Using Q-Learning to Personalize Pedagogical Policies for Addition Problems

Takara Truong [1] **Amanda Shen** [1] **Cortney Weintz** [1]

Abstract

The prevalence of COVID-19 over the past year has illuminated the need for effective digital education tools. With students studying from home, teachers have struggled to provide their students with adequately challenging coursework. Our project aims to solve this issue in the context of math. More specifically, our goal is to encourage thoughtful learning by supplying students with personalized two-number addition problems that take time to solve but that we expect the student can still answer correctly. Our solution is to model the process of selecting a math problem to give a student as a Markov Decision Process (MDP) and then use Q-learning to determine the best policy for arriving at the most optimally challenging two-number addition problem for that student. The project creates three student simulators based on group member data. We show that it took student one: (162 ± 134) iterations to give appropriate level problems where the first entry is mean and the second is standard deviation. Student two took (230 ± 205) iterations, and student three took (247 ± 236) iterations. Lastly, we demonstrate that pre-training our model on students two and three and testing on student one showed a significant improvement from (162 ± 134) iterations to (35 ± 44) iterations.

1. Introduction

With the onset of the COVID-19 pandemic at the start of 2020, students, teachers, and tutors have overwhelmingly been forced to adapt to an electronic learning (E-learning) environment. No longer do students have immediate access to their instructors during class hours or the ability to freely utilize classroom resources. Accordingly, this abrupt change in educational practices has presented numerous challenges to the learning experience (Irfan et al., 2020). Many schools and tutoring services have adopted learning management systems to administer E-learning. Unfortunately, however, many students have been dissatisfied with the quality of available learning resources and even more so with the quality of online classes (Sarwar et al., 2020).

Regardless, E-learning has potential. The ability to easily accommodate individual student needs, allow repeated access to class material, and conform to each student's ability level provides promise in the field of E-learning for intelligent tutoring systems (ITS).

Although E-learning via ITSs offers potential for improved learning (VanLehn, 2011), providing students with adequately challenging coursework that doesn't exacerbate negative affects (e.g. frustration) has proven to be increasingly difficult (Borracci et al., 2020).

1.1. Related Works

Efforts to create personalized ITSs using reinforcement learning (RL) have broadly come to one of two conclusions: (1) that applying RL to the task is unfeasible because the system requires extensive experience to learn to teach properly (Iglesias et al., 2003; Beck, 2001) or (2) that learning an initial value function by training on simulated students can reduce the experience required to learn an accurate pedagogical policy (Iglesias et al., 2009a;b; Carroll & Peterson, 2002).

Additional efforts, such as that of Walkington & Bernacki, have tried to personalizes math problems based on student interest (2019) (e.g. giving a student that plays baseball a word problem about baseballs instead of apples). Further, Borracci et al. provides a model for an ITS that assists students with algebraic problems rather than learning the appropriate difficulty of the problem itself (2020).

These ITSs tend not to personalize well due to the difficulty of defining a reward function that adequately captures relevant aspects of a student's learning. For example, Beck defined a reward function based on minimizing time spent on each problem which resulted in more shallow learning (2001). They do, however, personalize better when trained on both simulated and real student data and define rewards

[*]Equal contribution [1]Computer Science, Stanford University, Stanford, CA. Correspondence to: Takara Truong <takaraet@stanford.edu>, Amanda Shen <amshen@stanford.edu>, Cortney Weintz <cweintz@stanford.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

based on student performance as shown by the work of Iglesias et al. (2009b). Thus, we turned our attention for this paper to the reward function, defining it to target both the correctness of the user's response and the time it took them to arrive at their solution.

1.2. Our Contribution

Our algorithm addresses the topic of two-number addition, aiming to provide users with math problems that they will spend the most amount of time working on while still arriving at the correct answer. The input to our algorithm is the user's answer and the amount of time they spent working on the problem. We then use Q-learning to determine the next problem to supply the user, with the goal of maximizing the expected time spent to correctly answer the question – encouraging rich learning outcomes. The algorithm grows with the user and changes as the student progresses making it ideal for this situation.

2. Methods

This paper uses the Markov Decision Process (MDP) as a framework to model the problem. Mathematically, MDPs are 4-tuples (S, A, T, R) of states, actions, transition and successor function, reward function. The Q-learning algorithm defines an agent whcih navigates the environment defined by the MDP. By taking an action $a \in A$ in state s, the agent will transition to a new state s'. Any action taken from a state will result in the environment providing the agent with a reward $R(s, a, s')$. The objective of the agent is to maximize its total reward, in other words, to maximize the reward function. It accomplishes this by taking the action from a given state which maximizes the expected sum of future rewards. Restated, the agent determines its current action based upon the potential for future rewards where the potential reward is a weighted sum of the expected value of each future reward. For this paper, we defined relevant MDP elements as:

- $S :=$ **state space.** Set of state representations for each difficulty level of addition problems.
- $s_0 :=$ **start state.** The base case problem of lowest degree difficulty. These problems are zero added to any constant.
- $A :=$ **action function.** Set of all valid actions that allow transitioning between states (difficulty levels). For this MDP, a valid action is a one hot vector that can move a single feature up or down a level.
- $T :=$ **transition and successor function.** Deterministic transition from current state to next state given an action, $\forall s \in S. \forall a \in A. \exists! s' \in S. (T(s, a, s') = 1)$.
- $R :=$**reward function.** The difference between the response time at time t and and the predicted response time at time $t + 1$. Also, if the user gets the problem incorrect, the difference returned will be negative.

To start, the Q-learning algorithm will decide the next action to take based on the expected future reward that comes as a result of each subsequent step. The weight for each step is scaled by the discount factor, thus, the further a reward is from the current state, the lower it is valued. The algorithm, therefore, contains a function $Q : S \times A \to \mathbb{R}$, which maps a state–action pair to a real number, where Q is initialized as arbitrary constants. Then, for each time t, the agent chooses an action a_t to take from state s_t, receives a reward r_t and enters a new state s_{t+1}. Lastly, the agent updates the Q-table according to the update rule. This table's rows represent states and columns represent features, denoted as (s_t, a_t), and the the values are Q-scores $Q(s_t, a_t)$ defined as

$$Q^{new}(s_t, a_t) := Q^{old}(s_t, a_t) + \alpha \left[r_t + \gamma \cdot \max_{a \in A} \{Q(s_{t+1}, a)\} - Q^{old}(s_t, a_t) \right]$$

where α is the learning rate, r_t is the reward at time t and γ is the discount factor. The updating rule updates the Q-table for a specific state–action pair by taking it's original score times the weighted temporal differences

$$\text{diff} = \underbrace{r_t + \gamma \cdot \max_{a \in A} \{Q(s_{t+1}, a)\}}_{\text{new value}} - \underbrace{Q^{old}(s_t, a_t)}_{\text{old value}}.$$

3. Experiments

The following experiments aim to identify (1) how quickly the algorithm can give an appropriately difficult problem to the user and (2) whether pre-training makes this process faster.

An appropriately difficult problem is given by the scenario in which the optimal policy for the current state is to self-loop and stay at that state. This indicates that there are no other states that will lead to a longer problem solving time. A separate Q-learning algorithm is employed to find the optimal policy where the appropriately difficult state can be found. From here the trainer is run from scratch and completes when the optimal action is to stay in the same state for several iterations.

Table 1. State Representation Examples.

State Representation				Example Math Problems		
number 1 digit count	number 2 digit count	# of carry operations	# of zeros			
1	1	1	0	1+9	2+8	9+6
2	3	1	0	11+119	11+793	11+291
3	3	3	0	111+889	116+888	898+114
2	2	1	2	10+90	30+70	70+50

3.1. Features and Dataset

The math problems given to the user are additions of two natural numbers, each up to three digits. The Problem Generator produces all possible combinations of two three-digit integers and bins the problems based on a set of features. The math problems are characterized by the following features:

- First integer digit count
- Second integer digit count
- Number of required carry operations
- Number of zeros contained in both integers

A tuple of these features comprise each state in the definition of our MDP from which the Q-learning algorithm is ran on. Thus, each addition problem belongs to a particular state of feature combinations. Examples are given in Table 1.

For two three-digit addition problems, there are 1 million total possible problems that can be given. Our feature extractor bins these problems into 49 distinct states. To reduce the running time and improve user experience, the problem data are serialized to json and the generating processes are forced to be one-off jobs.

The data incorporates human-generated student data from which a simulated student is created and the algorithm trains on. For this project, we create a simulated student for each group member named: AmandaSim, CortneySim, TakaraSim. Each member was asked to answer four randomly generated addition problems for each state. As a handicap, all member solved the problems mentally. Once completed, a distribution of response time and probability of being correct was created for each state.

For experiment 2, we normalize the response time of each student's response to their max response time before creating a combined distribution.

3.2. Hyper Parameters

The simulation was ran with the following hyper-parameters:

- $\delta :=$ **convergence.** 5
- $\epsilon :=$**exploration factor.** 0.3
- $\gamma :=$ **discount factor.** 0.95
- $\alpha :=$ **step size.** .5

where δ is the limit on the number of iterations that the optimal action must stay in the appropriate level state, ϵ is the probability of taking a non-optimal action, γ is the discount factor, and α is the learning rate. This step size is kept constant because we do not want to guarantee convergence. The algorithm must adapt to the user as they progress and learn. All hyper-parameters were tuned heuristically.

4. Results

The first experiment aimed to answer how quickly the algorithm can give an appropriate level problem. Each experiment was ran 100 times per student. Outliers were defined by whether the point lay outside three standard deviations from the mean and removed from the analysis. The following results are reported as (mean $\pm$ std) and rounded. It took AmandaSim (162 ± 134) iterations to give appropriate level problems. CortneySim took (230 ± 205) iterations and TakaraSim took (247 ± 236).

The second experiment aimed to answer whether pre-training would make this process faster. Similar to the first experiment, the second experiment was ran 100 times, outliers were removed, and the result is reported as (mean $\pm$ std) and rounded. The algorithm was trained on the combined normalized data-set of TakaraSim and CortneySim and tested on AmandaSim. It took AmandaSim (35 ± 44) iterations to give appropriate level problems.

5. Discussion

Experiment 1 demonstrates that the algorithm can give appropriately difficult problems; however, it takes many iterations to arrive there. Experiment 2 shows that the algorithm converges faster when pre-trained. AmandaSim saw an improvement from 162 iterations to 35 iterations.

This observed improvement is reasonable since there are similar trends in what students find difficult. For instance, three digit problems will usually take longer to solve than two digit problems. Yet, inspection of the student data shows that each student has unique strengths and weaknesses. This is a likely explanation for why it takes several iterations for AmandaSim to adapt to the pre-trained Q-table from CortneySim and TakaraSim.

Since we have defined actions to keep subsequent states close (the following states are always a one hot vector away), the algorithm works well with incremental changes but has

difficulty narrowing large gaps. Most notably, if a user suddenly changes from being good at three digit additions to being good at one digit addition, the algorithm is unable to quickly unlearn and relearn because it must explore and update iteratively. This suggests a need for several pre-trained algorithms for different levels of users or a more expressive action function.

6. Conclusion and Future Work

In this project we share an adaptive algorithm that provides adequately challenging math problems for students. The next step is to evaluate the algorithm on a more varied group of students. Once complete, this project can make several improvements. The project can be easily scaled towards different types of problems such as subtraction, multiplication, and division. Additionally, this algorithm could be expanded towards solving algebraic problems or word-based math problems. Future work may also use deep-Q learning to alleviate the need for feature engineering.

7. Contributions

Takara Truong has taken lead on the project, bringing the long term objectives to the table and doing a lot of the group organizing and management. He also worked with Amanda to determine the features we are using to describe each math problem. Further, Takara developed our QLearning class that will be used in conjunction with our MDP class to learn which problems to give a student.

Amanda Shen has tackled much of the systems requirements, coming up with creative ways to better optimize our algorithm by storing feature data in an external file and constructing many useful data structures. Additionally, Amanda was responsible for determining how to convert a feature vector to an interpretable math problem that can be displayed to the user.

Cortney Weintz was very influential in determining how our group defined the MDP for our algorithm - specifying how to approach representing math problems as feature vectors and how to transition between states. He also implemented the "actions" function of our MDP class, making sure to account for invalid state representations of math problems.

References

Beck, J. E. *ADVISOR: a machine-learning architecture for intelligent tutor construction*. University of Massachusetts Amherst, 2001.

Borracci, G., Gauthier, E., Jennings, J., Sale, K., and Muldner, K. The effect of assistance on learning and affect in an algebra tutor. *Journal of Educational Computing Research*, 57(8):2032–2052, 2020.

Carroll, J. L. and Peterson, T. Fixed vs. dynamic sub-transfer in reinforcement learning. In *ICMLA*, pp. 3–8, 2002.

Iglesias, A., Martinez, P., and Fernández, F. An experience applying reinforcement learning in a web-based adaptive and intelligent educational system. 2003.

Iglesias, A., Martínez, P., Aler, R., and Fernández, F. Learning teaching strategies in an adaptive and intelligent educational system through reinforcement learning. *Applied Intelligence*, 31(1):89–106, 2009a.

Iglesias, A., Martínez, P., Aler, R., and Fernández, F. Reinforcement learning of pedagogical policies in adaptive and intelligent educational systems. *Knowledge-Based Systems*, 22(4):266–270, 2009b.

Irfan, M., Kusumaningrum, B., Yulia, Y., and Widodo, S. Challenges during the pandemic: Use of e-learning in mathematics learning in higher education. *Infinity Journal*, 9(2):147–158, 2020. ISSN 2460-9285. doi: 10.22460/infinity.v9i2.p147-158. URL http://e-journal.stkipsiliwangi.ac.id/index.php/infinity/article/view/1830.

Sarwar, H., Akhtar, H., Naeem, M., Khan, J., Waraich, K., Shabbir, S., Hasan, A., and Khurshid, Z. Self-reported effectiveness of e-learning classes during covid-19 pandemic: A nation-wide survey of pakistani undergraduate dentistry students. *European Journal of Dentistry*, 14: S34 – S43, 2020.

VanLehn, K. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221, 2011. doi: 10.1080/00461520.2011.611369. URL https://doi.org/10.1080/00461520.2011.611369.

Walkington, C. and Bernacki, M. L. Personalizing algebra to students' individual interests in an intelligent tutoring system: Moderators of impact. *International Journal of Artificial Intelligence in Education*, 29(1):58–88, 2019.