Personalized Intelligent Tutoring System using Reinforcement Learning

Ankit Malpani
Microsoft Corporation, India

B. Ravindran and **Hema Murthy**
Department of Computer Science and Engineering, IIT Madras

Abstract

In this paper, we present a Personalized Intelligent Tutoring System that uses Reinforcement Learning techniques to implicitly learn teaching rules and provide instructions to students based on their needs. The system works on coarsely labeled data with minimum expert knowledge to ease extension to newer domains.

Introduction

A Personalized Intelligent Tutoring System (ITS) presents individualized inputs based on a students needs and learning behaviour. In this paper, we use Reinforcement Learning (RL) techniques to implicitly train the ITS with an adaptive student model that estimates the student's learning pattern. RL has the advantage of requiring no training data to learn - it essentially bootstraps which suits us due to the unavailability of student specific labelling of data. As a proof of concept, we show that - using a coarsely labeled dataset our tutoring system is able to significantly improvement student's learning when compared against a non-intelligent tutoring system.

Based on its knowledge of the topic and experience gained during student interactions, the teaching system needs to suggest more relevant inputs to a student. To do this, an ITS must have a representation and understanding of the (i) topic or subject being taught (Knowledge/Data module), (ii) student being taught (Student Model) and (iii) methods of instruction to present the inputs to a student in an optimal way (Pedagogic module). The knowledge module consists of problems with solutions (or hints) through which a student learns, a measure of difficulty level, details of the skills required to solve a problem, relations or dependencies between different problems and topics etc. The student model is based on prior knowledge about students learning habits, external environment and on what constitutes skills and proficiency in a topic. The Pedagogic module represents the knowledge a tutor has about the methods of instructions, the optimal way of presenting it to students - the basis on which the ITS suggests questions to a student. It is for learning this Pedagogic module i.e to implicitly train an ITS to teach - that we use RL.

ITS as an RL problem

We look at the ITS as a finite horizon problem i.e. each episode lasts for a fixed finite time. To minimize action choices and labeling, each question in the data-set is categorized into different types based on its difficulty level along with a weightage (w_i) for each type. The weightage defines the importance of the question type in the final evaluation test-paper. At each step the ITS presents a question, records the student's answer and then presents the solution to enable the student to learn. The aim is to present questions such that they maximize the student's learning at the end of the episode. Formulating the ITS as an RL problem allows it to be divided into 4 logical parts as shown in figure 1.

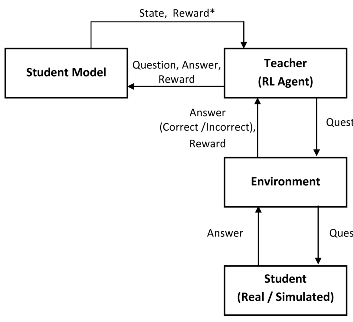


Figure 1: Problem Overview

The Teacher is an RL agent that interacts with the student through the environment suggesting a question-type (the action) based on the current state of the student. At the next time step, it receives an answer from the student and a corresponding reward.
The Environment implements the action suggested by choosing a question from the category selected by the agent. The Student component represents the student being taught which can either be a real or a simulated student.
The Student Model serves as a state description for the RL agent - providing an estimate of the student state and also a modified reward based on the observations from the environment. The state of the system is estimated with the tuple $< \{p_i | i \epsilon quesType\}, n >$ where p_i is the probability of the student answering a question of i^{th}-type correctly and n is the fraction of the number of questions left to be shown in the episode. Note that the p_is indicate the state of the student. After every k^{th} step, the student model provides a weighted reward $\frac{t}{N} \sum w_i p_i(t)$ to the agent where $p_i(t)$ is the probability of answering question of i^{th}-type correctly at time t in the episode. Using a weighted reward based on the objective function, instead of just based on the outcome of the immediate action, allows the agent to maximize return (long term reward) at the cost of immediate rewards.
It would be difficult for an agent to learn its policies quickly enough from a real student. Hence, the agent is initially trained against a simulated student allowing it to explore different policies and learn; circumventing the risk of having an adverse effect on a real student. It is with this learned policy and the learning mechanism still in place that the agent in-

teracts with the real student to fine-tune its performance to fit each student better. Depending on the current state of the student, the simulated student answers the question, views the solutions and updates its state according to equations

$$p_i(t+1) = p_i(t) + c \cdot X_{i,j} \tag{1}$$

$$p_i(t+1) = p_i(t) + w \cdot X_{i,j} \tag{2}$$

where, question of type j is shown. c, w are constants and

$$X_{i,j} = \alpha_{j,i} \frac{\sum_{k<j} p_k(t)}{\sum_{k<j}} \cdot (1 - p_i(t))$$

$\alpha_{j,i}$ are constants. Equation 1 is used for questions' answered correctly and equation 2 for questions' answered incorrectly. $X_{i,j}$ helps takes into account the current state of the student while performing an update.

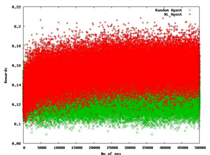
RL Agent

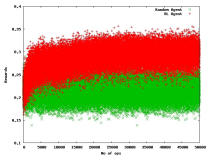
We adapt our RL Agent to use an Actor-Critic algorithm based on (Konda and Tsitsiklis 2000) where the RL Agent has two components -The Actor and the Critic. The Critic follows a value function based method - learning $Q(s, a)$ for each state s while the actor follows a Policy Gradient approach with parameters representing the preference of choosing actions. The critic calculates the error between the value of state obtained in the current run and the estimated value of the state. This error is then used by the Actor to update its preferences and refine its policy. The algorithm selects actions using a soft-max policy (Sutton and Barto 1998) with critic Q-values factored into it.

The student model needs to estimate the state of the student at every step i.e. it needs to estimate the start state and the update parameters. As relatively few interactions are available with a real student, estimating each of these parameters individually for each student is infeasible. Instead, we use a set of student models with different parameter values representing different types of students and following a set of update equations with their respective parameter values. The problem now reduces to selecting one of these representative models for the student as interaction proceeds. The simulated student on the other hand could follow update rules completely unknown to the student model.

We utilize the transformation selection mechanism used in (Ravindran and Barto 2003) to select the student model. A Bayesian Estimate of each model being the correct student model is maintained. At each time-step the best model according to the current estimate is selected moving to this model's estimated current state and suggesting actions based on the selected model's policy. To estimate the start state of the student a set of questions of different types are presented to the student - recording their answers without disclosing the solutions. The student start state is estimated by the frequency of questions correctly answered.

Figures 2 to 5 compare learning achieved with our RL agent (in red) against a random agent (in green) at the end of each lesson. Though we require atleast one lesson to select a student model we still observe some learning at the end of first lesson. This can be attributed to learning made through generalizations from states observed in subsequent lessons where a model has already been selected. We find that the more a student interacts with the system the more improvement is observed in his learning. Also, as the student becomes more proficient in a topic the agent minimizes variance such that the final reward values are mostly near the best case reward values of a random-agent. As expected, we find that maximum learning is observed for a Below-Average student and least learning for an Excellent student.


(a) After Lesson 1


(b) After Lesson 2

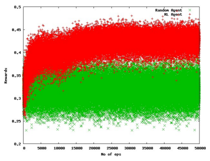
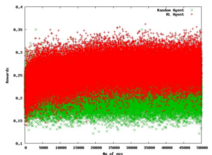
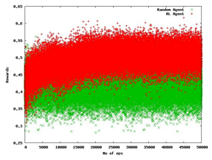

(c) After Lesson 3

Figure 2: *Below-Avg Student*


(a) After Lesson 1


(b) After Lesson 2

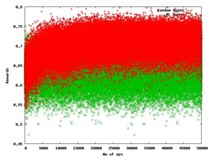
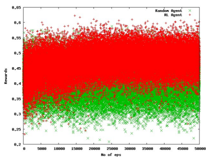
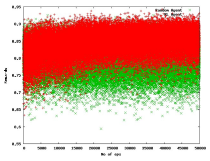

(c) After Lesson 3

Figure 3: *Average Student*


(a) After Lesson 1


(b) After Lesson 2

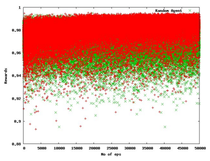
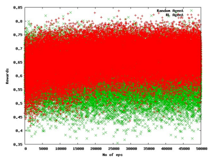
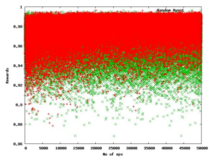

(c) After Lesson 3

Figure 4: *Above-Avg Student*


(a) After Lesson 1


(b) After Lesson 2

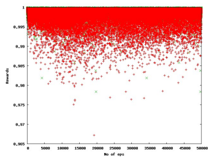

(c) After Lesson 3

Figure 5: *Excellent Student*

References

Konda, V., and Tsitsiklis. 2000. Actor-critic algorithms. In *SIAM Journal on Control and Optimization*, 1008–1014.

Ravindran, B., and Barto, A. G. 2003. Relativized options: Choosing the right transformation. In *ICML*, 608–615.

Sutton, R. S., and Barto, A. G. 1998. RL : Introduction.