla v_\pi(s) = \sum_a \left[\nabla \pi(a|s) q_\pi(s, a) + \pi(a|s) \sum_{s'} p(s'|s, a) \right. \\ \left. \nabla \sum_{a'} \left[\nabla \pi(a'|s') q_\pi(s', a') + \pi(a'|s') \sum_{s''} p(s''|s', a') v_\pi(s'') \right] \right] \end{aligned} \quad (\text{B-3})$$

Consider the transition from state s to state s' : $\sum_a \pi(a|s) p(s'|s, a) = \omega(s \rightarrow s', k=1)$ where k is the number of actions taken. If the goal is to go from s to x , in $k+1$ steps, the agent can transition from s to any intermediate step s' in k steps and then from s' to x in one step: $\sum_{s'} \omega(s \rightarrow s', k) \omega(s' \rightarrow x, 1)$.

To keep notation simple consider $\phi(s) = \sum_a \nabla \pi(a|s) q_\pi(s, a)$. Expressing (B-2) in terms of $\phi(s)$:

$$\begin{aligned} \nabla v_\pi(s) &= \phi(s) + \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) v_\pi(s') \\ &= \phi(s) + \sum_{s'} \sum_a \pi(a|s) p(s'|s, a) v_\pi(s') \\ &= \phi(s) + \sum_{s'} \omega(s \rightarrow s', 1) v_\pi(s') \\ &= \phi(s) + \sum_{s'} \omega(s \rightarrow s', 1) v_\pi(s') \\ &= \phi(s) + \sum_{s'} \omega(s \rightarrow s', 1) \left[\phi(s') + \sum_{a'} \pi(a'|s') \sum_{s''} p(s''|s', a') v_\pi(s'') \right] \\ &\hspace{15em} \text{Unrolling following (B-3)} \\ &= \phi(s) + \sum_{s'} \omega(s \rightarrow s', 1) \phi(s') + \sum_{s''} \omega(s \rightarrow s'', 2) v_\pi(s'') \\ &= \sum_{x \in S} \sum_{k=0}^{\infty} \omega(s \rightarrow x, k) \phi(x) \quad \text{After continuous unrolling of the value function} \end{aligned}$$

The gradient of the objective function to maximize is $\nabla J(\theta) = \nabla v_\pi(s_0)$:

$$\begin{aligned}
\nabla J(\theta) &= \nabla v_\pi(s_0) \\
&= \sum_s \sum_{k=0}^{\infty} \omega(s_0 \rightarrow s, k) \phi(s) \\
&= \sum_s \eta(s) \phi(s) \\
&= \left(\sum_s \eta(s) \right) \sum_s \frac{\eta(s)}{\sum_s \eta(s)} \phi(s) \\
&= \left(\sum_s \eta(s) \right) \sum_s \mu_\pi(s) \phi(s) \\
&\propto \sum_s \mu_\pi(s) \phi(s) \\
&= \sum_s \mu_\pi(s) \sum_a \nabla \pi(a|s) q_\pi(s, a)
\end{aligned} \tag{B-4}$$

□

Where $\mu_\pi(s)$ is the on-policy distribution and describes the normalized fraction of time spent in each state.

C. Reinforcement Learning

Hyper-parameters

Description about hyper-parameters:

1. **agent_gamma**: Discount factor used for future rewards and training process. Most of the models have a better performance when this value is large due to the importance given to future rewards.
2. **checkpoint_interval**: Number of episodes or samples (for DQN) taken before creating a checkpoint.
3. **collect_episodes_per_iteration**: Number of episodes taken per training iteration.
4. **collect_steps_per_iteration**: Number of steps or trajectories taken per training iteration.
5. **epsilon_greedy**: Epsilon parameter for exploration in the DQN strategy.
6. **importance_ratio_clipping**: Ratio clipping value for the PPO algorithm.
7. **num_epochs_per_training**: Number of epochs used to train the model per iteration.

Name	agent_gamma	checkpoint_interval	collect_episodes_per_iteration
PPO_quiet-breeze-67	0,90	1000	150
PPO_daily-field-66	0,90	1000	50
DQN_firm-smoke-61	0,90	500	
DQN_whole-bush-48	0,90	500	
REI_stellar-frog-50	0,90	1000	12
REI_worldly-pond-8	0,48	1000	12

Name	collect_steps_per_iteration	epsilon_greedy	importance_ratio_clipping
PPO_quiet-breeze-67			0.2
PPO_daily-field-66			0.2
DQN_firm-smoke-61	500	0,3	
DQN_whole-bush-48	500	0,3	
REI_stellar-frog-50			
REI_worldly-pond-8			

Name	model_type	num_epochs_per_training
PPO_quiet-breeze-67	PPO Clip	10
PPO_daily-field-66	PPO Clip	10
DQN_firm-smoke-61	DQN	1
DQN_whole-bush-48	DQN	1
REI_stellar-frog-50	REINFORCE - Baseline	1
REI_worldly-pond-8	REINFORCE - Baseline	1

Table C-1.: Reinforcement Learning Hyper-parameters

Bibliografía

- [Anderson, 1984] Anderson, T. (1984). *An Introduction to Multivariate Statistical Analysis*. Wiley series in probability and mathematical statistics. John Wiley & Sons.
- [Biewald, 2020] Biewald, L. (2020). Experiment tracking with weights and biases. Software available from wandb.com.
- [Birnbaum, 1968] Birnbaum, A. L. (1968). Some latent trait models and their use in inferring an examinee's ability. *Statistical theories of mental test scores*.
- [Casella and Berger, 2002] Casella, G. and Berger, R. (2002). *Statistical Inference*. Duxbury advanced series in statistics and decision sciences. Thomson Learning.
- [Chalmers, 2012] Chalmers, R. P. (2012). mirt: A multidimensional item response theory package for the r environment. *Journal of Statistical Software, Articles*, 48(6):1–29.
- [Chalmers, 2016] Chalmers, R. P. (2016). Generating adaptive and non-adaptive test interfaces for multidimensional item response theory applications. *Journal of Statistical Software, Articles*, 71(5):1–38.
- [Chang and Ying, 1996] Chang, H.-H. and Ying, Z. (1996). A global information approach to computerized adaptive testing. *Applied Psychological Measurement*, 20(3):213–229.
- [Fisher, 1925] Fisher, R. A. (1925). Theory of statistical estimation. *Mathematical Proceedings of the Cambridge Philosophical Society*, 22(5):700–725.
- [Fletcher, 1987] Fletcher, R. (1987). *Practical Methods of Optimization*. Number v. 2 in A Wiley-Interscience publication. Wiley.
- [Frey et al., 2016] Frey, A., Seitz, N.-N., and Brandt, S. (2016). Testlet-based multidimensional adaptive testing. *Frontiers in Psychology*, 7.
- [Guadarrama et al., 2018] Guadarrama, S., Korattikara, A., Ramirez, O., Castro, P., Holly, E., Fishman, S., Wang, K., Gonina, E., Wu, N., Kokiopoulou, E., Sbaiz, L., Smith, J., Bartók, G., Berent, J., Harris, C., Vanhoucke, V., and Brevdo, E. (2018). TF-Agents: A library for reinforcement learning in tensorflow. <https://github.com/tensorflow/agents>. [Online; accessed 25-June-2019].

- [Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.
- [Kakade and Langford, 2002] Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *IN PROC. 19TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING*, pages 267–274.
- [Kelderman and Rijkes, 1994] Kelderman, H. and Rijkes, C. P. M. (1994). Loglinear multidimensional IRT models for polytomously scored items. *Psychometrika*, 59(2):149–176.
- [Lord, 1980] Lord, F. (1980). *Applications of Item Response Theory to Practical Testing Problems*. L. Erlbaum Associates.
- [Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–33.
- [Moroney, 2020] Moroney, L. (2020). *AI and Machine Learning for Coders*. O’Reilly Media, Incorporated, Sebastopol.
- [Mulder and van der Linden, 2008] Mulder, J. and van der Linden, W. J. (2008). Multidimensional adaptive testing with optimal design criteria for item selection. *Psychometrika*, 74(2):273–296.
- [Reckase, 2009] Reckase, M. (2009). *Multidimensional Item Response Theory*. Statistics for Social and Behavioral Sciences. Springer New York.
- [Schulman et al., 2015] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015). Trust region policy optimization. *CoRR*, abs/1502.05477.
- [Segall, 1996] Segall, D. (1996). Multidimensional adaptive testing. *Psychometrika*.
- [Sutton and Barto, 2018] Sutton, R. and Barto, A. (2018). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press.
- [Simpson, 1978] Simpson, J. B. (1978). A model for testing with multidimensional items.
- [van Hasselt et al., 2015] van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning.
- [Veldkamp and Linden, 2002] Veldkamp, B. and Linden, W. (2002). Multidimensional adaptive testing with constraints on test content. *Psychometrika*, 67:575–588.
- [Watkins, 1989] Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King’s College, Oxford.

[Wiering and van Otterlo, 2012] Wiering, M. and van Otterlo, M. (2012). *Reinforcement Learning: State-of-the-Art*. Adaptation, Learning, and Optimization. Springer Berlin Heidelberg.