Artificial Intelligence 219 (2015) 67–91

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Integrating representation learning and skill learning in a human-like intelligent agent

Nan Li *, Noboru Matsuda, William W. Cohen, Kenneth R. Koedinger

5000 Forbes Ave, Pittsburgh, PA 15232, USA

ARTICLE INFO

Article history:
Received 18 April 2012
Received in revised form 2 July 2014
Accepted 5 November 2014
Available online 4 December 2014

Keywords:
Agent learning
Representation learning
Student modeling

ABSTRACT

Building an intelligent agent that simulates human learning of math and science could potentially benefit both cognitive science, by contributing to the understanding of human learning, and artificial intelligence, by advancing the goal of creating human-level intelligence. However, constructing such a learning agent currently requires manual encoding of prior domain knowledge; in addition to being a poor model of human acquisition of prior knowledge, manual knowledge-encoding is both time-consuming and error-prone. Previous research has shown that one of the key factors that differentiates experts and novices is their different representations of knowledge. Experts view the world in terms of deep functional features, while novices view it in terms of shallow perceptual features. Moreover, since the performance of learning algorithms is sensitive to representation, the deep features are also important in achieving effective machine learning. In this paper, we present an efficient algorithm that acquires representation knowledge in the form of "deep features", and demonstrate its effectiveness in the domain of algebra as well as synthetic domains. We integrate this algorithm into a machine-learning agent, SimStudent, which learns procedural knowledge by observing a tutor solve sample problems, and by getting feedback while actively solving problems on its own. We show that learning "deep features" reduces the requirements for knowledge engineering. Moreover, we propose an approach that automatically discovers student models using the extended SimStudent. By fitting the discovered model to real student learning curve data, we show that it is a better student model than human-generated models, and demonstrate how the discovered model may be used to improve a tutoring system's instructional strategy.

1. Introduction

One of the fundamental goals of artificial intelligence is to understand and develop intelligent agents that simulate human-like intelligence. A considerable amount of effort [1–3] has been put toward this challenging task. Further, education in the 21st century will be increasingly about helping students not just learn content but to become better learners. Thus, we have a second goal of improving our understanding of how humans acquire knowledge and how students vary in their abilities to learn.

* Corresponding author.
E-mail addresses: nli1@cs.cmu.edu (N. Li), Noboru.Matsuda@cs.cmu.edu (N. Matsuda), wcohen@cs.cmu.edu (W.W. Cohen), koedinger@cmu.edu (K.R. Koedinger).

http://dx.doi.org/10.1016/j.artint.2014.11.002
0004-3702/

To contribute to both goals, there have been recent efforts [4–7] in developing intelligent agents that model human learning of math, science, or a second language. Although such agents produce intelligent behavior with less human knowledge engineering than before, there remains a non-trivial element of knowledge engineering in the encoding of the prior domain knowledge given to the simulated student agent at the start of the learning process. For example, to build an algebra learning agent, the agent developer needs to provide prior knowledge by coding functions that describe, for instance, how to extract a coefficient or how to add two algebraic terms. Such manual encoding of prior knowledge can be time-consuming and the constructed prior knowledge may not naturally correspond with a human student's prior knowledge.

Since real students entering a course do not usually have substantial domain-specific or domain-relevant prior knowledge, it is not realistic in a model of human learning to assume this knowledge is given rather than learned. For example, for students learning about algebra, we cannot assume that they all know beforehand what a coefficient is, or what the difference between a variable term and a constant term is. An intelligent system that models automatic knowledge acquisition with a small amount of prior knowledge could be helpful both in reducing the effort in knowledge engineering intelligent systems and in advancing the cognitive science of human learning.

Previous work in cognitive science [8,9] showed that one of the key factors that differentiates experts and novices in a field is their different prior knowledge of world state representation. Experts view the world in terms of deep functional features (e.g., coefficient and constant in algebra), while novices only view in terms of shallow perceptual features (e.g., integer in an expression). Deep features are often domain-specific, whereas shallow perceptual features are domain-independent. Having the correct representation of the deep features aids the process of solving the domain task. For example, in algebra, students need to learn to encode equation input into "terms" and "coefficients". A shallow feature encoding of a coefficient (e.g., of the "*5*" in "*5x*") is as a number before a letter. A deep feature encoding requires the learner to develop knowledge, which may include implicit perceptual processing capabilities, to recognize coefficients more generally such as the "*-5*" in "*-5x*", the "*a*" in "*ax*", the "*3*" in "*3(x+2)*", the "*-1*" in "*-x*". In general, experts develop deep feature knowledge that allows them to see the world in the way novices do not – expert readers see "run" as a word whereas novices see letters or just lines, experts in physics see force contact points whereas novices see blocks and inclined planes, chess experts see configurations of pieces like a knight fork whereas novices see pieces. Such deep feature perception knowledge is learned for specific domains, perhaps as much by implicit experience as by explicit instruction. In algebra, students learn to see terms and coefficients in equations building upon more general prior knowledge of numbers. That prior knowledge may be the basis for initial shallow feature encoding as in the example above. In general, we consider deep features as part of representation knowledge. Representation knowledge organizes low-level perceptual input into a structured form that assists the agent to understand and solve problems in a particular domain. Even if the same perceptual input was given, for different problem solving tasks in different domains, the ideal representation of the world can be different for different tasks. Deep features can be viewed as the key features that differentiate a well-structured representation from a poorly-structured representation.

Deep feature learning is a major component of human expertise acquisition, but has not received much attention in AI. Learning deep features changes the representation on which future learning is based and, by doing so, improves future learning. However, how these deep features are acquired is not clear. Therefore, we have recently developed a learning algorithm that acquires deep features automatically with only domain-independent knowledge (e.g., what is an integer) as input [10]. We evaluated the effectiveness of the algorithm in learning deep features, but not its impact on future skill learner.

In order to evaluate how the deep feature learner could affect future learning of an intelligent agent, in this paper, we integrated this deep feature learning algorithm into *SimStudent* [11], an agent that learns problem-solving skills by example and by feedback on performance. The original SimStudent relies on a hand-engineered representation that encodes an expert representation given as prior knowledge. This limits its ability to model novice students. The extended SimStudent first acquires the representation of the problems using the deep feature learner. Then, it makes use of the learned representation to acquire skill knowledge in later tasks. Integrating the deep feature learner into the original SimStudent both reduces the amount of engineering effort and builds a better model of student learning.

We show that the extended SimStudent with better representation learning performs much better than the original SimStudent when neither of them are given domain-specific knowledge. Furthermore, we also show that even compared to the original SimStudent with the domain-specific knowledge, the extended SimStudent is able to learn nearly as well without being given domain-specific knowledge. For the sake of simplicity, we only report experiment results in the algebra domain in this paper, but similar results are also observed in other domains [12]. In addition, we use the extended SimStudent to automatically discover models of real students, and show that the discovered models are better student models than human-generated models [13]. Although not reported here, we further use the extended SimStudent to better understand how problem orders affect learning effectiveness by inspecting SimStudent's learning processes and learning outcomes, which are not easily obtainable from human subjects [14].

To summarize, the main contributions of this paper are two-fold. By integrating representation learning into skill learning, 1) we reduce the amount of knowledge engineering effort required in constructing an intelligent agent; 2) we get a better model of human behavior.

In the following sections, we start with a brief review of SimStudent. We then present the deep feature learning algorithm together with its evaluation results. Next, we describe how to integrate the deep feature learner into SimStudent, and illustrate the algorithm with an example in algebra. After that, we present experimental results for both the original Sim-

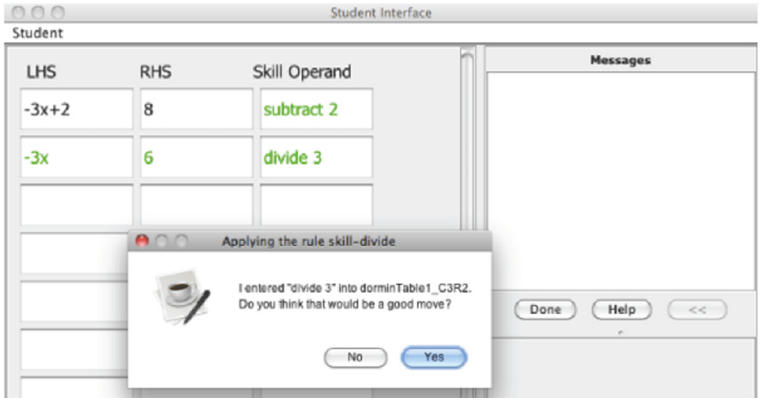

Fig. 1. The interface where SimStudent is being tutored in an equation solving domain. The given problem was $-3x + 2 = 8$, and SimStudent has been tutored to subtract both sides by 2. After entering *divide* 3, SimStudent asks the author user/tutor whether this step is correct. If SimStudent has not learned an applicable production rule, it asks the author to demonstrate a good next step and then learns a production rule to reproduce steps like this.

Student and the extended SimStudent trained with problem sets used by real students in learning algebra, and show that the extended SimStudent is able to achieve performance comparable to the original SimStudent without requiring domain-specific knowledge as input. We present a method for using SimStudent to automatically discover student models, and show that the student model discovered by the extended SimStudent is better than the human-generated models. Finally, we discuss related work limitations, and possible future extensions.

2. A brief review of SimStudent

SimStudent is an intelligent agent that inductively learns skills to solve problems from demonstrated solutions and from problem solving experience. It is an extension of programming by demonstration [16] using inductive logic programming [17] as one of its underlying learning techniques.

SimStudent, as a theory of learning in academic domains, makes the following assumption: 1. Problem-solving skills can be modeled as condition-response patterns. Much of academic learning (expertise development) involves skill acquisition of well represented production rules (or schemas) that indicate a mental or external response (how-part) may be made when the information needed is found (where-part) and pre-conditions are met (when-part) in hierarchical representations of the perceived world. 2. Multiple learning mechanisms are used in skill learning. Effective learning of skills requires three kinds of generalizations: where, when, how. 3. Skill learning has a perceptual grounding bias. How the input world is represented (e.g., in a perceptual abstraction hierarchy) is an essential inductive bias in skill learning. This learning bias produces real differences between learning success or failure, from an engineering perspective, and produces real differences in the quality of predictions about human academic learning, from a scientific perspective. 4. Representation learning is modeled as an unsupervised induction of perceptual structures. An abstract perceptual representation can be acquired in an unsupervised way using spatial and temporal proximity information to create chunks. Later in this paper, we present our approach in implementing the fourth tenet by using grammar induction.

SimStudent is similar to the ACT-R [18] and Soar [19] theories of learning. Like those theories, SimStudent represents the product of learning as a production system. Unlike those theories, it puts more emphasis on knowledge-level learning (cf., [20]) achieved through induction from positive and negative examples. SimStudent is more than a problem-solving agent, but is a theory of learning in academic domains, with an emphasis on learning skills and their representational basis. As with Soar and ACT-R, the theory is made concrete in a software architecture and has been applied to model learning (as well as problem solving or skilled performance) across a variety of domains, including math, science, and second language [12,21]. Compared to SOAR or ACT-R, not as many models have been created with SimStudent; however, the models that have been created cover new territory, demonstrating knowledge-level learning not achieved in Soar or ACT-R., specifically without domain-specific prior knowledge of math, science, and language skills.

Figs. 1 and 2 are screenshots of SimStudent learning to solve algebra equations. Fig. 1 is an interface used to teach SimStudent equation solving, and Fig. 2 shows how SimStudent keeps track of the demonstrated steps and acquires skill knowledge based on them. In this paper, we will use equation solving as an illustrative domain to explain the learning mechanisms. However, the learning algorithms are domain general. In fact, SimStudent has been used and tested across various domains, including multi-column addition, fraction addition, stoichiometry, and so on. In the rest of this section, we will briefly review the learning mechanism of SimStudent. For full details, please refer to [6].

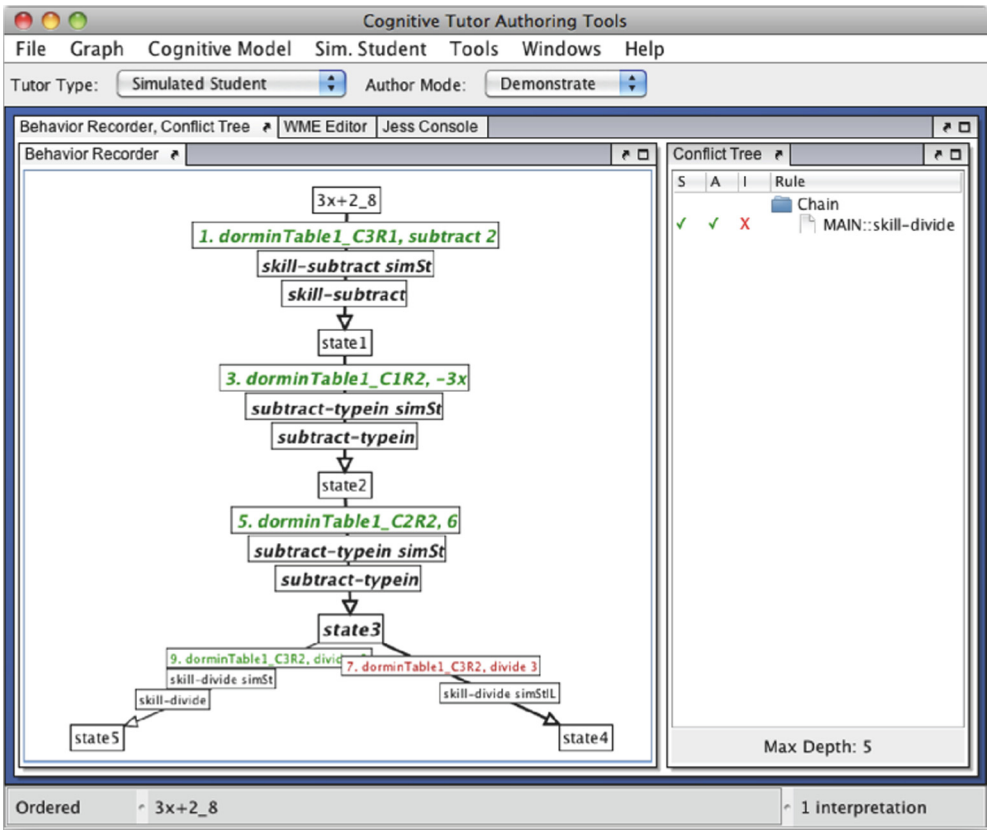

Fig. 2. CTAT's [15] behavior recorder is used to show how SimStudent traces each demonstrated step. Each entry in a GUI element is traced. In this example, the state changes (steps) are the four entries in the table cells shown in Fig. 1. These are either traced by an existing production rule or used to learn a new production rule. The conflict tree panel in the figure shows that SimStudent applied the skill "divide" incorrectly (i.e., dividing both sides by 3 instead of -3).

2.1. Learning task

SimStudent is given a set of (ideally simple) *feature predicates* and a set of (ideally simple) *operator functions* as prior knowledge before learning. Each feature predicate is a boolean function that describes relations among objects in the domain. For example, *(has-coefficient -3x)* means *-3x* has a coefficient.

Operator functions specify basic functions (e.g., add two numbers, get the coefficient) that SimStudent can apply to aspects of the problem representation. Operator functions are divided into two groups, domain-independent operator functions and domain-specific operator functions. Domain-independent operator functions can be used across multiple domains, and tend to be simpler (like standard operations on a programming language). Examples of such operator functions include adding two numbers, *(add 1 2)* or copying a string, *(copy -3x)*. These operator functions are not only useful in solving equations, but can also be used in other domains such as multi-column addition and fraction addition. Because these domain general functions are involved in domains that are acquired before algebra, we can assume that real students know them prior to algebra instruction. Because these domain general functions can be used in multiple domains, there is a potential engineering benefit in reducing or eliminating a need to write new operator functions when applying SimStudent to a new domain. Domain-specific operator functions, on the other hand, are more complicated functions, such as getting the coefficient of a term, *(coefficient -3x)*, or adding two terms, *(add-term 5x-8 2x+3)*. Performing such operator functions implies some domain expertise that real students are less likely to have.

Domain-specific operator functions tend to require more knowledge engineering or programming effort than domain-independent operator functions. For example, compare the "add" domain-independent operator function with the "add-term" domain-specific operator function. Adding two numbers is one step among the many steps in adding two terms together (i.e., parsing the input terms into sub-terms, applying an addition strategy for each term format, and concatenating all of the sub-terms together).

Note that operator functions are different from *operators* in traditional planning systems, operator functions have no explicit encoding of preconditions and may not produce correct results when applied in context. For example, dividing both sides by *3* (i.e., *(divide 3)*) is an incorrect step for problem *3x+2=5*. Thus, SimStudent is different from traditional planning

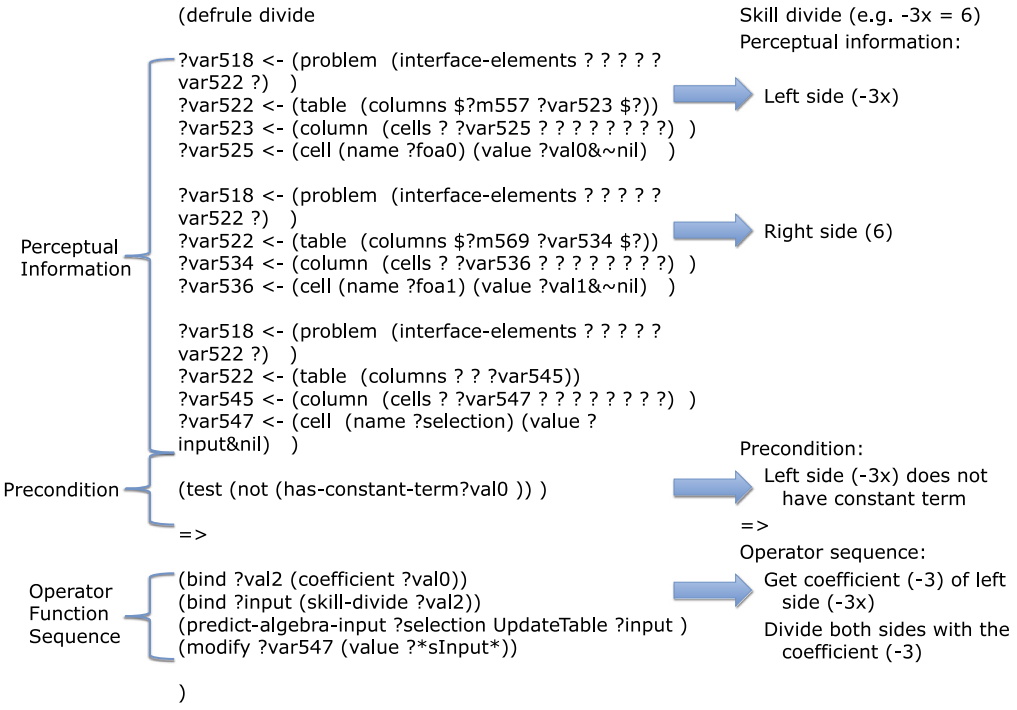

Fig. 3. A production rule for divide.

algorithms that engage on speed up learning. SimStudent engages in knowledge level learning [20], and inductively acquires complex reasoning rules. These rules are represented as *production rules*, which we will explain later.

During the learning process, given the current state of the problem (e.g., *-3x=6*), SimStudent first tries to find an appropriate production rule that proposes a plan for the next step (e.g., *(coefficient -3x ?coef) (divide ?coef)*). If it finds one and receives positive feedback, it continues to the next step. If the proposed next step is incorrect, negative feedback is given, and if SimStudent has no other alternatives, a correct next step demonstration is provided. SimStudent will attempt to modify or learn production rules accordingly. Although other feedback mechanisms are also possible, in our case, the feedback is given by an existing automatic cognitive tutor, CTAT [15], which has been used to teach real students. For each demonstrated step, the tutor specifies 1) *perceptual information* (e.g., *-3x* and *6* for *-3x=6*) from a graphical user interface (GUI) showing where to find information to perform the next step, 2) *a skill label* (e.g., divide) corresponding to the type of skill applied, 3) *a next step* (e.g., *(divide -3)* for problem *-3x=6*). This simulates the limited information available to real students. Taken together, the three pieces of information form an *example action record* indexed by the skill label, $R = \langle label, \langle percepts, step\rangle\rangle$. In the algebra example, an example action record is $R = \langle divide, \langle(-3x, 6), (divide\ -3)\rangle\rangle$. For each incorrect next step proposed by SimStudent, an example action record is also generated as a negative example. During learning, SimStudent typically acquires one production rule for each skill label, l, based on the set of associated (both positive and negative) example action records gathered up to the current step, $\mathcal{R}_l = (R_1, R_2, ..., R_n)$ (where $R_i.label = l$, $i = 1, 2, ..., n$).

In summary, since we would like to model how real students are tutored, the learning task presented to SimStudent is challenging. First, the total number of world states is large. In equation solving, for instance, there are infinite variety of algebraic expressions that can be entered and there are many possible alternative solution strategies. Second, the operator functions given as prior knowledge do not encode any preconditions (neither for applicability nor for search control) or postconditions. Last, the semantics of a demonstrated step is only partially observable. It usually takes more than one operator function to move from one observed state to the next observed state. Correct intermediate outputs of operator functions are unobservable to SimStudent. Taken together, the learning task SimStudent is facing is learning skill knowledge within infinite world states given incomplete operator function descriptions and partially observable states.

2.2. Production rules

The output of the learning agent is represented as production rules [1,2]. The left side of Fig. 3 shows an example of a production rule learned by SimStudent with a simple English description shown on the right. A production rule indicates "where" to look for information in the interface (perceptual information), "how" to change the problem state (an operator function sequence), and "when" (precondition) to apply a rule (a set of features indicting the circumstances under which performing the how-part will be useful). For example, the rule to "divide both sides of *-3x=6* by *-3*" shown in Fig. 3 can be read as "given a left-hand side (i.e., *-3x*) and a right-hand side (i.e., *6*) of an equation, when the left-hand side does not

have a constant term, then get the coefficient of the term on the left-hand side (i.e., -3) and write "divide" followed by the coefficient (i.e., *(divide -3)*."

The perceptual information part represents paths to identify useful information from the GUI. During execution, SimStudent matches the perceptual hierarchy against the acquired production rules. With a properly-structured perceptual hierarchy, SimStudent would be able to quickly focus its attention on essential information in solving the problem. As we will discuss later, with the extension proposed in this paper, SimStudent is also able to update its perceptual hierarchy based on inputs from the environment, which aids the learning process. The precondition (just before "⇒" in Fig. 3) includes a set of feature tests representing desired conditions in which to apply the production rule. The last part (after "⇒" in Fig. 3) is the operator function sequence which computes what to output in the GUI.

2.3. Learning mechanisms

With all the challenges presented, we have developed three learning mechanisms in SimStudent to acquire the three parts of the production rules [6]. The first component is a perceptual learner that learns the where-part of the production rule by finding paths to identify useful information in the GUI. The elements in the interface are typically organized in a tree structure. The left side of Fig. 10 shows an example perceptual hierarchy in the algebra domain. For example, the table node has columns as children, and each column has multiple cells as children. The percepts specified in the above production rule are cells associated with the sides of the algebra equation, which are *Cell 11* and *Cell 21* in this case. Hence, the perceptual learner's task is to find the right paths in the tree to reach the specified cell nodes. There are two ways to reach a percept node in the interface: 1) by the exact path to its exact position in the tree, or 2) by a generalized path to a set of GUI elements that may have a specific relationship with the GUI element where the next step is entered (e.g., cells above next step). A generalized path has one or more levels in the tree that are bound to more than one node. For example, a cell in the second column and the third row, *Cell 23*, can be generalized to any cell in the second column, *Cell 2?*, or any cell in the table, *Cell ??*. In the example shown in Fig. 3, the production rule has an over-specific where-part that produces a next step only when the sides of the current step are in row two.

SimStudent assumes that example action records for the same skill have a fixed number of percepts. Therefore, for each positive example action record associated with skill label l, $R_i \in \mathcal{R}_l$, the percept field, $R_i.percepts$, is an m-dimensional vector, i.e., $R_i.percepts = (percepts_{i1}, percepts_{i2} ... percepts_{im})$, where $percepts_{ij}$ stands for the jth percept in the ith example action record. Each percept in the vector, $percepts_{ij}$, is a GUI element.[1] The set of percepts from all positive examples form an $n \times m$ matrix, $\mathcal{P} = (R_1.percepts, R_2.percepts, ... R_n.percepts)^T$, where each row $\mathcal{P}_{i?}$ is a percepts field in one example action record, and each column $\mathcal{P}_{?j}$ is composed of percepts of the same index in all of the example action records. For each position j, the set of paths, where each path can reach all percepts in $\mathcal{P}_{?j}$, defines a version space $\mathcal{V}_j$ [22] (i.e., the subset of all hypotheses that are consistent with the observed training examples). The learner searches for the least general path in the version space $\mathcal{V}_j$. This process is done by a brute-force depth-first search. For example, if only given the example *-3x=6* in row two, the production rule learned as shown in Fig. 3 has an over-specific where-part. If given more examples in other rows (e.g., *4x=12* in row three), the where-part will be generalized to any row in the table.

The second part of the learning mechanism is a feature test learner that learns the when-part of the production rule by acquiring the precondition of the production rule using the given feature predicates. The acquired preconditions should contain information about both applicability (e.g., getting a coefficient is not applicable to the term *3x+5*) and search control (e.g., it is not preferred to add *5* to both sides for problem *-3x=6*). The feature test learner utilizes FOIL [23], an inductive logic programming system that learns Horn clauses from both positive and negative examples expressed as relations. FOIL is used to acquire a set of feature tests that describe the desired situation in which to fire the production rule. For each rule, the feature test learner creates a new predicate that corresponds to the precondition of the rule, and sets it as the target relation for FOIL to learn. The arguments of the new predicate are associated with the percepts. Each training action record serves as either a positive or a negative example for FOIL. The task of FOIL is to find a set of feature tests that best separate positive examples from negative examples. For example, $(precondition - divide\ ?percept_1\ ?percept_2)$ is the precondition predicate associated with the production rule named "divide". *(precondition-divide -3x 6)* is a positive example for it. The feature test learner computes the truthfulness of all predicates bound with all possible permutations of percept values, and sends it as input to FOIL. Given these inputs, FOIL will acquire a set of clauses formed by feature predicates describing the precondition predicate.

The last component is an operator function sequence learner that acquires the how-part of the production rule. For each positive example action record, R_i, the learner takes the percepts, $R_i.percepts$, as the input, and sets the step, $R_i.step$, as the output. We say an operator function sequence *explains* a percepts-step pair, $\langle R_i.percepts, R_i.step \rangle$, if the system takes $R_i.percepts$ as the input and yields $step_i$ after applying the operator functions. For example, if SimStudent first receives a percepts-step pair, $\langle (2x, 2), (divide2) \rangle$, both the operator function sequence that directly divides both sides with the right-hand side (i.e., *(divide ?rhs)*), and the sequence that first gets the coefficient, and the divides both sides with the coefficient (i.e., *(coefficient ?lhs ?coef) (divide ?coef)*) are possible explanations for the given pair. Since we have multiple example action records for each skill, it is not sufficient to find one operator function sequence for each example action

[1] In the case of the equation solving domain, all percepts are associated with cells, but the learning algorithm is not limited to cells.

Table 1
Probabilistic context free grammar for coefficient in algebra.

Terminal symbols: $-, x$;
Non-terminal symbols: *Expression*, *SignedNumber*, *Variable*, *MinusSign*, *Number*;
Expression → 1.0, [*SignedNumber*]*Variable*
Variable → 1.0, *x*
SignedNumber → 0.5, *MinusSign Number*
SignedNumber → 0.5, *Number*
MinusSign → 1.0, −

record. Instead, the learner attempts to find a shortest operator function sequence that explains all of the $\langle percepts, step\rangle$ pairs using iterative-deepening depth-first search within some depth-limit. As in the above example, since *(divide ?rhs)* is shorter than (i.e., *(coefficient ?lhs ?coef) (divide ?coef)*), SimStudent will learn this operator function sequence as the how-part. Later, it meets another example, *-3x=6*, and receives another percepts-step pair, $\langle(-3x, 6), (divide - 3)\rangle$. The operator function sequence that divides both sides with the right-hand side is not a possible explanation any more. Hence, SimStudent modifies the how-part to be the longer operator function sequence *(coefficient ?lhs ?coef) (divide ?coef)*.

Last, although we said that SimStudent tries to learn one rule for each label, when a new training action record is added, SimStudent might fail to learn a single rule for all example action records. In that case, SimStudent learns a separate rule just for the last example action record. More specifically, SimStudent starts with a given set of skill labels associated with demonstrated steps. SimStudent tries to learn one rule for each label. It will fail when the perceptual information learner cannot find one path that covers all demonstrated steps, or the operator sequence learner cannot find one operator function sequence that explains all records. In that case, SimStudent learns a disjunctive rule just for the last record. This effectively splits the examples into two clusters. Later, for each new record, SimStudent tries to acquire a rule for each of the clusters with the new record, and stops whenever it successfully learns a rule with one of the clusters. If the new record cannot be added to any of the existing clusters, SimStudent creates another new cluster. With this approach, the number of clusters only increases as more records are seen. There is no merge operation among clusters.

3. Deep feature learning

Having reviewed SimStudent's production rule learning mechanisms, we move to a discussion of representation knowledge acquisition as deep feature learning. As mentioned above, deep feature learning is important both for human knowledge acquisition, and in achieving effective machine learning. We carefully examined the nature of deep feature learning in algebra equation solving, and discovered that it could be modeled as an unsupervised grammar induction problem given observational data (e.g., expressions in algebra). Expressions can be formulated as a context free grammar (CFG) and deep features are modeled as grammar non-terminal symbols in particular positions in a grammar rule. Table 1 illustrates a portion of a grammar for algebra expressions and the modeling of the deep feature "coefficient" as a non-terminal symbol in one of the grammar rules, as indicated by the square brackets (i.e., [*SignedNumber*]).

Viewing feature learning tasks as grammar induction provides a general explanation of how experts acquire perceptual chunks [9,24] and specific explanations for novice errors. In this account, some novice errors are the result of acquiring the wrong grammar for the task domain. Let us use the *-3x* example again. The correct grammar shown in Table 1 produces the correct parse tree shown on the left in Fig. 4. A novice, however, may acquire different grammar rules (e.g., because of plausible lack of experience with negative numbers) and these result in the incorrect parse tree shown on the right of Fig. 4. Instead of grouping - and *3* together, this grammar groups *3* and *x* first, and thus mistakenly considers *3* as the coefficient. In fact, a common strategic error students make in a problem like *-3x=12* is for the student to divide both sides by *3* rather than *-3* [13]. Based on observations like these, we built a deep feature learner by extending an existing probabilistic context free grammar (pCFG) learner [25] to support feature learning and transfer learning. Note that the deep feature learner is domain general. It currently supports domains where student input can be represented as a string of tokens, and can be modeled with a context-free grammar (e.g., algebra, chemistry, natural language processing).

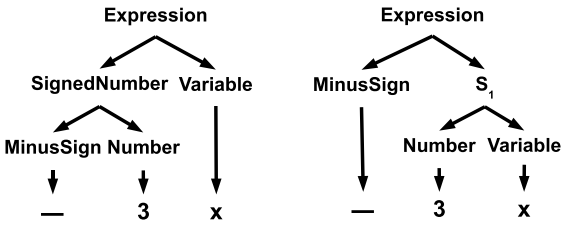

Fig. 4. Correct and incorrect parse trees for $-3x$.

Algorithm 1: *GSH* constructs an initial set of grammar rules, S, from observation sequences, O.

```
   Input: Observation Sequence Set O.
 1 S := terminal symbol grammar rules;
 2 while not-all-sequences-are-parsable(O, S) do
 3 |  if has-recursive-rule(O) then
 4 |  |  s := generate-recursive-rule(O);
 5 |  else
 6 |  |  s := generate-most-frequent-rule(O);
 7 |  end
 8 |  S := S + s;
 9 |  O := update-plan-set-with-rule(O, S);
10 end
11 S = initialize-probabilities(S);
12 return S
```

3.1. A brief review of the pCFG learner

Before introducing the deep feature acquisition algorithm, we first briefly review the pCFG learner [25] it is based on. The pCFG learner is a variant of the inside–outside algorithm [26] that acquires a probabilistic context-free grammar (pCFG) using an expectation-maximization (EM) algorithm [27]. The input to the pCFG learner is a set of observation sequences, $\mathcal{O}$. Each sequence is a string of tokens directly from user input. The output is a pCFG that can generate all input observation sequences with high probabilities. The system consists of two parts, a greedy structure hypothesizer (GSH), which creates non-terminal symbols and associated grammar rules, as needed, to cover all the training examples, and a Viterbi training step, which iteratively refines the probabilities of the grammar rules.

3.1.1. Greedy structure hypothesizer (GSH)

Pseudo code for the GSH algorithm is shown in Algorithm 1. GSH creates a xcontext-free grammar in a bottom-up fashion. It starts by initializing the rule set S to rules associated with terminal characters (e.g., *3* and *x* in *3x*) in the observation sequences, O. Next the algorithm (line 4) detects whether there are possible recursive structures embedded in the observation sequence by looking for repeated symbols. If so, the algorithm learns a recursive rule for them. If the algorithm fails to find recursive structures, it starts to search for the character pair that appears in the plans most frequently (line 6), and constructs a grammar rule for the character pair. To build a non-recursive rule, the algorithm will introduce a new symbol and set it as the head of the new rule. After getting the new rule, the system updates the current observation set O with this rule by replacing the character pairs in the observations with the head of the rule (line 9).

After learning all the grammar rules, the structure learning algorithm assigns initial probabilities to these rules. If there are k grammar rules with the same head symbol, then each of them are assigned the probability $\frac{1}{k}$. To break ties among grammar rules with the same head, GSH adds a small random number to each probability and normalizes the values again. This output of GSH is a redundant set of grammar rules, which is sent to the Viterbi training phase.

3.1.2. Refining schema probabilities: Viterbi training phase

The probabilities associated with the initial set of rules generated by the GSH phase are tuned by a Viterbi training algorithm. It considers the parse tree associated with each observation sequence as hidden variables. Each iteration involves two steps. Algorithm 2 shows the pseudo code of the Viterbi training phase.

In the first step, the algorithm computes the most probable parse tree for each observation example using the current rules. Any subtree of a most probable parse tree is also a most probable parse subtree. Therefore, for each observation sequence, the algorithm builds the most probable parse tree in a bottom-up fashion until reaching the start symbol. After getting the parse trees for all observation examples, the algorithm moves on to the second step. In this step, the algorithm updates the selection probabilities associated with the grammar rules. For a grammar rule with head a_i, the new probability of being chosen is simply the total number of times that schema appears in the Viterbi parse trees divided by the total number of times a_i appears in the parse trees.

In comparison with expectation-maximization [27], Viterbi training is considered as a "hard" EM learning algorithm, where only the most probable parse tree is viewed as the parse tree of the observation sequence. In contrast, in a typical EM learning procedure, all possible parse trees are used when updating grammar rule probabilities. Therefore, this learning procedure is a fast approximation of expectation-maximization, which approximates the posterior distribution of trees given parameters by the single MAP hypothesis.

After finishing the second step, the algorithm starts a new iteration until convergence. The output of the algorithm is a set of probabilistic grammar rules. As we will explain in Subsection 3.3.2, if trained on prior tasks, during the Viterbi training phase, the learning algorithm estimates rule frequency using a Dirichlet distribution.

Algorithm 2: *Viterbi-training* updates the probabilities associated with the set of given grammar rules, S, from observation sequences, O.

Input: Observation Sequence Set O, Grammar Rule Set S.
while *not-converged* **do**
  // Compute the most probable parse trees of the observation sequences.
  $T := \phi$;
  forall the $o_i \in O$ **do**
    T := T + get-the-most-probable-parsing(o_i, S);
  end
  // Update grammar rule probabilities according to the parse trees.
  forall the $s_i \in S$ **do**
    sel := number-of-times-selected(s_i, T);
    tol := number-of-times-selected($s_i.head$, T);
    $s_i.p := sel \mid tol$;
  end
end
return S

3.2. Feature learning

Having reviewed Li et al.'s [25] pCFG learning algorithm, before embedding the algorithm into SimStudent, let's first describe how it is extended to support deep feature learning. The input of the system is a set of pairs such as $\langle -3x, -3 \rangle$, where the first element is the input to a feature extraction mechanism (e.g., finding the coefficient of *-3x*), and the second is the extraction output (e.g., *-3* is the coefficient of *-3x*). The output is a pCFG with a non-terminal symbol in one of the rules set as the target feature (as shown by [*SignedNumber*] in Table 1). To produce this output, the deep feature learner uses the pCFG learner to produce a grammar, and then searches for non-terminal symbols that correspond to the example extraction output (e.g., the *-3* in *-3x*). The process is done in three steps.

The system first builds the parse trees for all of the observation sequences based on the acquired rules. For instance, in algebra, suppose we have acquired the pCFG shown in Table 1. The associated parse tree of *-3x* is shown at the left side of Fig. 4. Next, for each sequence, the learner traverses the parse tree to identify the non-terminal symbol associated with the target feature extraction output, and the rule to which the non-terminal symbol belongs. In the case of our example, the non-terminal symbol is *SignedNumber*, the associated feature extraction output is *-3*, and the rule is *Expression* → *1.0, SignedNumber Variable*. For some of the sequences, the feature extraction output may not be generated by a single non-terminal symbol, which happens when the acquired pCFG does not have the right structure. For example, the parse tree shown in the right side of Fig. 4 is an incorrect parse of *-3*, and there is no non-terminal symbol associated with *-3*. Last, the system records the frequency of each symbol rule pair, and picks the pair that matches the most training records as the learned feature. For instance, if most of the input records match with *SignedNumber* in *Expression* → *1.0, SignedNumber Variable*, this symbol-rule pair will be considered as the target feature pattern.

After learning the feature, when a new problem comes, the system will first build the parse tree of the new problem based on the acquired grammar. Then, the system recognizes the subsequence associated with the feature symbol from the parse tree, and returns it as the target feature extraction output (e.g., *-5* in *-5x*).

3.3. Transfer learning for deep feature learning

In order to achieve effective learning, we further extended the feature learner to support transfer learning within the same domain and across domains. Different grammars sometimes share grammar rules for some non-terminal symbols. For example, both the grammar of equation solving and the grammar of integer arithmetic problems should contain the sub-grammar of signed number. We extended the feature learning algorithm to transfer solutions to common sub-grammars from one task to another. Note that the tasks can be either from the same domain (e.g. learning what is an integer, and learning what is a coefficient), or from different domains (e.g. learning what is an integer, and learning what is a chemical formula). We consider two learning protocols: one in which the tutor provides hints to a shared grammar by highlighting subsequences that should be associated with a non-terminal symbol; and one in which the shared grammar is present, but no hints are provided. For transfer learning with sub-grammar hints, we applied what we will call a *feature focus mechanism* to the acquisition process. For transfer learning without sub-grammar hints, we extended the system to make use of grammar rule application frequencies from previous tasks to guide future learning, as explained below.

3.3.1. Explicitly labeled common sub-grammars

We first consider the situation where SimStudent's tutor provides a hint toward a shared sub-grammar (the deep feature). In the original learning algorithm, during the process of grammar induction, the learner acquires some grammar that generates the observation sequences, without differentiating potential feature subsequences (e.g. coefficients or constant terms) from other subsequences in the training examples. It is possible that two grammars can generate the same set of observation sequences, but only one grammar has the appropriate feature symbol embedded in it. We cannot be sure that the original learner will learn the right one.

However, it may be reasonable to assume that a tutor explicitly highlights example subsequences as targeted features as in a teacher giving examples of coefficient by indicating that *-3* is the coefficient of *-3x* and *-4* is the coefficient of *-4x*. With this assumption, the deep feature learner can focus on creating non-terminal symbols for such feature subsequences. We developed this *feature focus mechanism* as follows. First, we call one copy of the original learner to learn the sub-grammar for the deep feature. That is, we extract all the feature subsequences from training sequences, and then learn a sub-grammar for it. We then replace the feature subsequence with a special *semantic terminal symbol*, and invoke the original learner on this problem. Since this semantic terminal symbol is viewed as a terminal character in this phase of learning, it must be properly embedded in the observation sequence. Finally, the two grammars are combined, and the semantic terminal is relabeled as a *non-terminal symbol* and associated with the start symbol for the grammar for the feature.

3.3.2. Learning and transfer of common sub-grammars without hints

Aiding transfer learning by providing hints for common sub-grammars requires extra work for the tutor. A more powerful learning strategy should be able to transfer knowledge without adding more work for the tutor. Therefore, we considered a second learning protocol, where the shared grammar is present, but no hints to it are provided. An appropriate way of transferring previously acquired knowledge to later learning could improve the speed and accuracy of that later learning. The intuition here is that the perceptual chunks or grammar acquired with whole-number experience will aid grammar acquisition of negative numbers which, in turn, will aid algebra grammar acquisition. Our solution involves transferring the acquired grammar, including the application frequency of each grammar rule, from previous tasks to future tasks.

More specifically, during the acquisition of the grammar in previous tasks, the learner records the acquired grammar and the number of times each grammar rule appeared in a parse tree. Since previous research on algebra problem solving has pointed out that computational parsimony is an important factor in a quality cognitive model of human problem solving [28], when faced with a new task, the learning algorithm uses the existing grammar from previous tasks to build the smallest number of most probable parse trees for the new records. This process is done in a top-down fashion. For each sequence/subsequence, the algorithm first tries to see whether the given sequence/subsequence can be reduced to a single most probable parse tree. If it succeeds, the algorithm returns; if it fails, the algorithm separates the sequence/subsequence into two subsequences, and recursively calls itself. After building the least number of most probable parse trees for the training subsequences, the system switches to the original GSH and acquires new rules based on the partially parsed sequences.

For example, if the grammar learner acquired what is a signed number (e.g. *-3*) in a previous task, when faced with a new task of learning what is a term (e.g. *-3x*), the learner first tries to build a parse tree for the whole term (e.g. *-3x*). But it fails because the grammar for signed number can only build the parse trees for some subsequence (e.g. *-3* in *-3x*). Nevertheless, the grammar learner does get some partially parsed sequences (e.g. the partial reduced sequence for *-3x* is *SignedNumber x*), and calls the original grammar learner on these partially parsed sequences.

During the Viterbi training phase, the learning algorithm estimates rule frequency using a Dirichlet distribution based on prior tasks; that is, it adds the applied rule frequency associated with the training problems of the current task to the recorded frequency from previous tasks. Note that it is possible that after acquiring new rules with new examples, in the Viterbi training phase, the parse trees for the training examples in the previous tasks have changed, and the recorded frequencies are no longer accurate, so this is not equivalent to combining the examples from the old task with the examples of the new task. By recording only the frequencies, instead of rebuilding the parse trees for all previous training examples in each cycle, we save both space and time for learning.

One key SimStudent limitation is that it depends on having a tutor that provides examples and feedback at a fine grained, step-by-step level. The more the steps are coarse-grained, requiring lots of inferences between them (that are not part of the functions in prior knowledge), the less likely SimStudent will effectively learn. More generally, it is not clear how well SimStudent's representation learning will generalize to all needs for accumulating relevant prior knowledge, particularly new features (used in when-learning). Although some work has begun to explore this issue [29], more investigation is needed.

Having acquired the grammar for deep features, when a new problem is given to the system, the learner will extract deep features by first building the parse tree of the problem based on the acquired grammar, and then extracting subsequences associated with feature symbols from the parse tree as target features. This model presented so far learns to extract deep features in a mostly unsupervised way without any goals or context from SimStudent problem solving. Later, we describe how to extend its ability by integrating it into SimStudent.

4. Empirical evaluation on deep feature learner

To evaluate the proposed deep feature learner, we carried out two controlled experiments. We compared four alternatives of the proposed approach: 1) without transfer learning and no feature focus; 2) without transfer learning, but with feature focus; 3) with transfer learning (from unlabeled sub-grammars), and without feature focus; 4) with transfer learning from unlabeled sub-grammars and feature focus. Learners without labeled feature have no way of knowing what the feature is; instead, we report the accuracy that would be obtained using the non-terminal symbol that mostly frequently corresponds to the feature sub-grammar in the training examples. Note that we did not compare the proposed deep feature learner with the inside–outside algorithm, as Li et al. [30] have shown that the base learner (i.e. the learner with no extension)

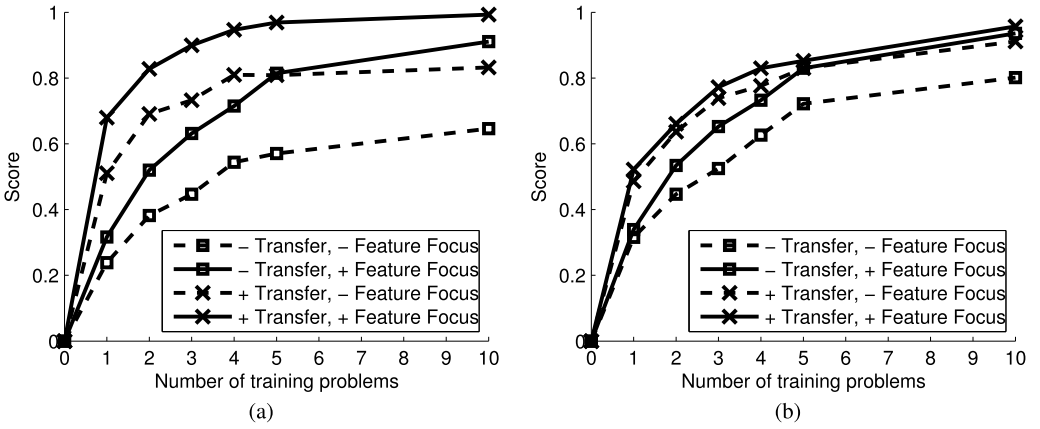

Fig. 5. Learning curve in synthetic domain with (a) subtask to task transfer problems (b) overlapping tasks transfer problems.

outperforms the inside–outside algorithm.[2] All the experiments were run on a 2.53 GHz Core 2 Duo Macbook with 4 GB of RAM.

4.1. Experimental design in synthetic domains

In order to understand the generality and scalability of the proposed approach, we first designed and carried out experiments in synthetic domains. We carried out the experiment with two types of transfer in non-recursive domains, the sub-grammar transfer and the overlapping-grammar transfer. The transfer learners (#3 and #4 above) were trained on the first grammar before being trained and tested on the second grammar. The non-transfer learners (#1 and #2 above) were only trained and tested on the second grammar.

In details, each non-recursive domain is formed by randomly generated rules that form a binary and-or tree. The first type is a sub-grammar transfer. We randomly generated two sets of non-recursive rules, S_1 and S_2. S_1 is a sub-grammar in S_2. The size (in terms of number of non-terminal symbols) of S_1 is roughly half of S_2. In order to further understand how transfer learning affects learning efficiency, we carried out a second experiment where the previous grammar is not a subset of the current grammar. Rather, this second experiment tests overlapping grammar transfer. We randomly generated two non-recursive domains, S_1 and S_2, where S_1 overlaps with S_2 in the feature sub-grammar.

The transfer learners (#3 and #4 above) were trained first on S_1. Each training record contains a full observation sequence and a set of subsequences (usually just one) associated with the feature. If S_1 contains n_1 non-terminal symbols in S_1, the number of training records is $10n_1$. Then, all four learners were trained and tested on S_2. The number of training records ranges from zero to ten. If S_2 contains n_2 non-terminal symbols in S_2, the number of testing sequences is $10n_2$.

For each testing record, we compared the feature recognized by the oracle grammar with those recognized by the acquired grammar. The score is the percentage of times the acquired grammar recognized the same feature as the oracle grammar averaged over 100 randomly generated feature extraction tasks. In the results section, we show the score achieved by the learner given increasing number of training examples, as *learning curve*.

4.2. Experiment results in synthetic domains

We evaluated both the efficiency and the scalability of the proposed algorithm. For learning efficiency, we measured the learning curves of the learners. For scalability, we looked at the accuracy, time, and size of the acquired grammars with different domain sizes.

Rate of learning: In order to test the learning speed, we randomly generated 100 sub-grammar-grammar pairs, $\langle S_{1,i}, S_{2,i}\rangle$, where $i = 1, 2, \ldots 100$. For each pair, $\langle S_{1,i}, S_{2,i}\rangle$, $S_{2,i}$ has 20 non-terminal symbols. We measured the scores of the four learners of every sub-grammar-grammar pair given different numbers of training sequences. The results are shown in Fig. 5(a) and 5(b). We can see that the learner with both transfer learning and feature focus (*+Transfer +Feature Focus*) has the steepest learning curve. In the sub-grammar transfer case, with ten training records, the learner (*+Transfer + Feature Focus*) achieves score 0.99, which is much higher than the score of the base learner without transfer learner and feature focus (*-Transfer -Feature Focus*), 0.65. Learners with a single extension (*-Transfer +Feature Focus*, and *+Transfer -Feature Focus*) have a slower learning curve comparing with the learner with both extensions (*+Transfer +Feature Focus*), but both outperform the base learner (*-Transfer -Feature Focus*).

[2] http://rakaposhi.eas.asu.edu/nan-tist.pdf.

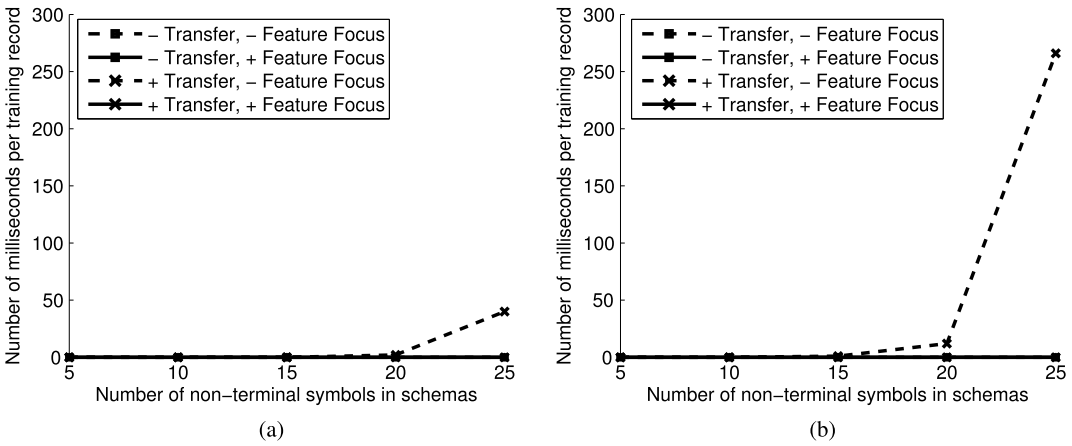

Fig. 6. Time during learning with different domain sizes with (a) subtask to task transfer problems. (b) overlapping tasks transfer problems.

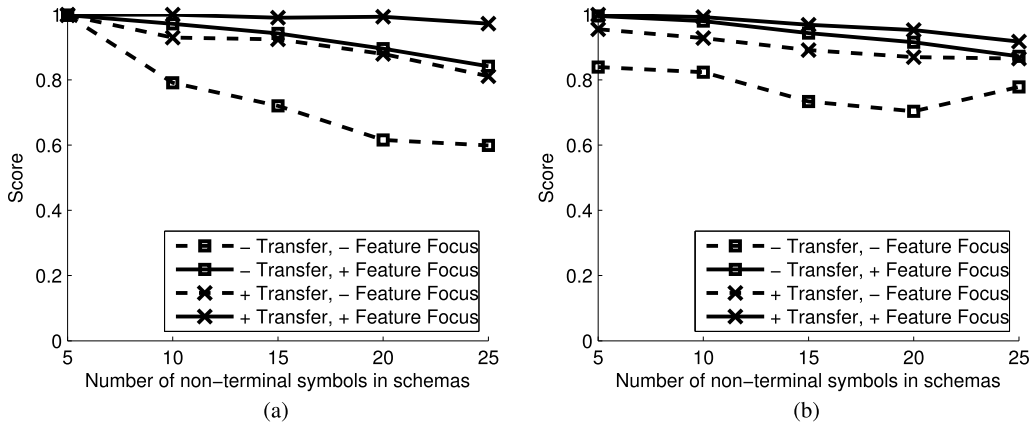

Fig. 7. Score with different domain sizes with (a) subtask to task transfer problems. (b) overlapping tasks transfer problems.

Note that the transfer-only learner (*+Transfer -Feature Focus*) has a better score with a small number of training records than the learner with only feature focus (*-Transfer +Feature Focus*). But after ten training records, the learner with only feature focus (*-Transfer +Feature Focus*) catches up with the transfer-only learner (*+Transfer -Feature Focus*). This situation appears in both transfer tasks.

Not surprisingly, since sub-grammar transfer is explicitly trained on the feature in task one before switching to task two, the difference between learners with and without transfer learning, is larger than that in overlapping task transfer. Moreover, since we randomly pick features from randomly generated grammars in the overlapping task transfer, it is possible that the selected feature is at a relatively low level in the hierarchy, and thus corresponds to very short subsequences or one specific action sequence (no disjunctions). In this case, the targeted feature is easy to learn, and the benefit of transfer is diminished. But even so, all the extended learners (*-Transfer +Feature Focus*, and *+Transfer -Feature Focus*, and *+Transfer +Feature Focus*) still outperform the base learner (*-Transfer -Feature Focus*).

Scalability of the learning algorithm: In order to understand the scalability of the proposed algorithms, we also tested the performance of the four learners in terms of accuracy, time, and size of acquired grammar, with different numbers of domain sizes. The scores of learners in domains with five to twenty-five non-terminal symbols are shown in Fig. 7(a) and 7(b). We can see that with ten training records, the learner with both extensions (*+Transfer +Feature Focus*) performs the best among all four learners, while the base leaner (*-Transfer -Feature Focus*) shows the fastest drop with increasing size of domains. The two learners with a single extension (*-Transfer +Feature Focus*, and *+Transfer -Feature Focus*) perform roughly equally well. With domains of size 25, the scores of the two learners are no more than 3% apart.

As for the average time spent on each training record, as shown in Fig. 6(a) and Fig. 6(b), all learners acquire the targeted feature within a reasonable amount of time. While the transfer-only learner (*+Transfer -Feature Focus*) took 266 milliseconds per training record with domains of size 25, all other learners (*-Transfer -Feature Focus*, and *-Transfer +Feature Focus*, and *+Transfer +Feature Focus*) took less than 1 millisecond per training record during learning. We found out that the learner with transfer learning from unlabeled sub-grammars, but without the feature focus (*+Transfer -Feature Focus*) runs slower with domains of larger sizes. In fact, it needs 266 milliseconds per training record. This is because in the second task,

maintaining the grammar rules acquired from previous task requires much more work in the Viterbi training step. Besides, the feature focus mechanism enables the learner to separate a whole sequence into small subsequences and focus on one small piece at a time during learning. Since the transfer-only learner (*+Transfer -Feature Focus*) does not consider feature focus during learning, it takes longer than the other learners.

We also looked at the conciseness of the acquired grammar, since grammars of larger sizes typically slow down the process of future learning, as well as feature extraction. To measure conciseness, we compute the *symbol ratio* between the number of symbols in the learned grammar and the original schema. With sub-grammar transfer, all four learners acquired grammars of roughly equal sizes. With overlapping grammar transfer, since the learners with transfer learning from unlabeled sub-grammars (*+Transfer -Feature Focus*, and *+Transfer +Feature Focus*) also need to maintain knowledge acquired from the previous task, both have a higher symbol ratio then the other two learners. This is reasonable since they do have more knowledge embedded in the grammar.

4.3. Experimental design in algebra

In order to understand whether the proposed algorithm is a good model of real students, we carried out a controlled simulation study in algebra using the *+Transfer+Feature Focus* learner. Accelerated future learning, in which learning proceeds more effectively and more rapidly because of prior learning, is an essential measure of robust learning. We have little by way of precise understanding of the learning mechanisms that yield these results. A computational model of accelerated future learning that fits student learning data would be a significant achievement in theoretical integration within the learning sciences, and reveal insights on improving current education technologies. In this study, we focus on using the proposed learning algorithm to model and to get a better understanding of accelerated future learning.

Two possible causes for accelerated future learning are a better learning strategy or stronger prior knowledge. Learning with feature focus is an example of using a better learning strategy during knowledge acquisition. Transfer learning from unlabeled sub-grammars is an example of developing stronger prior knowledge from previous training to prepare for better future learning. The objective of this study is to test 1) whether the proposed model could yield accelerated future learning with stronger prior knowledge and better learning strategies, 2) if so, how prior knowledge and learning strategies affect the learning outcome. In other words, can we model how students may learn later tasks more effectively after prior unsupervised or semi-supervised experience?

4.3.1. Method

In order to understand the behavior of the proposed model, we designed three sequences of learning tasks of increasing difficulties. Here, we refer these sequences as *curricula*. There were three tasks used across the three curricula. Each learning task is again a feature extraction task, that can be captured by an *oracle grammar*, where the target feature can be represented as a non-terminal symbol in a grammar rule. Task one is to learn about signed numbers. Task two is to learn how to recognize a coefficient from expressions in the form of *{SignedNumber x}*. Task three is to learn how to recognize a constant in the left-hand side from equations in the form of *{SignedNumber x - Integer = SignedNumber}*. The three curricula contain 1) task one, task two; 2) task two, task three; 3) task one, task two, and task three.

There were also 10 training sequences to control for a difference in training problems. The training data were randomly generated following the grammar corresponding to each task. For instance, task two's grammar is shown in Table 1. In all but the last task, each learner was given 10 training problems, which are enough for each learner to acquire the representation knowledge in the learning task. For the last task, each learner was given one to five training records to show the rate of learning as the number of training records increases.

To measure learning gain under each training condition, a separate testing phase was carried out. In this phase, the system no longer updated its knowledge based on the given records. It simply extracted features from the given records using the knowledge acquired in the training phase. The quality of the knowledge was evaluated by the accuracy of the extracted features. Both systems were tested on 100 expressions in the same form of the training data in the last task. For each testing record, we compared the feature extracted by the oracle grammars with that recognized by the acquired grammars. Note that in task two, 4% of the testing problems in task two were x and $-x$. To assess the accuracy of the model, we asked both systems to extract the feature from each problem. We then used the oracle grammar to evaluate the correctness of output. A brief summary of the method is shown in Table 2.

4.3.2. Measurements

To assess the learning outcome, we measured the learning rate in the last task of each curriculum to evaluate the effectiveness of the learners. The experiment tested whether the proposed model is able to yield accelerated future learning, that is, a faster learning rate in the last task either because of transfer prior learning or because of a better learning strategy. We compare the same four learners used in the previous simulations, that is, the combinations of transfer or not and feature focus or not. To evaluate the learning rate, we report learning curves for all four learners by the number of training problems given in the last task. The accuracy of the feature extraction task is averaged over 10 training conditions.

This experiment focuses on measuring the learning rate. In Section 7, we also test whether the proposed model fits with real student data. We show that after integrating the proposed model into a simulated student, the extended simulated student can be used to automatically discover student models. The discovered model fits with real student data better than

Table 2
Method summary.

Three tasks:	T1, learn signed number
	T2, learn to find coefficient from expression
	T3, learn to find constant from equation
Three curricula:	T1 → T2
	T2 → T3
	T1 → T2 → T3
Number of training condition:	10
Training size in all but last tasks:	10
raining size in the last task:	1, 2, 3, 4, 5
Testing size:	100

human-generated models. This indicates that the extended simulated student simulates the real student learning process well.

4.4. Experiment results in algebra

As shown in Fig. 8(a), with curriculum one, all four learners acquired better knowledge with more training examples.[3] With five training problems, either transfer or feature focus are sufficient to acquire knowledge of score 0.96, while the base learner with neither was only able to achieve a score around 0.5. Both learners with transfer learning (*+Transfer -Feature Focus*, and *+Transfer +Feature Focus*) have the steepest learning curve. In fact, they reached a score of 0.96 with only one training example. The feature focus learner (*-Transfer +Feature Focus*) learns more slowly than the learners with transfer learning (*+Transfer -Feature Focus*, and *+Transfer +Feature Focus*), but is able to reach a score of 0.96 after five training examples. Learners that transfer prior grammar learning achieve faster future learning than those without transfer learning. The base learner (*-Transfer -Feature Focus*) learns most slowly. A careful inspection shows that without feature focus and transfer learning, the base learner was not able to acquire a grammar rule with a non-terminal symbol generally corresponding with the feature "coefficient", though it does learn to identify positive coefficients (like many novice students). This causes the failure of identifying the feature symbol. Comparing the base learner (*-Transfer -Feature Focus*) and the learner with feature focus (*-Transfer +Feature Focus*) we can see that a better learning strategy also yields a steeper learning curve.

Similar results were also observed with curriculum two and curriculum three as shown in Fig. 8(a) and 8(b).[4] In curriculum two, case by case inspection shows that, in some conditions, if a transfer learner, (*+Transfer -Feature Focus*) remembers the wrong knowledge acquired from task two, and transferred this knowledge to task three, the learner will perform even worse than the learner with no prior knowledge (*-Transfer -Feature Focus*). This indicates that more knowledge does not necessarily lead to steeper learning curves. Transferring incorrect knowledge leads to less learning.

In all three curricula, the transfer learner (*+Transfer -Feature Focus*) always outperforms the learner with the semantic non-terminal constraint (*-Transfer +Feature Focus*). This suggests that prior knowledge is more effective in accelerating future learning than this learning strategy.

5. Integrating deep feature representation learning into SimStudent

Given the promising results shown above, we believe the proposed deep feature learner is effective in acquiring representation knowledge, and is a good model of real students. To evaluate how the deep feature representation learner could affect the problem-solving learning of an intelligent agent, in this section, we present an integration of deep feature learning into such an agent, SimStudent. This extension integrates ideas of theories of perceptual chunking [31] as a basis for improving knowledge representations that, in turn, facilitate better learning of problem solving skills. With the current extension, the representation is acquired by a deep feature learner before skill learning. Hence, the acquired representation does not change during the course of skill acquisition. As we will discuss in later sections, one possible future step is to use the feedback from the skill learning process to further refine the representation learning process.

As we have mentioned above, SimStudent is able to acquire production rules in solving complicated problems, but requires a set of operator functions given as prior knowledge. Some of the operator functions are domain-specific and require expert knowledge to build. In contrast, the feature learner acquires deep features that are essential for effective learning without requiring prior knowledge engineering. In order to both reduce the amount of prior knowledge engineering needed for SimStudent and to build a better model of real students, we present a novel approach that integrates the representation learner into SimStudent. Fig. 9 shows a comparison between a production rule acquired by the original SimStudent and the corresponding production rule acquired by the extended SimStudent. As

[3] The (*+Transfer -Feature Focus*, and *+Transfer +Feature Focus*) are overlapping.
[4] The (*+Transfer -Feature Focus*, and *+Transfer +Feature Focus*) are overlapping in Fig. 8(c).

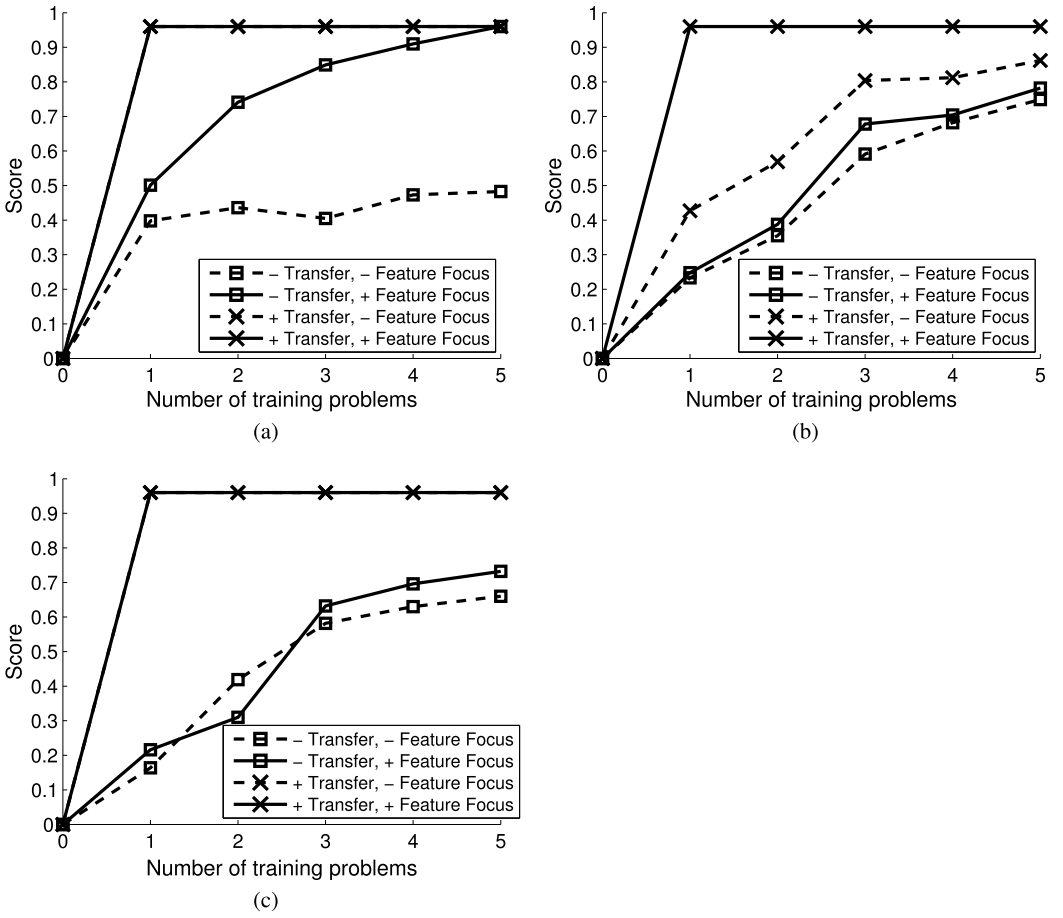

Fig. 8. Learning curves in the last task for four learners in curriculum (a) from task one to task two (b) from task two to task three (c) from task one and two to task three. Both prior knowledge transfer and the feature focus strategy produce faster learning.

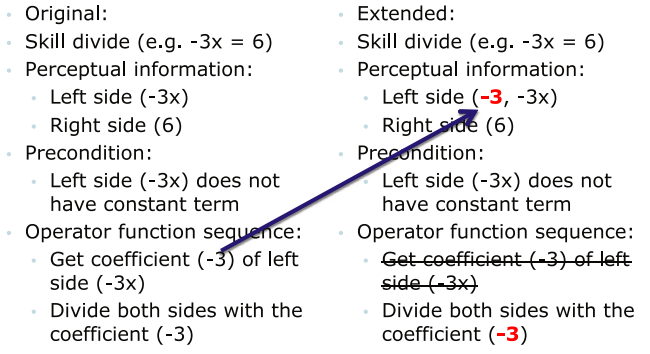

Fig. 9. Original and extended production rules for divide in a readable format. Grammar learning allows extraction of information in where-part of the production rule and eliminated the need for domain-specific function authoring (get-coefficient) for use in the how-part.

we can see, the essential change is that the coefficient of the left-hand side (i.e., *-3*) is included in the perceptual information part in the extended production rule. Therefore, the operator function sequence no longer needs the domain-specific operator, "get-coefficient". Then the question becomes how can the algorithm learn to include *-3* into the perceptual information part. To achieve this, we extended the perceptual learning algorithm as described below.

Previously, the perceptual information encoded in production rules was associated with elements in the graphical user interface (GUI) such as text field cells in the algebra equation solving interface. This assumption limited the granularity of observation SimStudent could achieve. As suggested by previous work [32], both experts and novices perceive mathematical expressions based on their syntactic structure. In fact, the deep features we have discussed previously directly correspond to

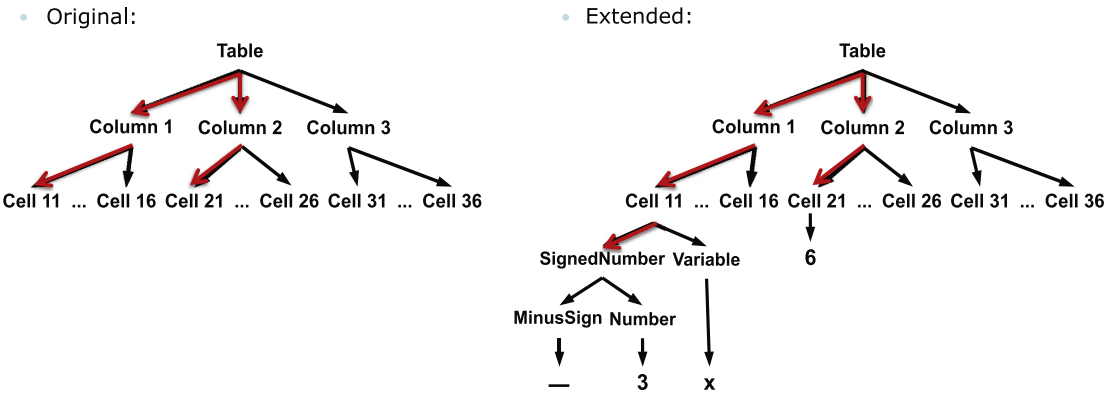

Fig. 10. Original and extended perceptual hierarchy.

such syntactic structure. Representing these deep perceptual features may enhance the performance of the learning agent, and may eliminate or reduce the need for authors/developers to manually encode domain-specific operator functions to extract appropriate information from appropriate parts of the perceptual input.

To improve perceptual representation, we extend the percept hierarchy of GUI elements to further include the parse tree for the content in the leaf nodes (e.g., text fields) by appending the parse trees as an extension of the GUI path learning to the associated leaf nodes. Fig. 10 shows a comparison of the original perceptual hierarchy and the extended perceptual hierarchy. Since our deep feature learner acquires probabilistic grammars, there may be more than one parse trees for a given input. In this case, we always use the most probable parse tree of the content in the leaf node. All of the inserted nodes are of type "subcell". In the algebra example, this extension means that for cells that represent expressions corresponding to the sides of the equation, the extended SimStudent appends the parse trees for these expressions to the cell nodes. Let's use *-3x* as an example. In this case, the extended hierarchy includes the parse tree for *-3x* as shown at of Fig. 4 as a subtree connecting to the cell node associated with *-3x*. With this extension, the coefficient (*-3*) of *-3x* is now explicitly represented in the percept hierarchy. If the extended SimStudent includes this subcell as a percept in production rules, as shown at the right side of Fig. 9, the new production rule does not need the first domain-specific operator function "coefficient".

However, extending the percept hierarchy presents challenges to the original perceptual learner. First of all, since the extended subcells are not associated with GUI elements, we can no longer depend on the author to specify relevant perceptual input for SimStudent, nor can we simply specify all of the subcells in the parse trees as relevant perceptual information; otherwise, the acquired production rules would include redundant information that would hurt the generalization capability of the perceptual learner. For example, consider problems *-3x=6* and *4x=8*. Although both examples could be explained by dividing both sides with the coefficient, since *-3x* has eight nodes in its parse tree, while *4x* has five nodes, the original perceptual learner will not be able to find one set of generalized paths that explain both training examples. Moreover, not all of the subcells are relevant percepts in solving the problem. For example, considering the problem *-3x=6*: among all inserted subcells, only *-3* is a relevant percept in solving the problem. Including unnecessary perceptual information into production rules could easily lead to computational issues. Second, since the size of the parse tree for an input depends on the input length, the fixed percept size assumption made by SimStudent no longer holds. Even with the same number of percepts, how the inserted percepts should be ordered is not immediately clear. To address these challenges, we extend the original perceptual learner to support acquisition of perceptual information with redundant and variable-length percept lists.

To do this, SimStudent first includes all of the inserted subcells as candidate percepts, and calls the operator function sequence learner to find an operator function sequence that explains all of the training examples. In our example, the operator function sequence for *(divide -3)* would only contain one operator function "divide", since *-3* is already included in the candidate percept list. The perceptual learner then removes all of the subcells that are not used by the operator function sequence from the candidate percept list. Hence, subcells such as -, *3* and *x* would not be included in the percept list any more. Since all of the training example action records share the same operator function sequence, the number of percepts remaining for each example action record should be the same. Next, the percept learner arranges the remaining subcell percepts based on their order of being used by the operator function sequences. After this process, the percept learner now has a set of percept lists that contains a fixed number of percepts ordered in the same fashion. We can then switch to the original percept learner to find the least general paths for the updated percept lists. In our example for skill "divide", as shown at the right side of Fig. 9, the perceptual information part of the production rule would contain three elements, the left-hand side and right-hand side cells which are the same as the original rule, and a coefficient subcell which corresponds to the left child of the variable term. Note that since we removed the redundant subcells, the acquired production rule now works with both *-3x=6* and *4x=8*.

Table 3
A list of operator functions used by SimStudents.

Operator function	Type	Example
GetOperand	domain-general	(get-operand *divide -3*) ⇒ *-3*
GenOne	domain-general	(generate-one) ⇒ *1*
Copy	domain-general	(copy *3*) ⇒ *3*
Add	domain-general	(add *3 5*) ⇒ *8*
Sub	domain-general	(subtract *8 2*) ⇒ *6*
Multiply	domain-general	(multiply *3 5*) ⇒ *15*
Divide	domain-general	(divide *8 3*) ⇒ *8/3*
Concat	domain-general	(concatenate *5 x*) ⇒ *5x*
ReverseSign	domain-general	(reverse-sign *8x*) ⇒ *-8x*
VarName	domain-general	(var-name *8x+2*) ⇒ *x*
Denominator	domain-general	(denominator *3/5x*) ⇒ *5x*
Numerator	domain-general	(numerator *3/5x*) ⇒ *3*
SkillAdd	domain-general	(skill-add *3*) ⇒ *add 3*
SkillSubtract	domain-general	(skill-subtract *3*) ⇒ *subtract 3*
SkillMultiply	domain-general	(skill-multiply *3*) ⇒ *multiply 3*
SkillDivide	domain-general	(skill-divide *3*) ⇒ *divide 3*
SkillCltOperand	domain-general	(skill-clt *3x+4+5x-2*) ⇒ *clt 3x+4+5x-2*
SkillMtOperand	domain-general	(skill-mt *3x*) ⇒ *mt 3x*
EvalArithmetic	domain-specific	(eval-arithmetic *3x+4+5x-2*) ⇒ *8x+2*
AddTerm	domain-specific	(add-term *3x+4 5x-2*) ⇒ *8x+2*
SubTerm	domain-specific	(subtract-term *3x+4 5x-2*) ⇒ *-2x+2*
DivTerm	domain-specific	(divide-term *8x+2 2*) ⇒ *4x+1*
MulTerm	domain-specific	(multiply-term *8x+2 2*) ⇒ *16x+4*
Coefficient	domain-specific	(coefficient *-3x*) ⇒ *-3*
FirstTerm	domain-specific	(first-term *3x+5*) ⇒ *3x*
LastTerm	domain-specific	(last-term *3x+5*) ⇒ *5*

As we mentioned before, SimStudent holds a perceptual grounding bias, which is how the input world is represented is an essential inductive bias in skill learning. With this extension, "deep features" become explicitly represented in the perceptual hierarchy, and the patterns in extracting them is automatically acquired by the perceptual information learning algorithm. In these cases, no feature extraction operator functions are needed to be predefined in order to calculate what the deep features are. The production rule can directly get such features by matching the learned paths of the perceptual information part with the extended the perceptual hierarchy. SimStudent gets a basic understanding of what is a term, what is a constant, and so on through the perceptual information paths. This extension reduces the number of operator functions an intelligent agent developer needs to define in order to generate a SimStudent in maths. In addition, it also reduces the number of operator functions required in the production rules, as the feature extraction operator functions are no longer needed, which eases the production rule learning process.

6. Experimental study on SimStudent integrated with deep feature learner

In order to evaluate whether the extended SimStudent is able to acquire correct knowledge with reduced prior knowledge engineering, we carried out an experiment in the algebra domain. We use algebra as the testing domain because it is one of the most important learning tasks for middle school students. It is also relatively more complicated than other similar domains such as multi-column addition and fraction addition. Although not reported here, we have demonstrated elsewhere [12] that the extended SimStudent yields better learning performance, with less knowledge engineering, than the original SimStudent in fraction addition and stoichiometry.

6.1. Experiment design

Since our goal is to build an intelligent agent that models skill acquisition of real students, instead of using randomly generated problems, as training sets four problem sets we select that were used to teach real students as training sets. More specifically, the problem sets are from high school students who used Carnegie Learning Algebra I Tutor. The sizes of the training sets are 13, 14, 35 and 35 problems. We also choose 10 other problems from real student data as the testing set.

We compare the extended SimStudent with the original SimStudent given different amounts of prior knowledge. The extended SimStudent is first trained on a sequence of deep feature learning tasks, which include learning what is a signed number, what is a term, and what is an expression. We then construct a weak operator function set and a strong operator function set, simulating weak and strong prior knowledge. The weak operator function set contains 24 domain-general operator functions such as copying a string, adding two numbers and so on. The strong operator function set includes the weak operator function set plus 12 domain-specific operator functions such as getting the coefficient, adding two terms and so on. Among all the given operator functions, Table 3 shows the list of operator functions that are used in the production rules acquired by the four SimStudents. Two original SimStudents and one extended SimStudent are tested. One of the

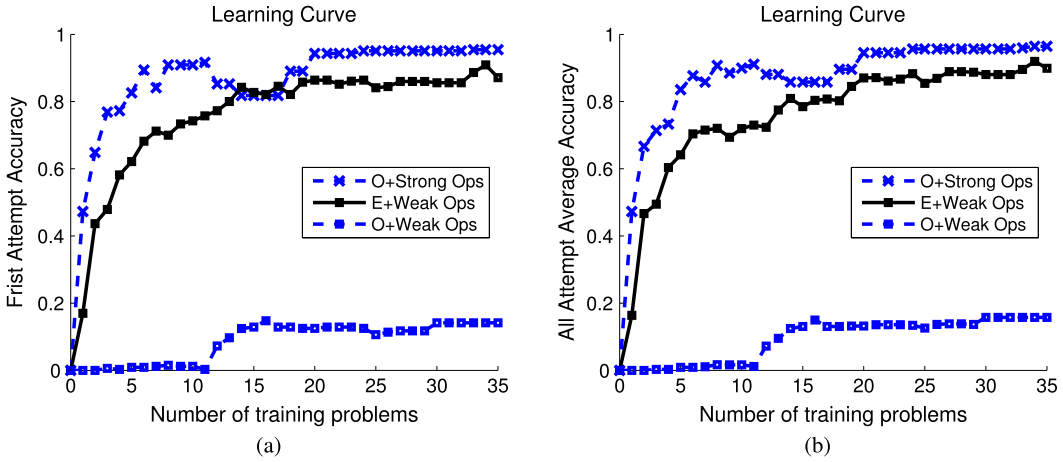

Fig. 11. Learning curves for three learners (a) first-attempt accuracy (b) all-attempt average accuracy.

original SimStudents is given the strong operator function set (*O+Strong Ops*), while the other is provided with the weak operator function set (*O+Weak Ops*). The extended SimStudent is given only the weak operator function set (*E+Weak Ops*).

6.2. Experiment results

Evaluation of learning speed: The first study we carried out focuses on evaluation of learning speed. Since it is often possible to have more than one way of solving the same algebra equation, it is also possible that there is more than one skill applicable at the same time. In order to evaluate the performance of all applicable skills, we use two different measurements in evaluating the learning efficiency. The first measurement is called *first-attempt accuracy*, where for each testing problem, the learner receives score 1 if it proposes a correct step at its first attempt, and gets 0 otherwise. This measurement is closest to the evaluation method used in real classroom settings, where even if the student has more than one thought in solving the problem, only the one solution he/she writes out is graded. The second measurement, *all-attempt average accuracy*, focuses more on the average performance across all applicable skills. Instead of only counting for the first attempt, the evaluator scores the correctness of all applicable skills, and reports the average score as the all-attempt average accuracy.

The average learning curves for the three SimStudents are shown in Figs. 11(a) and 11(b). The blue lines correspond to the original SimStudents, whereas the black lines represent the performance of the extended SimStudent. As we can see in the figures, with both measurements, there is a huge gap between the two original SimStudents with (*O+Strong Ops*) and without (*O+Weak Ops*) strong operator functions. Our focus is to test whether the extended SimStudent is able to achieve performance comparable to the original SimStudent with strong operator functions (*O+Strong Ops*) while given only the weak operator function set. As the result shows, the extended SimStudent (*E+Weak Ops*) learns slower than the original SimStudent with strong operator functions (*O+Strong Ops*) at the very beginning, but gradually catches up with the original SimStudent. With 35 training problems, the all-attempt average accuracy of the extended SimStudent (*E+Weak Ops*) reaches 90%, which is only 5% lower than the original SimStudent with strong operator functions (*O+Strong Ops*). This suggests that with the deep feature learner, the extended SimStudent is able to achieve comparable performance without prior domain-specific knowledge engineering.

Evaluation of knowledge engineering needed: We evaluate the learner performance with two measurements, the total amount of knowledge used and the learning speed. For the first measurement, we look at the production rules acquired from the two problem sets of size 35, and report the average number of domain-specific and domain-general operator functions used in the two rule sets. Recall that domain-specific operator functions usually require more knowledge engineering than domain-general operator functions. As shown in Table 4, the SimStudent (*O+Strong Ops*) that was given strong operator functions used 8 of the domain-specific operator functions, plus 7.5 domain-general operator functions on average across the two training sequences of size 35. In contrast, the extended SimStudent (*E+Weak Ops*) was not given any domain-specific functions and used 12 domain-general operator functions, which indicates much less knowledge engineering effort. In addition, the original SimStudent with only domain-general operator functions (*O+Weak Ops*) used 14.5 domain-general operator functions, which suggests that it needs a larger amount of prior knowledge engineering than the extended SimStudents. However, as we have seen before, it performs much worse than the extended SimStudents.

To get a better understanding on where the knowledge engineering effort is reduced, we took a closer look at the data. As we knew, the extended SimStudent (*E+Weak Ops*) was able to achieve comparable performance without using any of the domain-specific operators. How did the feature learning process aid skill acquisition? For example, the domain-specific operator “subtract-term” was needed by the original SimStudent (*O+Strong Ops*) for “skill-subtract” to perform operations

Table 4
Average number of strong and weak operator functions used in acquired production rules.

	Operator functions used	
	Strong domain-specific	Weak domain-general
Original+Strong Ops	8	7.5
Extended+Weak Ops	0	12
Original+Weak Ops	0	14.5

such as *(3x+4) - 4*. Since we have extended the perceptual hierarchy, *3x* is now explicitly represented. The extended SimStudent can simply test whether the constant term of *3x+4* (i.e., *4*) equals to the subtrahend *4*. If so, it directly grabs *3x* from the hierarchy and returns it as the result. In this case, the need of the operator "subtract-term" is removed. Instead, only an "equal" feature test and a "copy" operator functions were needed, which are much simpler than the "subtract-term" operator function.

7. Using SimStudent to discover better cognitive models

As mentioned above, we are not only interested in building a learning agent: we would also like to construct a learning agent that simulates how students acquire knowledge. In this section, we are going to present an approach that automatically discovers cognitive models using the extended SimStudent. If the discovered model turns out to be a good cognitive model, we should be able to conclude that the extended SimStudent simulates the real student learning process well. A cognitive model is a set of *knowledge components (KC)* encoded in intelligent tutors to model how students solve problems [33]. The set of KCs includes the component skills, concepts, or percepts that a student must acquire to be successful on the target tasks. For example, a KC in algebra can be how students should proceed given problems of the form *-Nv=N* (e.g., *-3x=6*). The cognitive model provides important information to automated tutoring systems in making instructional decisions. Better cognitive models match with real student learning behavior, that is, changes in performance over time. They are capable of predicting task difficulty and transfer of learning between related problems, and can be used to yield better instruction.

Traditional ways to construct models include structured interviews, think-aloud protocols, rational analysis, and so on. However, these methods are often time-consuming, and require expert input. More importantly, they are highly subjective. Previous studies [34,35] have shown that human engineering of these models often ignores distinctions in content and learning that have important instructional implications. Other methods such as Learning Factor Analysis (LFA) [36] apply an automated search technique to discover cognitive models. It has been shown that these automated methods are able to find better cognitive models than human-generated ones. Nevertheless, LFA requires a set of human-provided factors given as input. These factors are potential KCs. LFA carries out the search process only within the space of such factors. If a better model exists but requires unknown factors, LFA will not find it. Therefore, having an approach that is able to find the "right" set of factors that lead to better predictions of human data is essential in the success of cognitive modeling.

To address this issue, we propose a method that automatically discovers cognitive models without depending on human-provided factors. The system uses the extended SimStudent to acquire skill knowledge. Each production rule corresponds to a KC that students need to learn. The model then labels each observation of a real student based on skill application.

7.1. Method

In order to evaluate the effectiveness of the proposed approach, we carried out a study using an algebra dataset. We compared the SimStudent model with a human-generated KC model by first coding the real student steps using the SimStudent model and the human-generated KC model, and then testing how well the two model codings predict real student data. Note that DataShop [37] has 21 different KC models for the current dataset, the human-generated KC model we selected here is one of the best models in predicting human student behavior among the existing student models.

For the human-generated model, the real student steps were first coded using the "action" label associated with a correct step transaction, where an action corresponds to a mathematical operation(s) to transform an equation into another in a way that makes progress toward the solution. As a result, the "default" model contains eight KCs defined (called the Action KC model) – add, subtract, multiply, divide, distribute, clt (combine like terms), mt (simplify multiplication), and rf (reduce a fraction). Four KCs associated with the basic arithmetic operations (i.e., add, subtract, multiply, and divide) were then further split into two KCs for each, namely a skill to identify an appropriate basic operator and a skill to actually execute the basic operator. The former is called a *transformation* skill whereas the latter is a *typein* skill. As a consequence, there were 12 KCs defined (called the Action-Typein KC model). Not all steps in the algebra dataset were coded with these KC models – some steps are about a transformation that we do not include in the Action KC model (e.g., simplify division). There were 9487 steps that can be coded by both KC models mentioned above. The "default" KC model, which were defined by the productions implemented for the cognitive tutor, has only 6809 steps that can be coded. To make a fair comparison between the "default" and "Action-Typein" KC models, we took the intersection of those 9487 and 6809 steps. As a result, there were 6507 steps that can be coded by both the default and the Action-Typein KC models. We then defined a new KC

model, called the Balanced-Action-Typein KC model that has the same set of KCs as the Action-Typein model but is only associated with these 6507 steps, and used this KC model to compare with the SimStudent model.

To generate the SimStudent model, SimStudent was tutored on how to solve linear equations by interacting with the Carnegie Learning Algebra I Tutor, like a human student. As the training set for SimStudent, we selected 40 problems that were used to teach real students. Given all of the acquired production rules, for each step a real student performed, we assigned the applicable production rule as the KC associated with that step. In cases where there was no applicable production rule, we coded the step using the human-generated KC model (Balanced-Action-Typein). Each time a student encounters a step using some KC, it is considered as an "opportunity" for that student to show mastery of that KC, and learn the KC by practicing it.

Having finished coding real student steps with both models (the SimStudent model and the human-generated model), we used the Additive Factor Model (AFM) [36] to validate the coded steps. AFM is an instance of logistic regression that models student success using each student, each KC, and the KC by opportunity interaction as independent variables as shown in Equation 1 below,

$$\ln \frac{p_{ij}}{1-p_{ij}} = \theta_i + \sum_k \beta_k Q_{kj} + \sum_k Q_{kj}(\gamma_k N_{ik}) \tag{1}$$

Where:

i represents a student i.
j represents a step j.
k represents a skill or KC k.
p_{ij} is the probability that student i would be correct on step j.
θ_i is the coefficient for proficiency of student i.
β_k is coefficient for difficulty of the skill or KC k
Q_{kj} is the Q-matrix cell for step j using skill k, i.e., whether completing step j needs skill k.
γ_k is the coefficient for the learning rate of skill k;
N_{ik} is the number of practice opportunities student i has had on the skill k.

AFM assumes that each student has his/her own proficiency of learning to start with, while the rate of learning for each KC is roughly the same across students. Given the past behavior of a set of students on a set of tests, AFM is able to predict the probability of success for one specific student on a specific problem step. In our case, the KC model is used in defining the variables in the AFM model. Once the form of the AFM model (e.g., how many KCs are there in the domain, what are the KCs), we fit the AFM model with the human student dataset, and test to see how well it predicts future behavior of the students. We utilized DataShop [37], a large repository that contains datasets from various educational domains as well as a set of associated visualization and analysis tools, to facilitate the process of evaluation, which includes generating learning curve visualization, AFM parameter estimation, and evaluation statistics including AIC (Akaike Information Criterion) and cross validation.

7.2. Dataset

We analyzed the same data from 71 students who used an Carnegie Learning Algebra I Tutor unit on equation solving. The students were typical students at a vocational-technical school in a rural/suburban area outside of Pittsburgh, PA. The problems varied in complexity, for example, from simpler problems like *3x=6* to harder problems like *x/-5+7=2*. A total of 19,683 transactions between the students and the Algebra Tutor were recorded, where each transaction represents an attempt or inquiry made by the student, and the feedback given by the tutor.

7.3. Measurements

To evaluate the quality of the SimStudent model and the human-generated KC model, we measured how well the models can be used in predicting human student behavior. The quality of the prediction generated by the KC models using AFM is measured by AIC and a 3-fold cross validation. AIC measures the fit to student data while penalizing over-fitting. More specifically, AIC measures the log likelihood of the model comparing with the human data, and punishes the model on using too many parameters. The cross validation was performed over three folds with the constraint that each of the three training sets must have data points for each student and KC. We report the root mean-squared error (RMSE) averaged over three test sets.

7.4. Experiment results

The SimStudent model contains 21 KCs. Both the AIC (6448) and the cross validation RMSE (0.3997) are lower than the human-generated model (AIC 6529 and cross validation 0.4034). This indicates that the SimStudent model better predicts real student behavior.

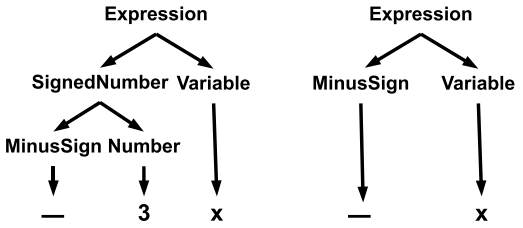

Fig. 12. Different parse trees for -3x and -x.

In order to understand whether the differences are statistically reliable or not, we carried out two significance tests. The first significance test evaluates whether the SimStudent model is actually able to make better predictions than the human-generated model. During the cross validation process, each student step was used once as the test problem. We took the predicated error rates generated by the two KC models for each step during testing. Then, we compared the KC models' predictions with the real student error rate (0 if the student was correct at the first attempt, and 1 otherwise). After removing ties, among all 6494 student steps, the SimStudent model made better predictions than the human-generated KC model on 4260 steps. A sign test on this shows that the SimStudent model is significantly ($p < 0.001$) better in predicting real student behavior than the human-generated model. In the second test, due to the random nature of the assignment to folds in cross validation, we evaluated whether the lower RMSE achieved by the SimStudent model was consistent or could be due to chance. To do this, we repeated the cross validation 20 times, and calculated the RMSE for both models. Across the 20 runs, the SimStudent model consistently outperformed the human-generated model: in particular, a paired t-test shows the SimStudent model is significantly ($p < 0.001$) better than the human-generated model.[5] Therefore, we conclude that the SimStudent model is a reliably better student model than the human-generated KC model.

7.5. Implications for instructional decision

We can inspect the data more closely to get a better qualitative understanding of why the SimStudent model is better and what implications there might be for improved instruction [38]. Among the 21 KCs learned by the SimStudent model, there were 17 transformation KCs and four typein KCs. It is hard to map the SimStudent KC model directly to the human-generated KC model. Approximately speaking, the distribute, clt (i.e. combine like terms), mt, rf KCs as well as the four typein KCs are similar to the KCs defined in the human-generated KC model. The transformation skills associated with the basic arithmetic operators (i.e., add, subtract, multiply and divide) are further split into finer grain sizes based on different problem forms.

One example of such split is that SimStudent created two KCs for division. The first KC (simSt-divide) corresponds to problems of the form *Ax=B*, where both *A* and *B* are signed numbers, whereas the second KC (simSt-divide-1) is specifically associated with problems of the form *-x=A*, where A is a signed number. This is caused by the different parse trees for *Ax* vs *-x* as shown in Fig. 12. To solve *Ax=B*, SimStudent simply needs to divide both sides with the signed number *A*. On the other hand, since *-x* does not have *-1* represented explicitly in the parse tree, SimStudent needs to see *-x* as *-1x*, and then to extract *-1* as the coefficient. If SimStudent is a good model of human learning, we expect the same to be true for human students. That is, real students should have greater difficulty in making the correct move on steps like *-x=6* than on steps like *-3x=6* because of the need to convert (perhaps just mentally) *-x* to *-1x*. To evaluate this hypothesis, we computed the average error rates for a set of problem types – these are shown with the solid line in Fig. 13 with the problem types defined in forms like *-Nv=N*, where the *N*s are any integer number and the *v* is a variable (e.g., *-3x=6* is an instance of *-Nv=N* and *-6=-x* is an instance of *-N=-v*). The problem types are sorted by increasing error rates. In other words, the problem types to the right are harder for human students than those to the left.

We also calculated the mean of the predicted error rates for each problem type for both the human-generated model and the SimStudent model. Consistent with the hypothesis, as shown in Fig. 13, we see that problems of the form *Ax=B* (average error rate 0.283) are much simpler than problems of the form *-x=A* (average error rate 0.719). The human-generated model predicts all problem types with similar error rates (average predicted error rate for *Ax=B* 0.302, average predicted error rate for *-x=A* 0.334), and thus fails to capture the difficulty difference between the two problem types (*Ax=B* and *-x=A*). The SimStudent model, on the other hand, fits with the real student error rates much better. It predicts higher error rates (0.633 on average) for problems of the form *-x=A* than problems of the form *Ax=B* (0.291 on average).

SimStudent's automatic split of the original division KC into two KCs, simSt-divide and simSt-divide-1, suggests that the tutor should teach real students to solve two types of division problems separately. In other words, when tutoring students with division problems, we should include two subsets of problems, one subset corresponding to simSt-divide problems (*Ax=B*), and one specifically for simSt-divide-1 problems (*-x=A*). We should perhaps also include explicit instruction that highlights for students that *-x* is the same as *-1x*.

[5] Note that differences between competitors in the KDD Cup 2010 (https://pslcdatashop.web.cmu.edu/KDDCup/Leaderboard) have also been in this range of thousands in RMSE.

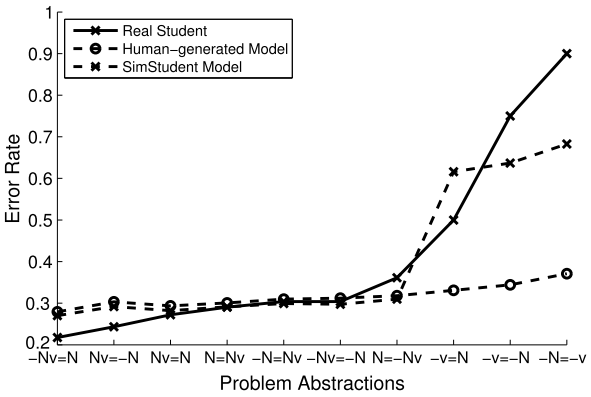

Fig. 13. Error rates for real students and predicted error rates from two student models.

8. Related work

The main contribution of this paper is to build a human-like intelligent agent, and to reduce the amount of knowledge engineering required by integrating feature learning into an agent. Previous work in cognitive science has shown that "chunking" is an important component of human knowledge acquisition. Theories of the chunking mechanisms [9,31,39] have been constructed. EPAM [9] is one of the first chunking theories proposed to explain key phenomena of expertise in chess. Learning occurs through the incremental growth of a discrimination network, where each node in the network is a chunk. It has been shown that chunks can be used to suggest plans, moves and so on. A later version of EPAM, EPAM-IV [31], extends the basic chunking mechanism to support a retrieval structure that enables domain-specific material to be rapidly indexed. In these theories, chunks usually refer to perceptual chunks. In addition, CHREST [39] proposes a template theory, where the discrimination network contains both perceptual chunks and action chunks. A more detailed review of these works can be found in [40]. Our work is similar to these works as we are also modeling the learning of perceptual chunks, a kind of deep feature learning, but differs from these theories since none of the above theories uses pCFG learning to model the acquisition of perceptual chunks.

There has also been considerable research on learning within agent architectures. Soar [19] uses a chunking mechanism to acquire knowledge that constrains problem-space search. Another architecture ACT-R [2] creates new production rules through a compilation process that gradually transforms declarative representations into skill knowledge [18]. Icarus [3] acquires complex conceptual predicates in the context of problem solving. Unlike those theories, SimStudent puts more emphasis on knowledge-level learning (cf., [20]) achieved through induction from positive and negative examples. It integrates ideas of theories of perceptual chunking [31] as a basis for improving knowledge representations that, in turn, facilitate better learning of problem solving skills.

Other research in cognitive science also attempts to use probabilistic approaches to model the process of human learning. Kemp and Xu [41] apply a probabilistic model to capture principles of infant object perception. Kemp and Tenenbaum [42] use a hierarchical generative model to show the acquisition process of domain-specific structural constraints. But again, neither of the above approaches tend to use the probabilistic model as a representation acquisition component in a learning agent.

Another closely related research area is learning procedural knowledge by observing others' behavior. Classical approaches include explanation-based learning [43,44], learning apprentices [45] and programming by demonstration [46,16]. Most of these approaches used analytic methods to acquire candidate procedures. Other works on transfer learning (e.g., [47–50]) also share some resemblance with our work. They focus on improving the performance of learning by transferring previously acquired knowledge from another domain of interest. However, to the best of our knowledge, none of the above approaches uses the transfer learning learner to acquire a better representation that reveals essential percept features, and to integrate it into an intelligent agent.

Additionally, research on deep architectures [51] shares a clear resemblance with our work and has been receiving increasing attention recently. Theoretical results suggest that in order to learn complicated functions such as AI-level tasks, deep architectures that are composed of multiple levels of non-linear operation are needed. Although not having been studied much in the machine learning literature due to the difficulty in optimization, there are some notable exceptions in the area including convolutional neural networks [52–55], sigmoidal belief networks learned using variational approximations [56–59], and deep belief networks [60,61]. While both the work in deep architectures and our work are interested in modeling complicated functions through non-linear features, the tasks we work on are different. Deep architectures are used more often in classification tasks whereas our work focuses on simulating human problem solving and learning of math and science.

A lot of efforts have been put toward comparing the quality of alternative student models. LFA automatically discovers student models, but is limited to the space of the human-provided factors. Other works such as [62,63] are less dependent

on human labeling, but may suffer from challenges in interpreting the results. In contrast, the SimStudent approach has the benefit that the acquired production rules have a precise and usually straightforward interpretation. Baffes and Mooney [64] apply theory refinement to the problem of modeling incorrect student behavior. Other systems [65,66] use a Q-matrix to find knowledge structure from student response data. None of the above approaches use simulated students to construct student models.

Besides SimStudent, there has been a lot of work on creating simulated students [7,67,68]. VanLehn's [69] Sierra models the impasse-driven acquisition of hierarchical procedures for multi-column subtraction from sample solutions. However, his work focused on explaining the origin of bugs for real students, which is not the focus here. In addition, Sierra is given a CFG for parsing the visual state of the subtraction problems, whereas our system automatically acquires a pCFG. Ohlsson [70] reviews how different learning models are employed during different learning phases in intelligent systems. Our work on integrating representation learning and skill learning also reflects how one learning mechanism is able to aid other learning processes in an intelligent systems. None of the above approaches compared the system with human learning curve data. To the best of our knowledge, our work is the first combination of the two whereby we use student model evaluation techniques to assess the quality of a simulated learner.

9. Further discussion

In spite of the promising results, there are still many fruitful possibilities to further improve SimStudent. First of all, we should carry out more extensive studies in more domains to evaluate the generality of the proposed system. Moreover, since the extended SimStudent only uses domain-independent operator functions, we should also evaluate whether this extension enables better transfer learning. In other words, after trained on one relevant task, does the extended SimStudent needs fewer number of extra operator functions than the original SimStudent in the new learning task. We have shown experimental results being repeated in other domains, such as fraction addition, stoichiometry, and second language learning [12,21]. Basically, any domain that could be represented as a set of production rules can be potentially addressed by SimStudent. In future studies, we would like to see whether the similar results would be reproduced in more probabilistic-based domains. Since we are interested in modeling real student learning, we would also like to carry out more experiments comparing our system performance with real student data. In particular, we are interested in matching the type of errors made by SimStudent with the common errors made by real students. We believe that with these extensions, we would be able to gain more insights of human learning, as well as to advance the process of creating an integrated intelligent agent.

Second, as mentioned above, pCFGs are more suitable in representing 1-dimensional information, which is an appropriate choice for many maths domains such as algebra, multi-column addition and so on. There are other domains that may require representation in a 2-dimensional space. In response, we have extended the feature learning mechanism to use a two-dimensional variant of a probabilistic context-free grammar (pCFG) in modeling how a user perceives the structure of a user interface [71]. The proposed 2-dimensional pCFG learning algorithm models acquisition of this representation. Our learning method exploits both the spatial layout of the interface, and temporal information about when users interact with the interface. Since the additional temporal and spatial information was used in the representation acquisition procedure, the search space is reduced and thus the learning process remains relatively effective. More specifically, experimental studies were carried out in both synthetic domains and tutoring domains including fraction addition, equation solving, and stoichiometry. We show that the proposed layout learner is able to acquire the high-quality layouts with a small number of training examples, and demonstrate the intelligent agent with the acquired layouts is able to perform equally well comparing with the agent given manually constructed layouts. We believe that similar changes can be made to further extend the 2-dimensional grammar to support domains such as integral or differential equations.

Additionally, the deep feature learning process is currently carried out before the SimStudent knowledge acquisition. Only the acquired grammar is used to provide better representation to SimStudent. It is possible that the feedback given to SimStudent could also provide feedback to the integrated grammar. For example, if the deep feature learner initially acquired the incorrect grammar, and caused a lot of failures for SimStudent learning, the extended SimStudent could potentially feed this information back to the deep feature learner, and ask it to revise its grammar. Moreover, the training records for the deep feature learner could also be automatically generated from the steps demonstrated to SimStudent. By doing this, SimStudent would be able to learn better representation knowledge during skill knowledge acquisition. The two learning systems would mutually assist each other in achieving better performance.

Lastly, SimStudent's learning mechanism presents a bias towards more generalized skills over specific ones without considering computational cost. This bias appears to be a limitation of SimStudent as a model of student learning. In cases where a more general strategy invokes a more complicated procedure (like dividing both sides by *-1* in solving *-x=A*), human students may prefer to use a less general but simple strategy (such as moving the minus sign from left-hand side to right-hand side). We have recently developed a conflict resolution strategy which could be used to prefer skills of smaller computational cost [21]. This extension potentially addresses this limitation of SimStudent as a student model.

10. Concluding remarks

As also shown in previous studies [8,9], we claim that representation learning is one of the important aspects in human knowledge acquisition, and should be modeled in cognitive theories of human learning. In our theory, perceptual chunks

correspond to nodes in a context-free grammar that organizes perceptual knowledge. Perceptual processing is modeled by parsing stimuli using this grammar, with the result being a parse tree that organizes their constituents. The learning process of the representation is carried out in a bottom-up, unsupervised fashion guided by statistical regularities in observed stimuli. While results show that the acquired representation is able to model some types of student errors, other forms of representation such as discrimination networks [9,31,39] and connectionist models (e.g., [72]) are also possible.

We integrate the proposed representation learner into skill learning. We claim that the speed of skill acquisition depends on the quality of the representation given to the skill learner. By integrating representation learning into skill learning, the speed of skill learning increases. In later studies [13], we showed that the extended agent can be used to discover better models of student learning, which suggests that the extended agent better models human knowledge acquisition.

To sum up, building an intelligent agent that simulates human-level learning is an essential task in AI and cognitive science, but building such systems often requires manual encoding of prior domain knowledge. In this paper, we proposed a novel algorithm that automatically acquires deep features from observations without any annotation or with light annotations. We then integrate this stand-alone feature learning algorithm into an intelligent agent, SimStudent, as an extension of the perception module. We showed that after the integration, the extended SimStudent is able to achieve comparable performance without requiring any domain-specific operator function as input. In addition to being an effective learner, we further showed that the extended SimStudent could be used to discover better models of real students. Given all the results, we conclude that the extended SimStudent is a good human-like intelligent agent that requires a small amount of knowledge engineering.

Acknowledgement

This research was sponsored by the National Science Foundation under grant numbers SBE-0354420, DLR0910176, OMA0836012, and EIA0205301; the University of California (ONR) under grand number G0607ES008; the Department of Education under grant number R305A090519; and the Mitre Corporation under grant number 62459. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

References

[1] J.E. Laird, A. Newell, P.S. Rosenbloom, Soar: an architecture for general intelligence, Artif. Intell. 33 (1) (1987) 1–64.
[2] J.R. Anderson, Rules of the Mind, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1993.
[3] P. Langley, D. Choi, A unified cognitive architecture for physical agents, in: Proceedings of the Twenty-First National Conference on Artificial Intelligence, Boston, 2006.
[4] C.M. Neves, J.R. Anderson, Knowledge compilation: mechanisms for the automatization of cognitive skills, in: Cognitive Skills and Their Acquisition, 1981, pp. 57–84.
[5] Y. Anzai, H.A. Simon, The theory of learning by doing, Psychol. Rev. 86 (2) (1979) 124–140.
[6] N. Matsuda, A. Lee, W.W. Cohen, K.R. Koedinger, A computational model of how learner errors arise from weak prior knowledge, in: Proceedings of Conference of the Cognitive Science Society, 2009.
[7] K. Vanlehn, S. Ohlsson, R. Nason, Applications of simulated students: an exploration, J. Artif. Intell. Educ. 5 (1994) 135–175.
[8] M.T.H. Chi, P.J. Feltovich, R. Glaser, Categorization and representation of physics problems by experts and novices, Cogn. Sci. 5 (2) (1981) 121–152.
[9] W.G. Chase, H.A. Simon, Perception in chess, Cogn. Psychol. 4 (1) (1973) 55–81.
[10] N. Li, W.W. Cohen, K.R. Koedinger, A computational model of accelerated future learning through feature recognition, in: Proceedings of 10th International Conference on Intelligent Tutoring Systems, 2010, pp. 368–370.
[11] N. Matsuda, W.W. Cohen, K.R. Koedinger, Teaching the teacher: tutoring SimStudent leads to more effective cognitive tutor authoring, Int. J. Artif. Intell. Educ. (2014), http://dx.doi.org/10.1007/s40593-014-0020-1.
[12] N. Li, W.W. Cohen, K.R. Koedinger, Efficient cross-domain learning of complex skills, in: Proceedings of the 11th International Conference on Intelligent Tutoring Systems, 2012.
[13] N. Li, W.W. Cohen, N. Matsuda, K.R. Koedinger, A machine learning approach for automatic student model discovery, in: Proceedings of the 4th International Conference on Educational Data Mining, Eindhoven, 2011, pp. 31–40.
[14] N. Li, W.W. Cohen, K.R. Koedinger, Problem order implications for learning transfer, in: Proceedings of the 11th International Conference on Intelligent Tutoring Systems, 2012.
[15] V. Aleven, B.M. Mclaren, J. Sewall, K.R. Koedinger, A new paradigm for intelligent tutoring systems: example-tracing tutors, Int. J. Artif. Intell. Educ. 19 (2009) 105–154.
[16] T. Lau, D.S. Weld, Programming by demonstration: an inductive learning formulation, in: Proceedings of the 1999 International Conference on Intelligence User Interfaces, 1998, pp. 145–152.
[17] S. Muggleton, L. de Raedt, Inductive logic programming: theory and methods, J. Log. Program. 19 (1994) 629–679.
[18] N.A. Taatgen, F.J. Lee, Production compilation: a simple mechanism to model complex skill acquisition, Hum. Factors 45 (1) (2003) 61–75.
[19] J.E. Laird, P.S. Rosenbloom, A. Newell, Chunking in SOAR: the anatomy of a general learning mechanism, Mach. Learn. 1 (1986) 11–46.
[20] A. Newell, The knowledge level, Artif. Intell. 18 (1) (1982) 87–127.
[21] N. Li, Y. Tian, W.W. Cohen, K.R. Koedinger, Integrating perceptual learning with external world knowledge in a simulated student, in: 16th International Conference on Artificial Intelligence in Education, 2013, pp. 400–410.
[22] T. Mitchell, Generalization as search, Artif. Intell. 18 (2) (1982) 203–226.
[23] J.R. Quinlan, Learning logical definitions from relations, Mach. Learn. 5 (3) (1990) 239–266.
[24] K.R. Koedinger, J.R. Anderson, Abstract planning and perceptual chunks: elements of expertise in geometry, Cogn. Sci. 14 (1990) 511–550.
[25] N. Li, S. Kambhampati, S. Yoon, Learning probabilistic hierarchical task networks to capture user preferences, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, CA, 2009.
[26] K. Lari, S.J. Young, The estimation of stochastic context-free grammars using the inside–outside algorithm, Comput. Speech Lang. 4 (1990) 35–56.
[27] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, J. R. Stat. Soc. B 39 (1) (1977) 1–38.

[28] K.R. Koedinger, B.A. MacLaren, Developing a pedagogical domain theory of early algebra problem solving, CMU-HCII Tech. report 02-100, Carnegie Mellon University, 2002.
[29] N. Li, A. Schreiber, W.W. Cohen, K.R. Koedinger, Creating features from a learned grammar in a simulated student, in: Proceedings of the 20th European Conference on Artificial Intelligence, 2012.
[30] N. Li, W. Cushing, S. Kambhampati, S. Yoon, Learning probabilistic hierarchical task networks as probabilistic context-free grammars to capture user preferences, ACM Trans. Intell. Syst. Technol. 5 (2) (2014) 29.
[31] H.B. Richman, J.J. Staszewski, H.A. Simon, H.B. Richman, J.J. Staszewski, H.A. Simon, Simulation of expert memory using EPAM IV, Psychol. Rev. (1995) 305–330.
[32] A.R. Jansen, K. Marriott, G.W. Yelland, Constituent structure in mathematical expressions, in: Proceedings of the 22th Annual Meeting of the Cognitive Science Society, vol. 22, Mahwah, NJ, 2000, p. 238.
[33] K.R. Koedinger, A.T. Corbett, C. Perfetti, The knowledge-learning-instruction (KLI) framework: toward bridging the science–practice chasm to enhance robust student learning, Cogn. Sci. 36 (2012) 757–798.
[34] K.R. Koedinger, M.J. Nathan, The real story behind story problems: effects of representations on quantitative reasoning, J. Learn. Sci. 13 (2) (2004) 129–164.
[35] K.R. Koedinger, E.A. McLaughlin, Seeing language learning inside the math: cognitive analysis yields transfer, in: Proceedings of the 32nd Annual Conference of the Cognitive Science Society, Austin, TX, 2010, pp. 471–476.
[36] H. Cen, K. Koedinger, B. Junker, Learning factors analysis – a general method for cognitive model evaluation and improvement, in: Proceedings of the 8th International Conference on Intelligent Tutoring Systems, 2006, pp. 164–175.
[37] K.R. Koedinger, R.S. Baker, K. Cunningham, A. Skogsholm, B. Leber, J. Stamper, A data repository for the EDM community: the PSLC DataShop, in: Handbook of Educational Data Mining, 2010.
[38] K.R. Koedinger, E.A. McLaughlin, J.C. Stamper, Automated student model improvement, in: Proceedings of the 5th International Conference on Educational Data Mining, 2012, pp. 17–24.
[39] F. Gobet, H.A. Simon, Five seconds or sixty? Presentation time in expert memory, Cogn. Sci. 24 (4) (2000) 651–682.
[40] F. Gobet, Chunking models of expertise: implications for education, Appl. Cogn. Psychol. 19 (3) (2005) 183–204.
[41] C. Kemp, F. Xu, An ideal observer model of infant object perception, in: D. Koller, D. Schuurmans, Y. Bengio, L. Bottou (Eds.), NIPS, MIT Press, 2008, pp. 825–832.
[42] C. Kemp, J.B.B. Tenenbaum, The discovery of structural form, Proc. Nat. Acad. Sci. USA 105 (31) (2008) 10687–10692.
[43] A. Segre, A learning apprentice system for mechanical assembly, in: Proceedings of the Third IEEE Conference on AI for Applications, 1987, pp. 112–117.
[44] R.J. Mooney, A General Explanation-Based Learning Mechanism and Its Application to Narrative Understanding, Morgan Kaufmann, San Mateo, CA, 1990.
[45] T.M. Mitchell, S. Mahadevan, L.I. Steinberg, Leap: a learning apprentice for VLSI design, in: Proceedings of the 9th International Joint Conference on Artificial intelligence, San Francisco, CA, 1985, pp. 573–580.
[46] A. Cypher, D.C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, B.A. Myers, A. Turransky (Eds.), Watch What I Do: Programming by Demonstration, MIT Press, Cambridge, MA, 1993.
[47] R. Raina, A.Y. Ng, D. Koller, Constructing informative priors using transfer learning, in: Proceedings of the 23rd International Conference on Machine Learning, New York, NY, 2006, pp. 713–720.
[48] A. Niculescu-Mizil, R. Caruana, Inductive transfer for bayesian network structure learning, in: Proceedings of the 11th International Conference on AI and Statistics, 2007.
[49] L. Torrey, J. Shavlik, T. Walker, R. Maclin, Relational macros for transfer in reinforcement learning, in: Proceedings of the 17th Conference on Inductive Logic Programming, Corvallis, Oregon, 2007.
[50] M. Richardson, P. Domingos, Markov logic networks, Mach. Learn. 62 (1–2) (2006) 107–136.
[51] Y. Bengio, Learning deep architectures for AI, Found. Trends Mach. Learn. 2 (2009) 1–127.
[52] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural Comput. 1 (1989) 541–551.
[53] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, in: Proceedings of the IEEE, 1998, pp. 2278–2324.
[54] P.Y. Simard, D. Steinkraus, J.C. Platt, Best practices for convolutional neural networks applied to visual document analysis, in: Proceedings of the Seventh International Conference on Document Analysis and Recognition, IEEE Computer Society, Washington, DC, USA, 2003.
[55] M. Ranzato, F.J. Huang, Y.L. Boureau, Y. LeCun, Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition, Computer Vision and Pattern Recognition, IEEE Computer Society Conference on 5 (2007) 1–8.
[56] P. Dayan, G.E. Hinton, R.M. Neal, R.S. Zemel, The Helmholtz machine, Neural Comput. 7 (5) (1995) 889–904.
[57] G.E. Hinton, P. Dayan, B.J. Frey, R.M. Neal, The "wake–sleep" algorithm for unsupervised neural networks, Science 268 (5214) (1995) 1158–1161.
[58] L.K. Saul, T. Jaakkola, M.I. Jordan, Mean field theory for sigmoid belief networks, J. Artif. Intell. Res. 4 (1996) 61–76.
[59] I. Titov, J. Henderson, Constituent parsing with incremental sigmoid belief networks, in: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, Association for Computational Linguistics, Prague, Czech Republic, 2007, pp. 632–639.
[60] G.E. Hinton, To recognize shapes, first learn to generate images, Prog. Brain Res. 165 (2007) 535–547.
[61] Y. Bengio, O. Delalleau, C. Simard, Decision trees do not generalize to new variations, Comput. Intell. 26 (4) (2010) 449–467.
[62] P.I. Pavlik, H. Cen, K.R. Koedinger, Learning factors transfer analysis: using learning curve analysis to automatically generate domain models, in: Proceedings of 2nd International Conference on Educational Data Mining, 2009, pp. 121–130.
[63] M. Villano, Probabilistic student models: Bayesian belief networks and knowledge space theory, in: Proceedings of the 2nd International Conference on Intelligent Tutoring Systems, Heidelberg, 1992, pp. 491–498.
[64] P. Baffes, R. Mooney, Refinement-based student modeling and automated bug library construction, J. Artif. Intell. Educ. 7 (1) (1996) 75–116.
[65] K.K. Tatsuoka, Rule space: an approach for dealing with misconceptions based on item response theory, J. Educ. Meas. 20 (4) (1983) 345–354.
[66] T. Barnes, The Q-matrix method: mining student response data for knowledge, in: Proceedings AAAI Workshop Educational Data Mining, Pittsburgh, PA, 2005, pp. 1–8.
[67] T.-W. Chan, C.-Y. Chou, Exploring the design of computer supports for reciprocal tutoring, Int. J. Artif. Intell. Educ. 8 (1997) 1–29.
[68] T.N. Pentti Hietala, The competence of learning companion agents, Int. J. Artif. Intell. Educ. 9 (1998) 178–192.
[69] K. VanLehn, Mind Bugs: The Origins of Procedural Misconceptions, MIT Press, Cambridge, MA, USA, 1990.
[70] S. Ohlsson, Computational Models of Skill Acquisition, Cambridge University Press, 2008, pp. 359–395, Ch. 13.
[71] N. Li, W.W. Cohen, K.R. Koedinger, Learning to perceive two-dimensional displays using probabilistic grammars, in: Proceedings of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases – Part II, ECML PKDD'12, Berlin, Heidelberg, 2012, pp. 773–788.
[72] J.R. Anderson, A spreading activation theory of memory, J. Verbal Learn. Verbal Behav. 22 (1983) 261–295.