Learning Expert Models for Educationally Relevant Tasks using Reinforcement Learning

Christopher J. MacLellan
College of Computing and Informatics
Drexel University
Philadelphia, PA 19104
christopher.maclellan@drexel.edu

Adit Gupta
College of Computing and Informatics
Drexel University
Philadelphia, PA 19104
ag3338@drexel.edu

ABSTRACT

There has been great progress towards Reinforcement Learning (RL) approaches that can achieve expert performance across a wide range of domains. However, researchers have not yet applied these models to learn expert models for educationally relevant tasks, such as those taught within tutoring systems and educational games. In this paper we explore the use of Proximal Policy Optimization (PPO) [25] for learning expert models for tutoring system tasks. We explore two alternative state and action space representations for this RL approach in the context of two intelligent tutors (a fraction arithmetic tutor and a multicolumn addition tutor). We compare the performance of these models to a computational model of learning built using the Apprentice Learner architecture. To evaluate these models, we look at whether they achieve mastery and how many training opportunities they take to do so. Our analysis shows that at least one PPO model is able to successfully achieve mastery within both tutors, suggesting that RL models might be successfully applied to learn expert models for educationally relevant tasks. We find that the Apprentice model also achieves mastery, but requires substantially less training (thousands of times less examples) than PPO. Finally, we find that there is an interaction between the PPO representation and task (one representation is better for one tutor and the other representation is better for the other tutor), suggesting that the design of the state and action representations for RL is important for success. Our work showcases the promise of RL for expert model discovery in educationally relevant tasks and highlights limitations and challenges that need further research to overcome.

Keywords

Reinforcement Learning, Simulated Students, Expert Model Authoring

1. INTRODUCTION

Researchers have made great progress towards developing Reinforcement Learning (RL) models that can meet or exceed human skill at complex tasks across a broad range of domains. For example, the recently developed Proximal Policy Optimization (PPO) algorithm [25] can learn to play a broad range of Atari games at an expert level through trial and error. A team of five PPO-trained models can beat a team of five human professional champions at DOTA2, a collaborative online multiplayer battle arena game [4]. Researchers have applied a related RL approach called A3C [19] to develop agents that can beat top human experts at Starcraft2, a multiplayer real-time strategy game [28]. Finally, RL has been applied widely to the area of robotics and autonomous systems; e.g., RL models can fly an F16 to beat a expert human pilots in simulated 1v1 dogfights [7].

Despite these successes, there has been surprisingly little work exploring the applicability of RL to educationally relevant tasks, such as those found in K12 or higher education. We do not mean that RL has not been applied to support learning and education; in fact, there is a large amount of work exploring how RL can be applied to optimize students instructional sequences [9]. However, we assert that there has been very little exploration of how emerging RL approaches perform on the kinds of educationally relevant tasks that humans often engage in; e.g., learning math.

Given this gap we might ask, what are the benefits of applying RL to educationally relevant tasks? The recent work on computational models of learning highlights many possible benefits. First, machine learning agents can support researchers and instructional designers in authoring cognitive models [17, 30] and discovering knowledge component models [14] that can drive personalized learning technologies. Although RL models utilize different representations than more traditional expert-system models (e.g., statistical representations), learned models do still represent an expert model.[1] Thus, tutors might apply these models to provide feedback on student behaviors. Researchers and designers might also use machine learning agents to cognitively crash test instruction before more costly human trials [16, 15, 31].

Given these benefits, why has more work not explored the use of recent RL methods for these tasks? One possibility is that applying RL to educational tasks is not straightforward. There exist toolkits, like GymAI [5], MuJoCoEnv [23], and PyBullets [6], for interfacing RL algorithms with simulation

[1]An expert model maps states to correct next action(s).

environments and games platforms like Atari, StarCraft2, and DOTA2. These toolkits have powered the explosion of RL research. Unfortunately, no such interfaces exist for educational tasks, such as those found in intelligent tutoring systems or educational games. Also, many educational tasks do not fit cleanly into the standard RL paradigm of observing the state, choosing an action, receiving a reward, and repeating; e.g., many tutors let learners request hints and worked examples, which RL systems cannot leverage.[2]

Beyond these challenges, it is possible that the tasks themselves are less attractive for RL research. For example, educationally relevant tasks, such as fraction arithmetic, seem simple when compared to tasks such as flying an F16 or playing DOTA2. It is possible that these tasks do not challenge current RL systems. However, it is also possible that these tasks present challenges that have prevented researchers from successfully applying RL to them. For example, tutor tasks often have much larger action spaces than game-based tasks where RL has been successfully applied (e.g., actions for inputting the numbers 1-50 in any interface field vs. six buttons on an Atari controller).

In this paper, we set out to investigate these ideas and to lay a foundation for future research programs to apply RL to the kinds of educationally relevant tasks that humans regularly engage with. Our goal is not to show that RL provides a good model of human learning and behavior (we do not think that it does). Instead, we simply aim to show how RL methods might be applied to tasks relevant to human learning. Our hope is that RL approaches might offer new means for authoring and evaluating educational technologies.

To support these investigations we present TutorGym, an open-source toolkit for interfacing RL agents (as well as other kinds of machine learning agents) with intelligent tutoring system tasks. This toolkit lets us apply RL models to two educational environments: a fraction arithmetic tutor and a multicolumn addition tutor. We developed two PPO models that vary in their features and action representations. We also compared these models to a previously developed Apprentice Learner model [16], which is a more cognitively inspired model of how people learn from examples and feedback within intelligent tutors. We conducted a factorial study design where we applied these three models to our two educational tasks. Our key findings are:

1. The PPO models are able to achieve mastery at these tasks, suggesting that they do generalize from games and robotics tasks to educationally relevant tasks;
2. The PPO models require much more training than our Apprentice Learner model to achieve mastery (thousands of times more training), even when we provide PPO with the same background knowledge as Apprentice (the PPO-Operator variant). This suggests that human-like models, such as Apprentice, are more efficient than PPO.
3. We find that there is an interaction between PPO's representation and the task, suggesting that representation is central to RL performance and that it needs to be tailored for each task.

[2]Inverse RL [1] can learn from expert examples, but typically this is done offline in batch rather than interactively interleaved with RL.

We claim that there is an synergistic opportunity to do research at the intersection of RL and education that has not yet been fully explored and this paper aims to lay the foundation for these future explorations. There are many potential ways that educational data mining and learning analytic communities might benefit from the development and use of RL models, such as PPO. Similarly, there is an opportunity to improve RL by exploring its application to the kinds of educationally relevant learning tasks that humans engage in during K12 and higher education.

2. BACKGROUND

2.1 OpenAI Gym

OpenAI Gym is an open-source toolkit for RL development [5]. Gym provides an standardized interface for applying RL to tasks. An environment created with Gym has standardized state and action descriptions and supports methods, for querying the state, taking an action, and collecting rewards. Gym currently supports multiple environments such as robot simulations or Atari games. Our research builds on Gym, so that we can directly interface existing RL implementations with educationally relevant tasks without having to create custom implementations.

2.2 Proximal Policy Optimization (PPO)

PPO is a deep RL algorithm that was recently developed by OpenAI [25]. It is a policy gradient method that achieves state-of-the-art performance across many tasks. We chose PPO over alternatives, such as TRPO [24] and ACER [29], because it supports a broad range of state and action representations and is much easier to tune than alternatives. For this work, we use the stable-baselines3 implementation of the PPO algorithm, which has verified performance on multiple RL benchmarks [22].

2.3 Apprentice Learner Architecture

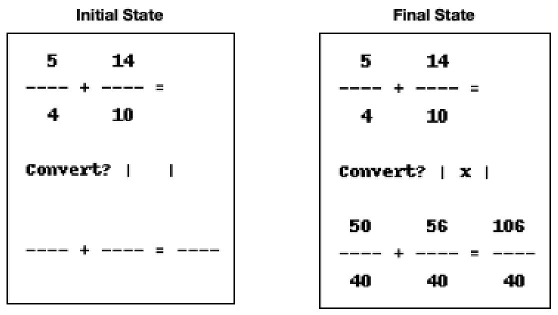
The open-source Apprentice Learner Architecture [16] generalizes prior simulated student models [13, 18] and provides a platform for investigating and comparing alternative simulated student models. Apprentice models have been successfully applied to learn expert models for 8 different tutor tasks spanning multiple domains (math, language, chemistry, and engineering) [17]. Emerging work explores the use of Apprentice models for supporting domain experts, such as teachers, in authoring tutors through teaching rather than programming [30]. In this work, we use one of the standard Apprentice models as a baseline for evaluating the PPO models because it have been successfully applied in previous work to learn expert models for educationally relevant tasks. For a complete description of this model see [17].

3. TUTORGYM

To support the development of machine learning agents we created TutorGym, a toolkit that provides a machine interfaces for multiple tutor environments.[3] TutorGym leverages the Gym [5] to enable existing RL implementations (that support Gym) to interface with these environments.

Our toolkit extends Gym to enable agents to request worked examples. Tutors generate both next-step hints and feedback, so the examples are automatically generated by the

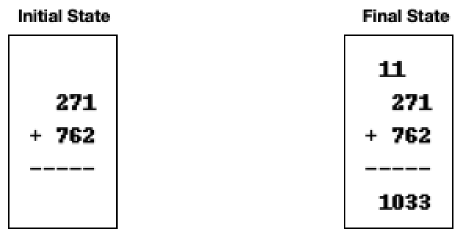
[3]We have open-sourced TutorGym under an MIT license and made it publicly available here: https://tutorgym.ai



```
{ initial_num_left: "5",
  initial_denom_left: "4",
  initial_operator: "+",
  initial_num_right: "14",
  initial_denom_right: "10",
  check_convert: "",
  convert_num_left: "",
  convert_denom_left: "",
  convert_num_right: "",
  convert_denom_right: "",
  answer_num: "",
  answer_denom: "" }
```

```
{ initial_num_left: "5",
  initial_denom_left: "4",
  initial_operator: "+",
  initial_num_right: "14",
  initial_denom_right: "10",
  check_convert: "x",
  convert_num_left: "50",
  convert_denom_left: "40",
  convert_num_right: "56",
  convert_denom_right: "40",
  answer_num: "106",
  answer_denom: "40" }
```

Figure 1: Fractions tutor, as rendered within TutorGym with its underlying base feature representation.



```
{ thousands_carry: "",
  hundreds_carry: "",
  tens_carry: "",
  upper_hundreds: "2",
  upper_tens: "7",
  upper_ones: "1",
  lower_hundreds: "7",
  lower_tens: "6",
  lower_ones: "2",
  operator: "+",
  answer_thousands: "",
  answer_hundreds: "",
  answer_tens: "",
  answer_ones: "" }
```

```
{ thousands_carry: "1",
  hundreds_carry: "1",
  tens_carry: "",
  upper_hundreds: "2",
  upper_tens: "7",
  upper_ones: "1",
  lower_hundreds: "7",
  lower_tens: "6",
  lower_ones: "2",
  operator: "+",
  answer_thousands: "1",
  answer_hundreds: "0",
  answer_tens: "3",
  answer_ones: "3" }
```

Figure 2: Multicolumn tutor, as rendered within TutorGym with its underlying base feature representation.

underlying tutor. As RL models only learn from feedback on actions, these interactions are mainly used by the Apprentice models, which learn from both examples and feedback.

TutorGym logs agent interactions in DataShop format [12], which is a common educational data format. Outputting data in this format lets us analyze it using the same techniques used to analyze human tutor data. In particular, it lets us conduct learning curve analysis to investigate agents' first attempt correctness as they receive more practice.

We implemented two tutors within TutorGym: a fraction arithmetic tutor [21] and a multicolumn addition tutor [30]. We chose these tutors because they exhibit interesting state and action spaces characteristics that are relevant to our analysis of emerging RL approaches.

3.1 Fraction Arithmetic Tutor

This tutor was used to study both human [21] and agent [16] learning. It presents students with three kinds of fractions problems: addition with the same denominator, addition with different denominators, and multiplication. The students check a box indicating whether they need to convert to common denominators before solving. If the fractions need to be converted, then they input values into the conversion fields. The tutor requires students to convert fractions to common denominators using cross multiplication. If students do not need to convert, then they directly enter values into the final fraction fields. Figure 1 shows the visual representation of the tutor state generated by TutorGym along with the underlying attribute-value representation that it maintains internally. The tutor gives students randomly generated problems where the initial numerators range from 1-15, the initial denominators range from 2-15, and the type of problem can be either addition or multiplication. There is also an "easy" version of the tutor that generates a much smaller range of numbers (numerators range from 1-5 and denominators range from 2-5). The tutoring system also has a done button (not shown) that the agent can select and it can provide worked examples on request.

3.2 Multicolumn Addition Tutor

The second tutor was used in previous research on simulated students [30]. It presents students with two numbers to add, with each digit presented in its own field. To compute the solution, the students needs to add and carry values where necessary. The tutor requires students to enter the answer values right to left, carrying where necessary. The tutor will mark an answer incorrect if they have not yet filled in the answer field to the right or they have not yet carried over a value from the previous column (if required). Figure 2 shows a simple visual output created by TutorGym along with the underlying attribute-value representation. The tutor also has a done button (not shown) and can provide worked example on demand.

4. LEARNING MODELS

4.1 Apprentice Learner

We created three alternative learning models to train within the TutorGym tutors. We built our first model using the Apprentice Learner architecture [16, 30]. From this architecture, we used the an Apprentice model developed in prior work [17]. For each tutor, we provided the apprentice model with background relational knowledge (for augmenting the state description) and primitive operators (for explaining demonstrations). For the fractions tutor, we provided equality knowledge, which adds features to the state description for each pair of fields denoting whether they have equal values. We also provided three primitive operators: copy, add, and multiply, which give the agent the ability to copy, add, and multiply values from the interface.

For the multicolumn tutor, the knowledge was slightly more complicated. We added four relational knowledge operators: add2-ones, add2-tens, add3-ones, and add3-tens. The first two add the values from every pair of fields in the interface and add features to the state denoting the ones component and tens component of each sum. Add3-ones and add3-tens do the same, but for every triplet of fields. This provides the agent with the ability to determine if a column of numbers (either of length two or three) will generate a value that needs to be carried or not. We also added these exact same operators as primitive operators, so the agent can use them to explain and perform the actual steps of computing each column sums and generating the appropriate carry values.

4.2 PPO-Number

A PPO model is defined in terms of its state (input) and action (output) representations. For the fractions and multicolumn tutors, the PPO-Number model makes use of the base state-representations shown in Figures 1 and 2. To convert these representations into a format that is acceptable to PPO (a fixed-length feature vector), we used an approach called one-hot encoding. Under this scheme, every unique attribute-value pair from the state is mapped to a particular feature in a feature vector. If the attribute-value is present in the state, then the feature is a 1, otherwise it is a 0.

Unfortunately, precomputing all possible attribute-values is non-trivial. To address this issue, we created an *online* one-hot encoder that always outputs a vector with a fixed length preselected by the user. Whenever the encoder encounters a new attribute-value, it maps it to a previously unused feature within the vector. After a mapping between an attribute-value and a feature has been made, that feature is only ever used to represent that particular attribute-value. This scheme enables the use of RL approaches that expect fixed-length vectors (PPO) even though the system might encounter a large number of sparse features that are not known in advance. The end result is that states from the fraction and multicolumn tutors are mapped to fixed length feature vectors, where every feature is either a 0 or a 1 (e.g., the initial state from Figure 2 would have a feature for *upper_tens* = 7 with a value of 1). For the fractions tutor, 2000 features was sufficient to describe states in the standard tutor and 900 features was sufficient to describe the states in the easy tutor (with a smaller set of problems). For the multicolumn tutor, 110 features were sufficient.

Given this state representation, PPO-Number utilizes a multidiscrete output. This type of output has multiple independent discrete action outputs; e.g., in Atari it might have an output for the arrow pad (left, right, up, down) and another output for the action action buttons (A, B, or None). PPO-Number also has two outputs: one that outputs a field to enter a value into (e.g., answer_num) and a second that outputs a number to enter into that field (e.g., 1).

For the fractions tutor, there are eight fields that can be selected for input and there are 450 possible numbers that can be entered into one of these fields (1-450). For the easy version of the tutor, there are only 50 possible numbers (1-50). Taken together, this means that the standard tutor has 3,600 unique actions ($8 \times 450 = 3600$) and the easy tutor has 400 unique actions ($8 \times 50 = 400$). There are slightly less actions in practice because outputs are ignored in certain cases; if the system selects the done or the check_convert fields, than the number component is ignored.

For the multicolumn tutor, there are also eight possible fields that can be selected. Each field represents a single digit, so there are only 10 numbers that can be input into each field (0-9). This yields a total action space of 80 actions ($8 \times 10 = 80$). Similar to fractions, the total is slightly less in practice because the number component of the output is ignored when the done button is selected.

4.3 PPO-Operator

This model uses a different state and action representation from PPO-Number. The representation aims to mirror the representation used by the Apprentice model. We apply the relational knowledge used by Apprentice to augment the base state representation from each tutor. In the fractions tutor, we apply the equality relation to add an additional feature describing which pairs of fields are equal. For the multicolumn tutor, we apply the add2-one, add2-tens, add3-ones, and add3-tens relations to compute the ones and tens values for the sums of every unique pair and triple of values from the tutor fields. We applied the same one-hot encoding approach used for PPO-Number to convert attribute-values into fixed-length feature vectors. We increased the size of the feature vectors to support the combinatorial number of additional relational features (2000 for fractions and 5000 for multicolumn).

The action space is multidiscrete, but the number and type of outputs are slightly different from PPO-Number. For the fractions tutor, the model has four outputs. The first is similar to PPO-Number's selection output, it identifies the field to update with a result. There are eight possible fields that might be updated by an action. The second output corresponds to an operator to apply. The operators are the same as those available to Apprentice: copy, add, or multiply. The remaining two outputs correspond to fields in the interface that provide the two argument for each operator and there are ten possible fields that can be used for either of these arguments. Using this scheme, an agent might choose to update the answer_num field using the add operator, and it might provide the initial_num_left and initial_num_right as arguments. There are 2400 possible unique actions under this representation ($8 \times 3 \times 10 \times 10 = 2400$). However, this number is smaller in practice. If the model chooses to update the done or check_convert fields than the reset of the action outputs are ignored. Additionally, if the model chooses to use the copy operator, than only the first argument is used (the second is ignored).

For the multicolumn tutor there are five outputs instead of four. The first corresponds to a field to update (there are eight possible fields). The second corresponds to the operator to apply. There are five operators corresponding to those used by Apprentice: copy, add2-tens, add2-ones, add3-tens, add3-ones. Finally, there are three argument fields because some of the operators (add3-tens and add3-ones) take three arguments. There are 13 possible options for each argument. With these outputs, there are 87,880 unique actions ($8 \times 5 \times 13 \times 13 \times 13 = 87880$). In practice this number is much smaller because if the done field is updated, then all the other outputs are ignored. Similarly if the copy operator is selected, than only the first argument is used (second and

Model	Domain	# Inputs	# Discrete Outputs
Number	Fractions	2000	8, 450
Number	Fractions-Easy	900	8, 50
Number	Multicolumn	110	8, 10
Operator	Fractions	2000	8, 3, 10, 10
Operator	Multicolumn	5000	8, 5, 13, 13, 13

Table 1: Size of PPO model input/output for each task.

third arguments are ignored). Finally, if the add2-tens or add2-ones operator are selected, than the third argument is ignored. Table 1 shows a summary of the number of inputs and outputs for each model and tutor.

5. SIMULATION STUDY

5.1 Tuning and Training Models

We conducted a simulation study with a factorial design, where every agent (Apprentice, PPO-Number, and PPO-Operator) was trained in each environments (fractions and multicolumn). The hyperparameters used by PPO greatly affect its performance [10] and they must be tuned independently for each model and task. We used Optuna, an open-source hyperparameter optimization framework to automate hyperparameter search [2]. Using Optuna, we ran approximately 100 iterations of hyperparameter tuning for each PPO model and task pair. Tuning one model for one task took approximately 38 hours. The Apprentice model does not have any hyperparameters that need to be tuned.

We trained each model in each environment using the best hyperparameters. We trained Apprentice on 500 fractions problems and 5000 multicolumn problems. These amounts provided enough practice to reach mastery while minimizing unnecessary computation. We trained each PPO model for 1 million steps, which translates into a varying number of problems depending on the amount of incorrect steps. To analyze the simulation logs, we assigned knowledge component labels to each field for each problem type (e.g., answer one's place for multicolumn), computed the first-attempt correctness on each knowledge components for each problem, and plotted this correctness on a log scale with values smoothed using binomial Gaussian additive smoothing (to account for the 0/1 nature of the correctness values).

5.2 Results

See Appendix A for the results of hyperparameter tuning. During tuning, we were unable to get PPO-Number to converge to a correct model in the fractions tutor. We hypothesized this was due to the large number of actions for this model/task. To test this hypothesis, we trained PPO-Number on the easy fractions tutor, which has substantially less actions. PPO-Number converged to correct behavior on this tutor, supporting our hypothesis.

Figures 3 and 4 shows the learning curves for the different models in the two tutor environments. We find that Apprentice converges to mastery after 10 opportunities for each knowledge component in the fractions tutor and 125 in the multicolumn tutor. In contrast, PPO-Number requires over 10,000 opportunities to reach mastery on the *easy* fractions tutor and over 10,000 practice opportunities to reach mastery in the multicolumn tutor. PPO-Operator requires less opportunities (3,000) within the fractions tutor , but never quite reaches mastery within the multicolumn tutor, even after 10,000 opportunities. Even though both PPO-Number and PPO-Operator receive the amount of training steps (1 million), PPO-Operator makes more mistakes per problem and receives less problems as a result.

5.3 Discussion

At least one PPO model was able to achieve mastery in each tutor. PPO-Operator achieved mastery in the fractions tutor and PPO-Number achieved mastery in the multicolumn tutor. This suggests that PPO can generalize from game and robotics tasks to tutor tasks. However, the finding that no single representation is best suggests that the representations must be customized for each task.

PPO-Number was unable to master the standard fractions tutor. We suspect this is due to the single output channel

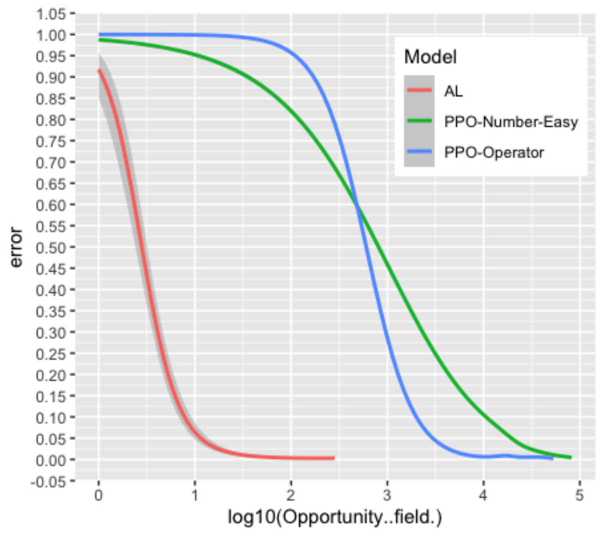


Figure 3: Fraction arithmetic learning curves.

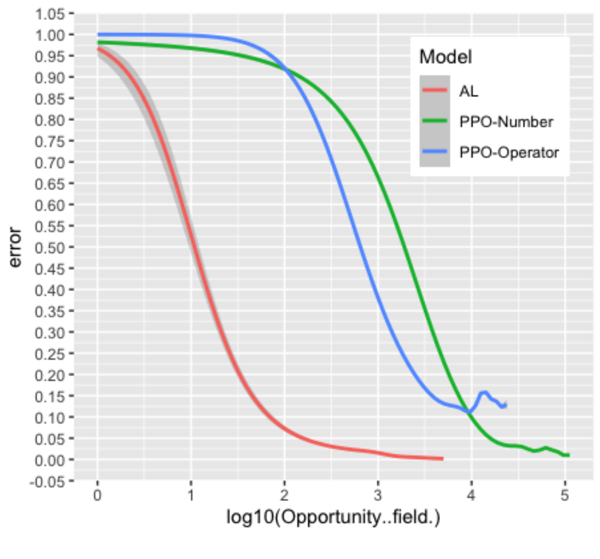


Figure 4: Multicolumn addition learning curves.

with 450 actions. Based on our experience, model performance degrades when the number of actions on one of the multidiscrete outputs gets large. Future work should explore replacing the 450 action output with three outputs: a hundreds digit output (0-4) a tens digit output (0-9) and a ones digit output (0-9). We also found that PPO-Operator was unable to achieve mastery in the multicolumn tutor. It may achieve mastery with more training (e.g., 1.5 million training steps rather 1 million). Assuming this is true, then PPO-Operator, which uses the same relational and operator knowledge as Apprentice seems to be more generally applicable than PPO-Number. Apprentice achieves mastery in both tasks with substantially less practice (thousands of times less), suggesting Apprentice has more efficient learning. Apprentice models have been shown to have similar learning curves to human students [16] for the fractions tutor. This implies that PPO models require substantially more training than human learners.

One limitation of the current study is that PPO may be more efficient when trained with multiple simultaneous environments (e.g., 8 tutors in parallel). Parallel training provides more diversity and improves learning. We tested this idea by training PPO models on 8 parallel environments for both the fractions and multicolumn tutors. We found that parallel PPO required an equivalent amount of practice to achieve mastery as non-parallel PPO; however, parallel PPO took less total wall time to train (e.g., 12 instead of 48 hours). Future work should explore the benefits and trade-offs of parallel training. Also, PPO is an on-policy RL approach, as opposed to an off-policy approach like Deep Q-Networks (DQN) [20]. As such, PPO only trains on the data that is immediately sampled from the environment; it discards old training data because it will cause the model to diverge. In contrast, DQN saves all experiences and continues to train on them over the course of learning. We would have liked to compare PPO to DQN, but DQN does not support multi-discrete action outputs, so could not be evaluated using the current Number and Operator representations. Future work should explore modifications of DQN (or other off-policy models) to see how they perform on these tasks.

6. RELATED WORK

There has been substantial work exploring the use of RL to optimally sequencing students' practice [9]. Unfortunately, this approach requires a large amount of data. One solution is to train models using simulated student data. However, simulated student models are often simplistic and not representative of real student behavior [8]. As a result, sequencing models built from synthetic data typically perform poorly with human students. The RL models we propose might serve as better simulated student models. Adopting a rational analysis perspective [3, 11], we hypothesize that agents that face the same task and processing constraints as humans will have similar behavior; i.e., sequencing models that are best for agents should be best for humans. However, future work is needed to investigate this hypothesis.

A similar parallel hypothesis is that when the task and processing constraints between RL and humans differ, the their behavior is likely to differ. To investigate this idea, Stamper et al. [26] explore differences in human vs. RL expertise for two games: Connect Four and Space Invaders. We view our work as complementary to this research, and future work should compare the behavior of expert models learned in this work to the behavior of human experts.

Simulated students have been used for a wide range of applications including theory testing [16], expert model authoring [17, 30], and teachable agents [18]. However, we are unaware of previously developed simulated students that make use of RL. Some of this prior work aims to model human learning and behavior. In contrast, this work makes little effort to model humans. We view this as a shortcoming of our current study, due to its preliminary nature. Future work should explore how RL approaches, such as those explored here, might be integrated within human-like simulated student models, such as Apprentice.

7. FUTURE WORK

TutorGym and our initial PPO models lay the foundation for a number of novel research directions. One promising directions we hope to explore concerns the use of RL to discover buggy student knowledge. During learning, RL agents make many mistakes. We should explore how these mistakes relate to the kinds of mistakes that humans make. VanLehn [27] investigated the "mind bugs" that human students exhibit in multicolumn arithmetic. Future work should explore how RL bugs compare to human bugs and if RL can support the discovery of bug knowledge for tutor tasks.

8. CONCLUSIONS

We explore the application of the PPO—an emerging RL approach—to educationally relevant tasks. While RL has been successfully applied to learn expert models across many tasks and domains, it has not yet been applied in the context of educationally relevant tasks. To support this exploration, we created TutorGym, a toolkit for interfacing RL models with educational training environments. We created two tutor-based environments within TutorGym: a fraction arithmetic tutor and a multicolumn addition tutor.

We created two PPO models that differ in their state and action representations (PPO-Number and PPO-Operator). For comparison purposes, we created a simulated student model using the Apprentice Architecture that has a similar state and action representation to the PPO-Operator model, but uses different (non-RL) learning mechanisms that are specifically designed to model human learning. We conducted a factorial study that varied the model and task. We found that at least one PPO model is able to achieve mastery within each tutor, suggesting that PPO is applicable to educationally relevant tasks. Despite this success, we found that both PPO models require substantially more training to reach mastery than Apprentice. This suggests that educationally relevant tasks present an interesting use case for the study and advancement of RL research. We also found an interaction between the type of PPO model and the task (PPO-Operator is best for fractions, but PPO-Number is best for multicolumn). This suggests that PPO's representation affects its performance and must be customized specifically for each task.

This work lays the foundation for future research to study and develop RL approaches for educationally relevant tasks. Our hope is that TutorGym and our initial models enable new research into how RL can support human education.

9. REFERENCES

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2004.

[2] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2019.

[3] J. R. Anderson. *The adaptive character of thought*. Psychology Press, 1990.

[4] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[6] BulletPhysics. Pybullets real-time physics simulation. `https://pybullet.org/wordpress/`, March 2021.

[7] DARPAtv. Alphadogfight trials final event. `https://www.youtube.com/watch?v=NzdhIA2S35w`, August 2020.

[8] S. Doroudi, V. Aleven, and E. Brunskill. Robust evaluation matrix: Towards a more principled offline exploration of instructional policies. In *Proceedings of the ACM Conference on Learning@Scale*, 2017.

[9] S. Doroudi, V. Aleven, and E. Brunskill. Where's the reward? *International Journal of Artificial Intelligence in Education*, 29(4):568–620, 2019.

[10] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Proceedings of the Conference on Artificial Intelligence*, volume 32, 2018.

[11] A. Howes, R. L. Lewis, and A. Vera. Rational adaptation under task and processing constraints: Implications for testing theories of cognition and action. *Psychological review*, 116(4):717, 2009.

[12] K. R. Koedinger, R. S. Baker, K. Cunningham, A. Skogsholm, B. Leber, and J. Stamper. A data repository for the edm community: The pslc datashop. *Handbook of educational data mining*, 43:43–56, 2010.

[13] N. Li, N. Matsuda, W. W. Cohen, and K. R. Koedinger. Integrating representation learning and skill learning in a human-like intelligent agent. *Artificial Intelligence*, 219:67–91, 2015.

[14] N. Li, E. Stampfer, W. Cohen, and K. Koedinger. General and efficient cognitive model discovery using a simulated student. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, 2013.

[15] C. MacLellan, K. Stowers, and L. Brady. Optimizing human performance using individualized computational models of learning. In *Proceedings of Advances in Cognitive Systems*, 2020.

[16] C. J. Maclellan, E. Harpstead, R. Patel, and K. R. Koedinger. The apprentice learner architecture: Closing the loop between learning theory and educational data. In *Proceedings of International Conference on Educational Data Mining*, 2016.

[17] C. J. MacLellan and K. R. Koedinger. Domain-general tutor authoring with apprentice learner models. *International Journal of Artificial Intelligence in Education*, pages 1–42, 2020.

[18] N. Matsuda, W. Weng, and N. Wall. The effect of metacognitive scaffolding for learning by teaching a teachable agent. *International Journal of Artificial Intelligence in Education*, pages 1–37, 2020.

[19] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 1928–1937. PMLR, 2016.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[21] R. Patel, R. Liu, and K. R. Koedinger. When to block versus interleave practice? evidence against teaching fraction addition before fraction multiplication. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, 2016.

[22] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

[23] Robotii, Inc. Mujoco advanced physics simulation. `http://www.mujoco.org/`, March 2021.

[24] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the International Conference on Machine Learning*, pages 1889–1897. PMLR, 2015.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[26] J. Stamper and S. Moore. Exploring teachable humans and teachable agents: Human strategies versus agent policies and the basis of expertise. In *Proceedings of the International Conference on Artificial Intelligence in Education*, pages 269–274. Springer, 2019.

[27] K. VanLehn. *Mind bugs: The origins of procedural misconceptions*. MIT press, 1990.

[28] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[29] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[30] D. Weitekamp, E. Harpstead, and K. R. Koedinger. An interaction design for machine teaching to develop ai tutors. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2020.

[31] Q. Zhang and C. MacLellan. Going online: A simulated student approach for evaluating knowledge tracing in the context of mastery learning. In *Proceedings of International Conference on Educational Data Mining*, 2021.

	PPO-Number Fractions-Easy	PPO-Operator Fractions	PPO-Number Multicolumn	PPO-Operator Multicolumn
n_steps	1024	256	32	128
batch_size	512	32	32	64
gamma	0.0	0.0	0.0	0.0
learning_rate	4.75e-3	4.56e-5	1.43e-3	7.13e-4
lr_schedule	constant	constant	linear	constant
entropy_coef	6.27e-3	3.28e-2	4.21e-2	2.91e-3
clip_range	0.2	0.1	0.2	0.4
n_epochs	1	10	5	1
gae_lambda	0.98	0.99	0.92	1.0
max_grad_norm	0.8	0.5	0.7	0.3
vf_coef	0.915	0.240	0.401	0.568
net_arch	small	tiny	medium	small
shared_arch	False	True	False	True
activation_fn	tanh	tanh	relu	tanh

Table 2: PPO hyperparameters identified using hyperparameter optimization.

APPENDIX

A. HYPERPARAMETER TUNING

For each tuning trial, Optuna selects hyperparameter from a prior sampling distribution, trains the model using these values, and measures the resulting performance. Within a trial, the PPO model is trained for 350,000 steps. The final model performance is used to update the hyperparameter sampling distribution, so subsequent iterations sample more promising hyperparameters. Optuna also implements a sample pruner, which detects PPO trials that are under performing (e.g., if PPO performance gets worse with training rather than better) and prunes these samples early.

Table 2 shows the hyperparameters that were identified by Optuna for each PPO model and domain. The hyperparameter values are not particularly interpretable, but we report them here so other researchers can replicate our results. It is worth noting that we manually fixed the gamma value at 0.0, since tutoring systems provide immediate reward and future rewards do not need to be factored into decision making. Additionally, the tiny net architecture used a neural network with two layers and 32 nodes per layer, the small network used 64 nodes per layer and the medium network used 128 nodes. If the architecture was shared, then the second layer of the network was shared by both the value and the policy head of the network. However, if they were not shared then there were separate second layers for the value and policy heads. Finally, a constant lr_schedule means that the learning rate is held constant over the course of training, whereas a linear schedule means that learning rate is decreased linearly towards 0 over the course of training.