Integrating Representation Learning and Skill Learning in a Human-Like Intelligent Agent

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

January 2012
CMU-ML-12-100

Carnegie Mellon

Integrating Representation Learning and Skill Learning in a Human-Like Intelligent Agent

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

January 2012
CMU-ML-12-100

Carnegie Mellon

Integrating Representation Learning and Skill Learning in a Human-Like Intelligent Agent

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

January 2012
CMU-ML-12-100

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Keywords: representation learning, deep feature learning, intelligent agent

Abstract

Building an intelligent agent that simulates human learning of math and science could potentially benefit both education, 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 work showed 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 many existing learning algorithms is sensitive to representation, the deep features are also important in achieving effective learning. In this paper, we present an efficient algorithm that acquires representation knowledge in the form of "deep features" for specific domains, 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 [21, 2, 23] has been put toward this challenging task. Further, education in the 21^{st} 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.

To contribute to both goals, recent efforts [30] have been pursued to develop 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, the agent developer needs to encode functions that describe how to extract a coefficient, or how to add two terms, as prior knowledge given to the intelligent agent. Such manual encoding of prior knowledge can be time-consuming and error-prone.

Moreover, since real students entering a course do not necessarily 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 [10] 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 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 [28]. We reported the effectiveness of the algorithm in the algebra domain and synthetic domains.

In order to further evaluate how the deep feature learner could affect the performance of an intelligent agent, in this paper, we integrated this deep feature learning algorithm into a machine-learning agent, *SimStudent* [30], to provide a better representation. 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. 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. Fur-

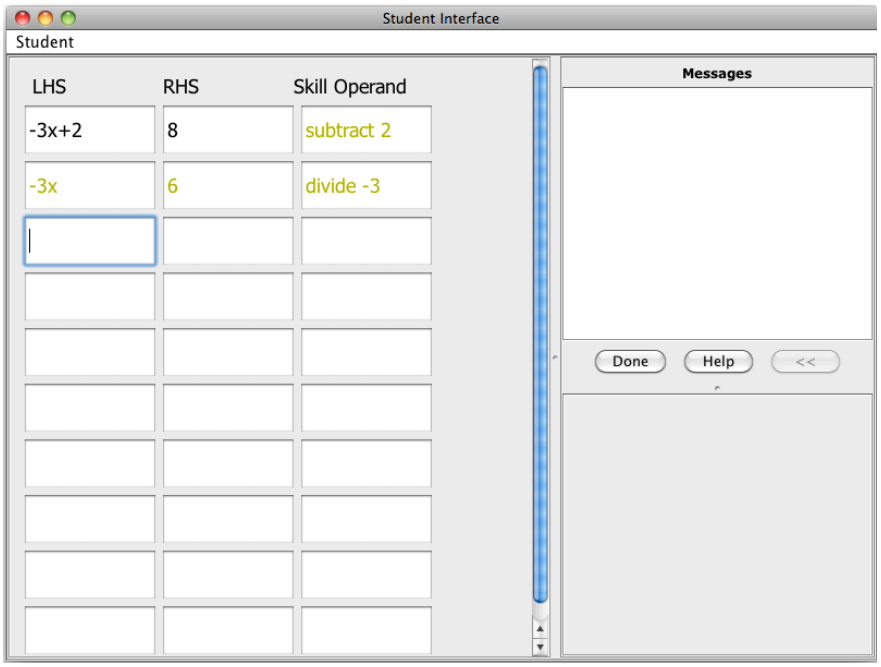


Figure 1: The interface where SimStudent is being tutored in an equation solving domain.

thermore, we also show that even compared to the original SimStudent with the domain-specific knowledge, the extended SimStudent is able to achieve nearly as good results without domain-specific knowledge given reasonable number of training examples. In addition, we use the extended SimStudent to automatically discover models of real students, and show that the discovered model is a better student model than human-generated models, as well as the model discovered by the original SimStudent.

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 modeling 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 feature learner into SimStudent, and illustrate the algorithm with an example in algebra. After that, we present experimental results for both the original SimStudent 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. Then, we present how to use the extended SimStudent to automatically discover student models, and show that the discovered model is better than the human-generated models. Finally, we discuss related work and possible future extensions.

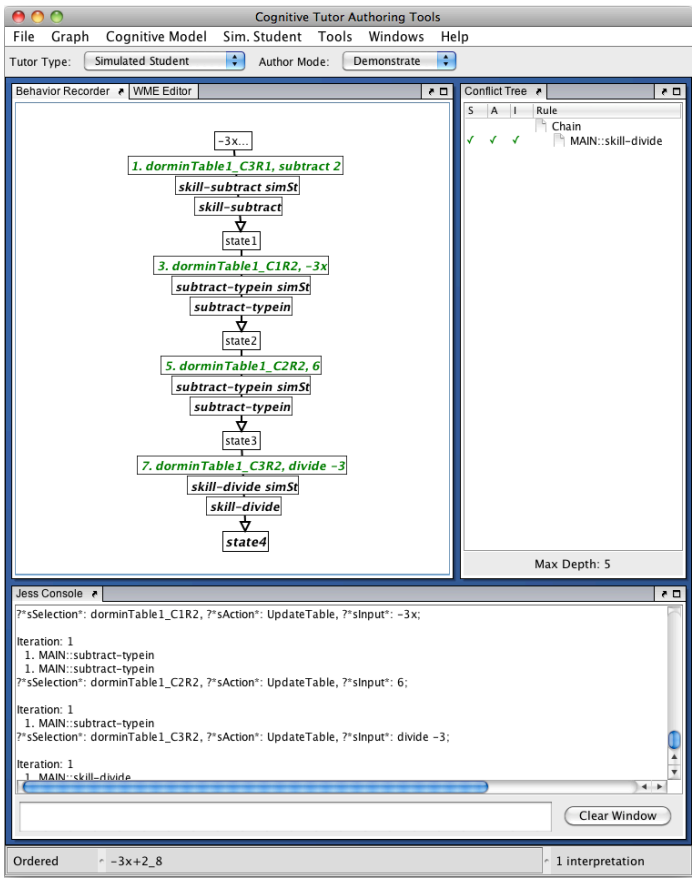


Figure 2: The interface that shows how SimStudent traces each demonstrated step and learns production rules.

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 [25] using inductive logic programming [34] as an underlying learning technique. Figure 1 and 2 are a screenshot of SimStudent learning to solve algebra equations. Figure 1 is the interface used to teach SimStudent, and Figure 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. But we would like to point out that 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 [30].

2.1 Learning Task

SimStudent is given a set of (ideally simple) *feature predicates* and a set of *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 skills (e.g., add two numbers, get the coefficient) that SimStudent can apply to the problem. Operator functions are divided into two groups, domain-independent operator functions and domain-specific operator functions. Domain-independent operator functions are basic skills used across multiple domains. Examples of such operator functions include adding two numbers, *(add 1 2)*, copy a string, *(copy -3x)*, and so on. These operator functions are not only used in solving equations, but also used in other domains such as multi-column addition and fraction addition. Hence, we assume that real students usually have knowledge of these simple skills prior to class. Domain-specific operator functions, on the other hand, are more complicated skills, such as getting the coefficient of a term, *(coefficient -3x)*, add two terms, *(add-term 5x-5 5)*, and so on. Performing such operator functions usually require domain expert knowledge, which real students may not.

Domain-specific operator functions require more knowledge engineering than domain-independent operator functions. For example, consider the "add" domain-independent operator function with the "add-term" domain-specific operator function. Adding two numbers is actually just one step among all the 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). Also note that since real students do not necessarily know when to apply these skills, unlike traditional planning problems, no explicit encoding of preconditions and effects are provided in the operator functions. These functions are not guaranteed to produce correct results. Thus, SimStudent is different from traditional planning algorithms, which can engage on speed up learning. SimStudent engages in knowledge level learning [35], 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 and a correct next step demonstration are provided to SimStudent. The learning agent will attempt to learn or modify its production rules accordingly. If it has not learned enough skill knowledge and fails to find a plan, a correct next step is directly demonstrated to SimStudent for later learning. Although other feedback mechanisms are also possible, in our case, the feedback is given by an existing automatic tutor CTAT [1], 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 graphical user interfaces (GUI) showing where to pay attention when plans for 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*). Note that unlike normal learning algorithm which gives the full plan (e.g., *(coefficient -3x ?coef) (divide ?coef)*) in getting the next step, the tutor only demonstrates the final output (e.g., (divide -3)) to SimStudent. This simulates the same amount of information available to real students. Taken together, the three pieces of information form one *example action record* indexed by the skill label,

R=⟨*label,* ⟨*percepts, step*⟩⟩. In the algebra example, an example action record is *R*=⟨*divide,* ⟨*(-3x, 6), (divide -3)*⟩⟩. For each incorrect next step proposed by SimStudent, an example action record is also generated as a negative example. During learning, SimStudent acquires one production rule for each skill label, l, based on the set of associated example action records gathered till the current step, $\mathcal{R}_l = (R_1, R_2, ..., R_n)$ (where $R_i.label = l$).

In summary, since we would like to model how real students are tutored, the learning task presented to SimStudent is quite challenging. First, the total number of world states is very large. When being taught, students are allowed to write whatever they consider to be correct into their solutions. Hence, it is actually legal to input any number/string to the interface textboxes, even if they are incorrect answers. This assumption makes the search space to SimStudent to be very large. Second, the operator functions given as prior knowledge do not encode any preconditions (for both applicability and search control) and postconditions in it. Last, the demonstrated steps are only partially observable. It usually takes more than one operator functions to move from one observed state to the next observed state. Some intermediate states and actions taken by the tutor are unobservable to SimStudent. Taken together, the learning task SimStudent is facing is learning skill knowledge within infinite world states given incomplete operator function description and partially observable states.

2.2 Production Rules

As we mentioned before, the output of the learning agent is represented as production rules. The left side of Figure 3 shows an example of a production rule learned by SimStudent with its readable format at the right side. A production rule indicates "where" to look for information in the interface (perceptual information), "how" to change the problem state (operator function sequence), and "when" to apply a rule (precondition for rule to be useful). For example, the rule to "divide both sides of *-3x=6* by *-3*" shown in Figure 3 would be read as "given a left-hand side (i.e., *-3x*) and a right-hand side (6) of the equation, when the left-hand side does not have a constant term, then get the coefficient of the term on the left-hand side and divide both sides by the coefficient." The perceptual information part represents paths to identify useful information from GUI in solving the problem. The precondition of a production rule includes a set of feature tests representing desired conditions in which to apply the production rule. The last part is the operator function sequence which specifies a plan to execute in the next step.

2.3 Learning Mechanism

With all the challenges presented, we have developed three learning mechanisms in SimStudent to acquire the three parts of the production rules. 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. All of the elements in the interface are organized in a tree structure. 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 left-hand side and right-hand side 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

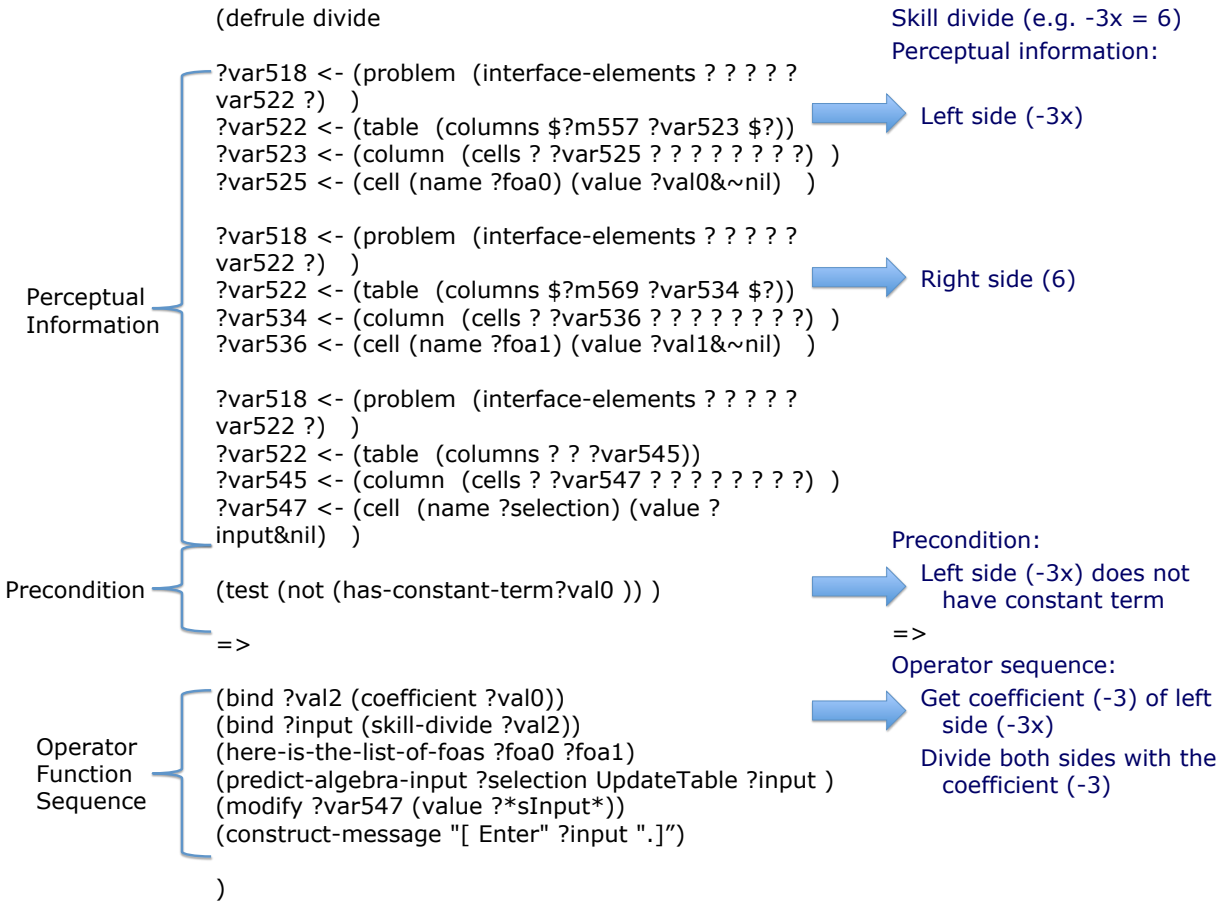


Figure 3: A production rule for divide.

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 includes the current node's position. A generalized path has one or more levels in the tree that are bound to more than one nodes. 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 ??*.

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 a m-dimensional vector, i.e., R_i.*percepts*= (*percepts*$_{i1}$, *percepts*$_{i2}$ ... *percepts*$_{im}$), where *percepts*$_{ij}$ stands for the j^{th} percept in the i^{th} 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 at the same position in all of the example action records. For each position j, the set of paths, where each

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

path can reach all percepts in $\mathcal{P}_j$, defines a version space $\mathcal{V}_j$ [31] (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.

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 [39], 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. 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 initial state, and sets the step, R_i*.step*, as the goal state. 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 an initial state and yields *step*$_i$ after applying the operator functions. For example, with the percepts-step pair in the example, $\langle$*(-3x, 6), (divide -3)*$\rangle$, the operator function sequence *(coefficient -3x ?coef) (divide ?coef)* is a possible explanation for this 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 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.

Last, although we said that SimStudent tries to learn one rule for each label, when some new training action record is added, it might fail to learn a single rule for all example action records (e.g., no operator function sequence is found that explains all percepts-step pairs including the new one). In that case, SimStudent learns a separate rule just for the last example action record. This breaking a single production rule into a pair of disjuncts effectively splits the example action records into two clusters. Later, for each new example action record, SimStudent tries to acquire a rule for each of the clusters plus the new example action record. If it cannot learn a rule that includes the new example, it creates another new cluster.

Table 1: Probabilistic context free grammar for coefficient in algebra

Terminal symbols: $-, x$;
Non-terminal symbols: $Expression, SignedNumber,$
$Variable, MinusSign, Number$;
$Expression \rightarrow 1.0,\ [SignedNumber]\ Variable$
$Variable \rightarrow 1.0,\ x$
$SignedNumber \rightarrow 0.5,\ MinusSign\ Number$
$SignedNumber \rightarrow 0.5,\ Number$
$MinusSign \rightarrow 1.0,\ -$

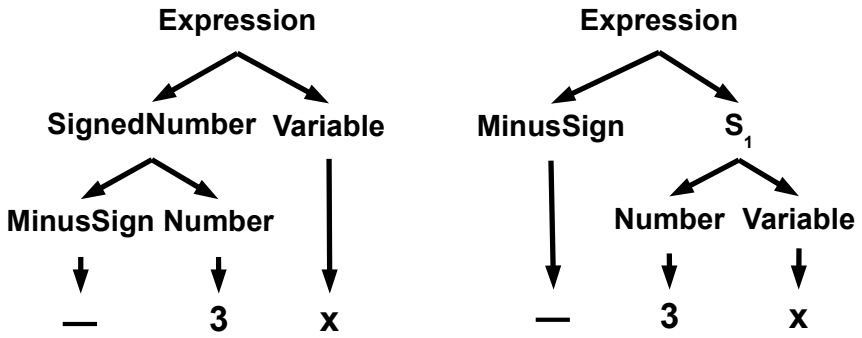


Figure 4: Correct and incorrect parse trees for $-3x$.

3 Deep Feature Learning

Having reviewed SimStudent, we move to a discussion of representation knowledge acquisition as deep feature learning. In this section, we are going to describe the deep feature learning algorithm. As mentioned above, deep feature learning is important both for human knowledge acquisition, and in achieving effective machine learning. Missing deep feature knowledge sometimes causes real students to make errors in learning. We carefully examined the nature of deep feature learning in algebra equation solving, and discovered that it could be modeled as a grammar induction problem given observational data (e.g. equations in algebra). Expressions can be formulated as a context free grammar as shown in Table 1. The deep feature "coefficient" is a non-terminal symbol in one of the grammar rules.

More interestingly, the perspective of viewing feature learning tasks as grammar induction problems also explains the cause of student errors. If we consider the learning task as a grammar induction problem, learning errors means acquiring the wrong grammar for the problem. Let us use the *-3x* example again. If instead of learning the grammar shown in Table 1, a student acquires another set of grammar rules. Based on the wrong grammar, the student constructs an incorrect parse tree as demonstrated at the right side of Figure 4. Instead of grouping - and *3* together, the

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
```

student groups *3* and *x* first, and thus mistakenly considers *3* as the coefficient. Based on these observations, we built a deep feature learner by extending an existing probabilistic context free grammar (PCFG) learner [29] to support feature learning and transfer learning. Note that the deep feature learner is domain general. It currently supports all domains where student input can be represented as a string of tokens, and can be modeled with a context-free grammar (e.g., algebra, stoichiometry, NLP).

3.1 A Brief Review of the pHTN Learner

Before introducing the deep feature acquisition algorithm, let's first briefly review the pHTN learner [29] it is based on. The pHTN learner is a variant of the inside-outside algorithm [24] that acquires a probabilistic context-free grammar (PCFG). The input to the pHTN learner is a set of observation sequences, $\mathcal{O}$. Each sequence is a string of characters 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, 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 context-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 trees *T* associated with each observation sequence as hidden variables. Each iteration involves two steps.

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 g. 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 it getting 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. (This learning procedure is a fast approximation of expectation-maximization [13], 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.

3.2 Feature Learning

Having reviewed Li et al.'s pHTN learning algorithm, we are ready to describe how it is extended to support deep feature learning without SimStudent. The input of the system is a set of pairs such as ⟨*-3x, -3*⟩, where the first element is the input to a feature extraction mechanism (e.g., coefficient), 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 in Table 1. To produce this output, the deep feature learner uses the pHTN 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 Figure 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 Figure 4 is an incorrect parse of *-3*, and there is no non-terminal symbol associated with *-3*. In this case, the system will ignore the current sequence. 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, if we assume that the tutor explicitly highlights example subsequences as targeted features (e.g. highlighting *-3* in *-3x* or *-4* in *-4x*), 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 subgrammar 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, 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

As mentioned above, aiding transfer learning by providing hints for common sub-grammars requires extra work for the tutors. 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. 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. When faced with a new task, the learning algorithm first 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 will first tries to build a parse tree for the whole term (e.g. *-3x*). But it fails, the grammar for signed number can only build the parse trees for some subsequence (e.g. *-3* in *-3x*). Then, the grammar learner gets 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.

Having acquired the grammar for deep features, when a new problem is given to the system, the learner will extract the deep feature by first building the parse tree of the problem based on the acquired grammar, and then extracting the subsequence associated with the feature symbol from the parse tree as the target feature. However, this model is only capable of learning and extracting deep features without using them to solve problems. Later, we will describe how to extend its ability by integrating it into SimStudent.

4 Empirical Evaluation on Deep Feature Learner

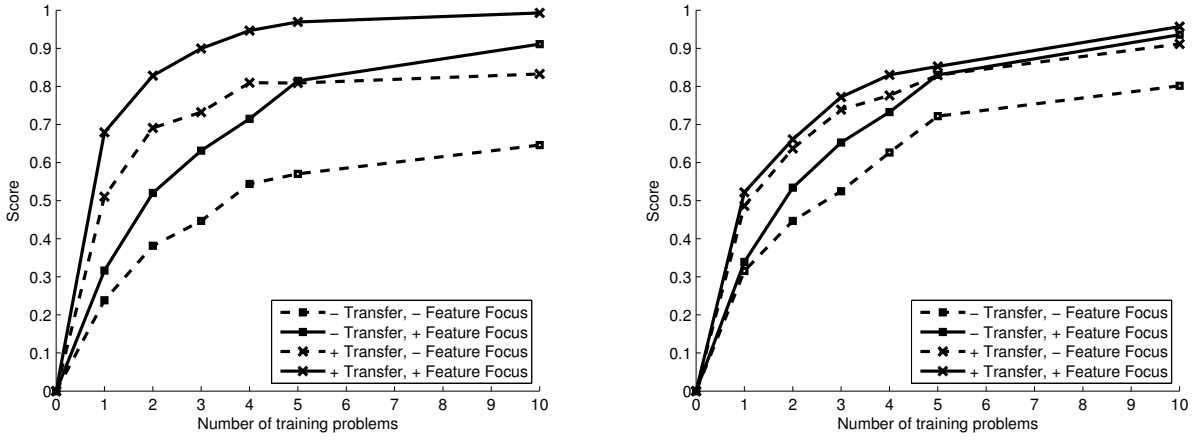
To evaluate the proposed deep feature learner, we carried out two controlled experiments. We compared four alternatives of the proposed approach, 1) without transfer learning and no feature focus; 2) without transfer learning, but with feature focus; 3) with transfer learning (from unlabeled sub-grammars), 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. have shown that the base learner (i.e. the learner with no extension) outperforms the inside-outside algorithm.[2] All the experiments were run on a 2.53 GHz Core 2 Duo Mackbook with 4GB 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, where randomly generated rules 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 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 previous grammar is not part of the current grammar. The second transfer is an 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.

Both transfer learners from unlabeled sub-grammars 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 $10n_1$. For each testing record, we compared the feature recognized by the oracle grammar with those recognized by the acquired grammar. The score is the average accuracy of 100 randomly generated feature extraction tasks.

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



(a) (b)

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

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 schemas 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 $i = 1, 2, ...100$. 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 Figure 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 (-*Transfer* -*Feature Focus*) (i.e. the learner without transfer learner and feature focus), 0.65. Learners with 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*).

One interesting thing you may have noticed is that the learner with only transferring from unlabeled sub-grammar has a better score (+*Transfer* -*Feature Focus*) with small number of training records comparing with the learner with only feature focus (-*Transfer* +*Feature Focus*). But with ten training records, the learner with only feature focus (-*Transfer* +*Feature Focus*) does have a higher score comparing with the learner with transfer learning (+*Transfer* -*Feature Focus*). This situation appears in both transfer tasks. It suggests that learners with transferring from unlabeled sub-grammars (+*Transfer* -*Feature Focus*) work better than those with feature focus (-*Transfer* +*Feature Focus*) with small number of training records, while learners with transferring from labeled sub-grammars (+*Transfer* -*Feature Focus*) perform better with large number of training

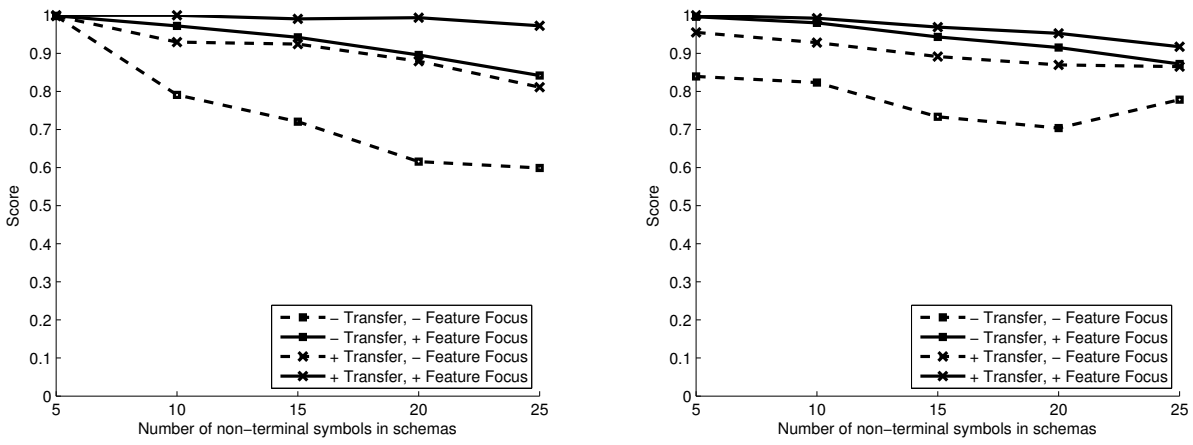
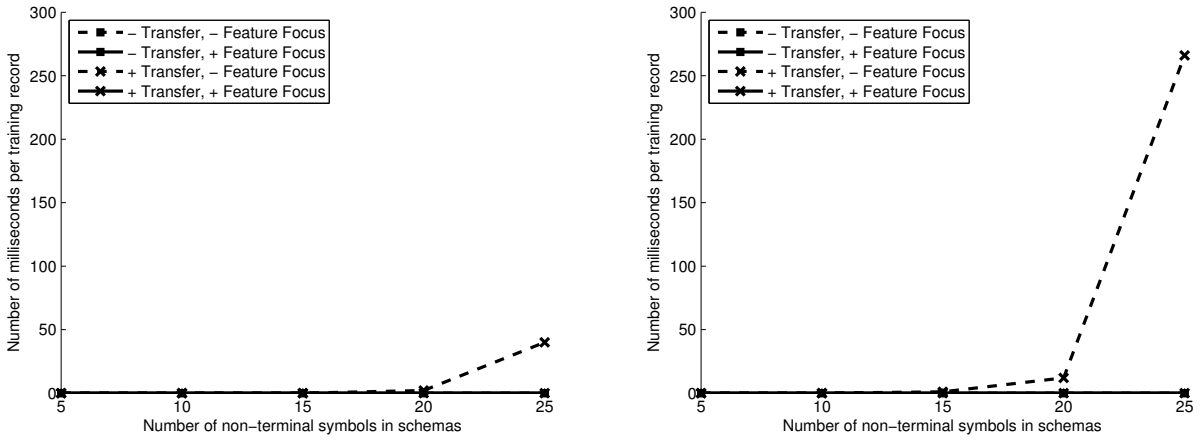


Figure 6: Score with different domain sizes with (a) subtask to task transfer problems. (b) overlapping tasks transfer problems.

records.

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 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 does not show up well. But even under this situation, 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 Figure 6(a) and 6(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 a fastest drop with increasing size of domains. The two learners with single extension (*-Transfer +Feature Focus*, and *+Transfer -Feature Focus*) perform roughly equally well. In fact, we have also tested the two learners with feature focus (*-Transfer +Feature Focus*, and *+Transfer +Feature Focus*) with domains of size 50. The learner with transfer learning as well as feature focus (*+Transfer +Feature Focus*) is able to perform quite well and get the score 0.90 for sub-grammar transfer and score 0.73 for overlapping sub-grammar transfer. We did not test the learner with only transfer learning (*+Transfer -Feature Focus*) since it took a longer time to run the experiment. More details can be found in the next paragraph.



(a) (b)

Figure 7: Average time spent on each training record with different domain sizes with (a) subtask to task transfer problems. (b) overlapping tasks transfer problems.

As for the average time spent on each training record, all learners acquire the targeted feature within a reasonable amount of time. The results are presented in Figure 7(a) and 7(b). While the learner with transfer learning (+*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 learner with only transfer learning (+*Transfer* -*Feature Focus*) does not consider feature focus during learning, it takes much longer time than other learners. Comparing with the base learner (-*Transfer* -*Feature Focus*), the learner with transfer learning (+*Transfer* -*Feature Focus*) contains not only knowledge of the current task, but also knowledge from previous tasks. Hence, it (+*Transfer* -*Feature Focus*) needs more learning time than the base learner. Regarding to learners with feature focus, even with domains of size 50, the learner with both extensions (+*Transfer* +*Feature Focus*) can still acquire schemas with an average of less than 1 millisecond per training record during learning.

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*), need to also maintain knowledge acquired from 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. Accelerated future learning, in which learning proceeds more effectively and more rapidly because of prior learning, is considered to be one of the most interesting measures of robust learning. It is considered that there are two reasons that could yield accelerated future learning: a better learning strategy and stronger prior knowledge. Learning with feature focus is considered to be a better learning strategy during knowledge acquisition. Since transfer learning from unlabeled sub-grammars maintains knowledge from previous task, it simulates the situation where a student has stronger prior knowledge due to previous training experience. 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.

4.3.1 Method

In order to understand the behavior of the proposed model, we designed three curricula. Three tasks are used across the three curricula. Task one is to learn signed number. 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 following the curriculum. For the last task, each learner was given one to five training records.

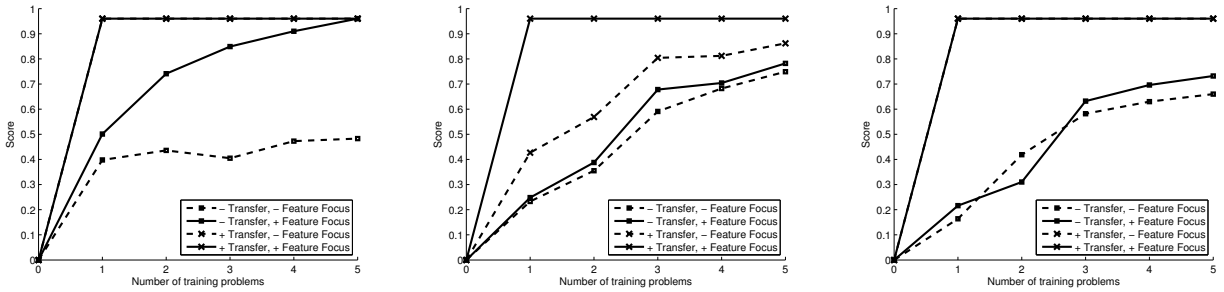
To measure learning gain, under each training condition, both systems were tested on 100 expressions in the same form of the training data in the last task. For each testing record, we compared the feature extracted by the oracle grammars with that recognized by the acquired grammars. Note that in task 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 the method is shown in Table 2.

4.3.2 Measurements

To assess the learning outcome, we measured the learning rate to evaluate the effectiveness of the learners. The experiment tested whether the proposed model is able to yield accelerated future learning, and how different extensions affect the learning rate. To evaluate the learning rate, we reported learning curves for all four learners by the number of training problems given in the last

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 $\rightarrow$ T2
	T2 $\rightarrow$ T3
	T1 $\rightarrow$ T2 $\rightarrow$ T3
Number of training condition:	10
Training size in all but last tasks:	10
Training size in the last task:	1, 2, 3, 4, 5
Testing size:	100



(a) (b) (c)

Figure 8: Learning curves for four learners in curriculum (a) from task one to task two (b) from task two to task three (c) from task one to task two to task three.

task. For each training size, we computed the average accuracy of the feature extraction task across 10 training conditions.

This experiment focuses on measuring the learning rate. In later chapters, we will also test whether the proposed model fits with real student data. We will show that after integrating the proposed model into a simulated student, the extended simulated student can be used to automatically discovers student models. The discovered model fits with real student data better than human-generated models. This indicates that the extended simulated student simulates the real student learning process well.

4.4 Impact of Accelerated Future Learning on the Rate of Learning

As shown in Figure 8(a), with curriculum one, all four learners acquired better knowledge with more training examples. With five training problems, all learners but the base learner were all able to acquire knowledge of score 0.96, and the base learner was able to achieve a score around 0.5.

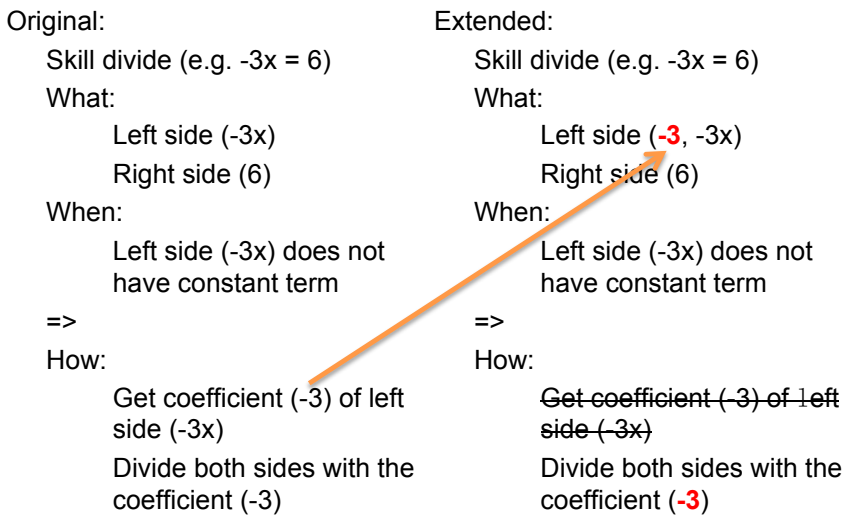


Figure 9: Original and extended production rules for divide in a readable format.

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 learner with a better learning strategy's (-*Transfer* +*Feature Focus*) learning curve is not as steep as the learners with transfer learning (+*Transfer* -*Feature Focus*, and +*Transfer* +*Feature Focus*), but is able to get score 0.96 with five training examples. This suggests that with transfer learning, learners are able to acquire knowledge quicker than those without transfer learning. The base learner's (-*Transfer* -*Feature Focus*) learning curve is the least steep one. A careful inspection shows that without feature focus and transfer learning, the base learner (-*Transfer* -*Feature Focus*) was not able to acquire grammar that contains one symbol associated with the feature "coefficient". 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 yields a steeper learning curve.

Similar results were also observed with curriculum two and curriculum three. In curriculum two, one interesting point is that, 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.

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

5 Integrating Deep Feature 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 better evaluate how the deep feature learner could affect the performance of an intelligent agent, in this section, we are going to present how to integrate deep feature learning into an intelligent agent, SimStudent. 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 them. On the other hand, 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 feature learner into SimStudent. Figure 9 shows a comparison between production rules acquired by the original and the extended SimStudents. As we can see 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, *(coefficient -3x)*. To achieve this, we extended both the perceptual learning algorithm and the precondition acquisition mechanism, as described below.

5.1 Extending Perceptual Learning

Previously, the perceptual information encoded in production rules is always associated with elements in the GUI such as cells in the algebra example. This assumption limits the granularity of observation SimStudent could achieve. In fact, the deep features we have discussed previously are perceptual information obtained at a much more fine-grained level. Failing to represent these deep perceptual features will harm the performance of the learning agent, and thus developers need to manually encode domain-specific operator functions to reduce the learning task's complexity.

To improve perceptual representation, we extend the percept hierarchy to further include the most probable parse tree for the content in the leaf nodes by appending the parse trees to their associated leaf nodes. All of the inserted nodes are of type "subcell". In the algebra example, this extension means that for cells that represent expressions corresponding to left-hand sides or right-hand 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 the left side of Figure 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 Figure 9, the new production rule would not need the first domain-specific operator function "coefficient" any more.

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 tutor 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 capabil-

ity 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 Figure 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*.

5.2 Extending Precondition Acquisition

In addition to extending the feature learner, we also extended the vocabulary of predicate symbols provided to the precondition learner. As implied by its name, the deep feature learner acquires information that reveals essential features of the problem state. It is natural to think that these deep features could also be used for describing desired situations to fire a production rule. Therefore, we construct a set of grammar features that are associated with the acquired PCFG. The set of new predicates describes positions of a subcell in the parse tree. For example, we create a new predicate called "is-left-child-of", which should be true for *(is-left-child-of -3 -3x)* based on the parse tree shown in the left side of Figure 4.

To make use of such predicates, we add them into the set of feature predicates for the precondition learner. When calling FOIL, the precondition learner evaluates the truthfulness of the extended set of predicates given all permutations of percept values. Since we have added relevant subcells

into percept lists, these permutations would include tuples such as *(-3, -3x)* for the grammar predicates such as "is-left-child-of". Thus, FOIL will be able to make use of this extra information to acquire better preconditions. Although not shown in the example, SimStudent may construct rules saying that when the left hand side (*-3x*) has a left child (*-3*) that is a constant, SimStudent should divide both sides with that left child.

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.

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, we select four problem sets that were used to teach real students as training sets. More specifically, the problem sets are from the same study of 71 high school students who used Carnegie Learning Algebra I Tutor. The sizes of the training sets are 13, 14, 35 and 35. We also choose 10 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 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, and also uses the set of grammar features for precondition learning (*E+Weak Ops*).

One interesting future study is to test the effectiveness of the grammar features. We did not carry out this experiment in the current study. In future studies, we would like to remove some of the original predicates, and compare the performance of the extended SimStudent with and without the grammar features. Previous studies [30] have also shown that the SimStudent given weak prior knowledge (i.e. the weak operator function set) often acquires incorrect production

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	(revserse-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*

Table 3: A list of operator functions used by SimStudents.

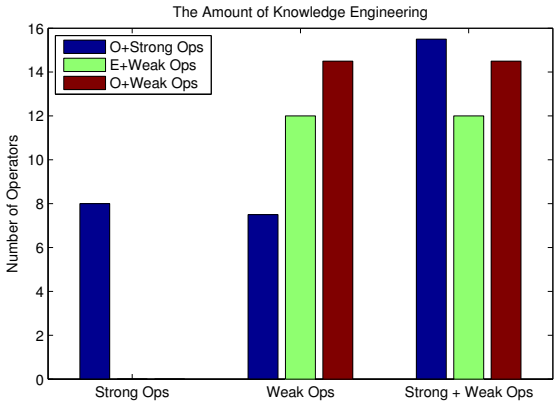


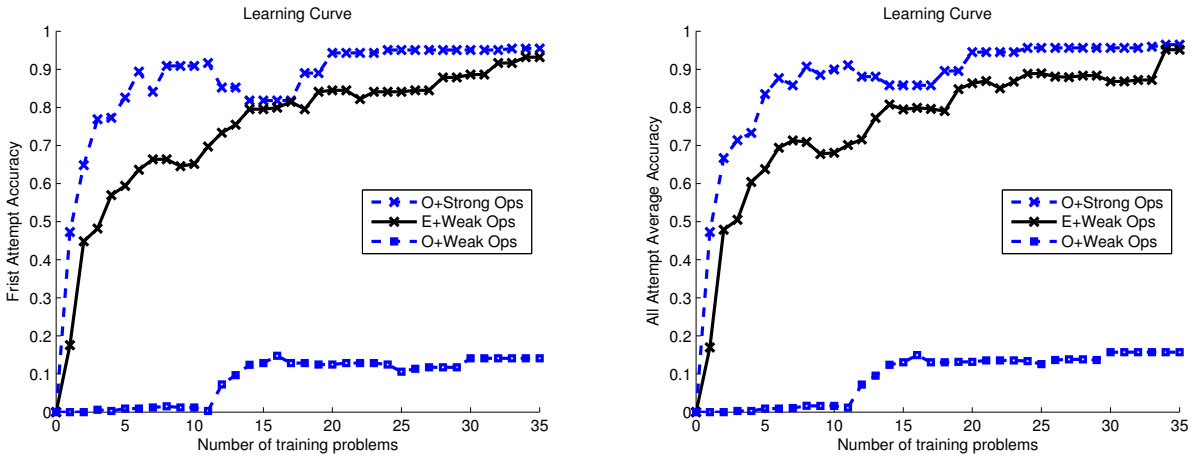
Figure 10: Number of strong and weak operator functions used in acquired production rules.

rules that produce the same errors the human students commonly mades. We do not present these results in this paper.

6.2 Experiment Results

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 Figure 10, only the SimStudent (*O+Strong Ops*) who was given strong operator functions used 8 of the domain-specific operator functions, plus 7.5 domain-general operator functions on average in the production rules. In contrast, the extended SimStudent (*E+Weak Ops*) 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 will see later, it performs much worse than the extended SimStudents.

Evaluation of Learning Speed: The second 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



(a) (b)

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

score as the all-attempt average accuracy.

The average learning curves for the three SimStudents are shown in Figures 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 95%, which is only 1% 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 knowledge engineering.

7 Using SimStudent to Discover Better Student 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 student models using the extended SimStudent. If the discovered model turns out to be a good student model, we should be able to conclude that the extended SimStudent simulates the real student learning process well. A student model is a set of *knowledge components (KC)* encoded in intelligent tutors to model how students solve problems. 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*). It provides important information to automated tutoring systems in making instructional decisions. Better student models match with real student behavior. They are capable of predicting task difficulty and transfer of learning between related problems, and often yield better instruction.

Traditional ways to construct student 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 [20, 19] 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) [8] apply an automated search technique to discover student models. It has been shown that these automated methods are able to find better student 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.

To address this issue, we propose a method that automatically discovers student 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 two models, and then testing how well the two model codings fit with real student data. Note that the human-generated KC model is one of the best models among 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. As a result, there were nine 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 can be 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 a Carnegie Learning Algebra I Tutor, like a human student. We selected 40 problems that were used to teach real students as the training set for SimStudent. 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 is considered as an "opportunity" for that student to show mastery of that KC.

Having finished coding real student steps with both models (the SimStudent model and the human-generated model), we used the Additive Factor Model (AFM) [8] 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,

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

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.

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;

We utilized DataShop [18], 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.

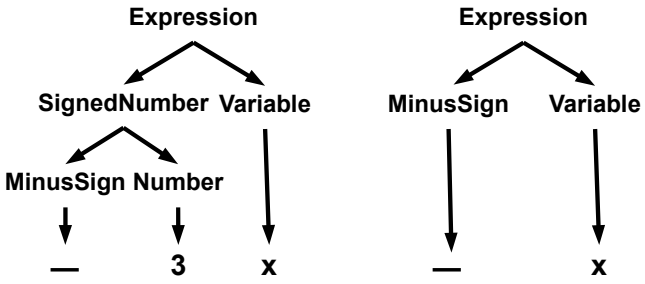


Figure 12: Different parse trees for -3x and -x.

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 test whether the generated model fits with real student data, we used AIC and a 3-fold cross validation. AIC measures the fit to student data while penalizing over-fitting. We did not use BIC (Bayesian Information Criterion) as the fit metric, because based on past analysis across multiple DataShop datasets, it has been shown that AIC is a better predictor of cross validation than BIC is. 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 also 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.

In order to understand whether the differences are significant 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 a better

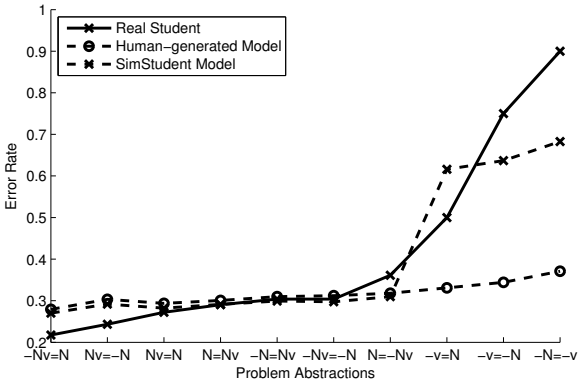


Figure 13: Error rates for real students and predicted error rates from two student models.

prediction than the human-generated KC model in 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. [3] Therefore, we conclude that the SimStudent model is a better student model than the human-generated KC model.

7.5 Implications on 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. 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 expert 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 expert 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 Figure 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

[3] 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.

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 relevant set of problem types – these are shown with the solid line in Figure 13 with the problem types defined in forms like *-Nv=N*, where the *N*s are any integrate 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*).

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 Figure 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 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*.

8 Related Work

The main contribution of this paper is to reduce the amount of knowledge engineering required in building a human-like intelligent agent by integrating feature learning into an agent. There has been considerable research on learning within agent architectures. Soar [22] 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 [46]. ICARUS [23] acquires complex conceptual predicates in the context of problem solving. PRODIGY uses an analytical technique to acquire control rules for means-ends problem solving (Minton et al., 1989).

Another closely related research area is learning procedural knowledge by observing others' behavior. Classical approaches include explanation-based learning [44, 33], learning apprentices [32] and programming by demonstration [11, 25]. Most of these approaches used analytic methods to acquire candidate procedures. Other works on transfer learning, e.g., [40, 36, 49, 42], 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 integrate it into an intelligent agent. Other research in cognitive science also attempt to use probabilistic approaches to model

the process of human learning. Kemp and Xu [17] apply a probabilistic model to capture principles of infant object perception. Kemp and Tenenbaum [16] 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.

Additionally, research on deep architectures [5] 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 [26, 27, 45, 41], sigmoidal belief networks learned using variational approximations [12, 15, 43, 48], and deep belief networks [14, 6]. 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 learning of math and science.

There has been considerable work on 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 [37, 52] 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 [3] apply theory refinement to the problem of modeling incorrect student behavior. Other systems [47, 4] 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 [51, 9, 38]. VanLehn [50] created a learning system and evaluated whether it was able to learn procedural "bugs" like real students. Biswas et al.'s [7] system learns causal relations from a conceptual map created by students. 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 Future Work

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. 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-D 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-D space. We believe the grammar learner can also be extended to support other domains that require 2-D representation. The geometry domain would be an example of such. We would like to further extend our feature learner to support 2-D domains by using bidirectional grammars.

Finally, 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.

10 Concluding Remarks

To sum up, building an intelligent agent that simulates human-level learning is an essential task in AI and education, 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.

References

[1] Vincent Aleven, Bruce M. Mclaren, Jonathan Sewall, and Kenneth R. Koedinger. A new paradigm for intelligent tutoring systems: Example-tracing tutors. *International Journal of Artificial Intelligence in Education*, 19:105–154, April 2009.

[2] John R. Anderson. *Rules of the Mind*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1993.

[3] Paul Baffes and Raymond Mooney. Refinement-based student modeling and automated bug library construction. *J. Artif. Intell. Educ.*, 7(1):75–116, 1996.

[4] Tiffany Barnes. The Q-matrix method: Mining student response data for knowledge. In *Proceedings AAAI Workshop Educational Data Mining*, pages 1–8, Pittsburgh, PA, 2005.

[5] Yoshua Bengio. Learning deep architectures for ai. *Foundations Trends in Machine Learning*, 2:1–127, January 2009.

[6] Yoshua Bengio, Olivier Delalleau, and Clarence Simard. Decision trees do not generalize to new variations. *Computational Intelligence*, 26(4):449–467, November 2010.

[7] Gautam Biswas, Daniel Schwartz, Krittaya Leelawong, and Nancy Vye. Learning by teaching: A new agent paradigm for educational software. *Applied Artificial Intelligence*, 19:363–392, March 2005.

[8] Hao Cen, Kenneth Koedinger, and Brian Junker. Learning factors analysis - a general method for cognitive model evaluation and improvement. In *Proceedings of the 8th International Conference on Intelligent Tutoring Systems*, pages 164–175, 2006.

[9] Tak-Wai Chan and Chih-Yueh Chou. Exploring the design of computer supports for reciprocal tutoring. *International Journal of Artificial Intelligence in Education*, 8:1–29, 1997.

[10] Michelene T. H. Chi, Paul J. Feltovich, and Robert Glaser. Categorization and representation of physics problems by experts and novices. *Cognitive Science*, 5(2):121–152, June 1981.

[11] Allen Cypher, Daniel C. Halbert, David Kurlander, Henry Lieberman, David Maulsby, Brad A. Myers, and Alan Turransky, editors. *Watch what I do: programming by demonstration*. MIT Press, Cambridge, MA, 1993.

[12] Peter Dayan, Geoffrey E. Hinton, Radford M. Neal, and Richard S. Zemel. The Helmholtz Machine. *Neural Computation*, 7(5):889–904, December 1995.

[13] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[14] G. E. Hinton. To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535–547, 2007.

[15] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, May 1995.

[16] Charles Kemp and Joshua B B. Tenenbaum. The discovery of structural form. *Proceedings of the National Academy of Sciences of the United States of America*, July 2008.

[17] Charles Kemp and Fei Xu. An ideal observer model of infant object perception. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *NIPS*, pages 825–832. MIT Press, 2008.

[18] Kenneth R. Koedinger, Ryan S.J.d. Baker, Kyle Cunningham, Alida Skogsholm, Brett Leber, and John Stamper. A data repository for the EDM community: The PSLC DataShop, 2010.

[19] Kenneth R. Koedinger and Elizabeth A. McLaughlin. Seeing language learning inside the math: Cognitive analysis yields transfer. In *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*, pages 471–476, Austin, TX, 2010.

[20] Kenneth R. Koedinger and Mitchell J. Nathan. The real story behind story problems: Effects of representations on quantitative reasoning. *The Journal of Learning Sciences*, 13(2):129–164, 2004.

[21] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: an architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[22] John E. Laird, Paul S. Rosenbloom, and Allen Newell. Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.

[23] Pat Langley and Dongkyu Choi. A unified cognitive architecture for physical agents. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, Boston, 2006.

[24] K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[25] Tessa Lau and Daniel S. Weld. Programming by demonstration: An inductive learning formulation. In *Proceedings of the 1999 international conference on intelligence user interfaces*, pages 145–152, 1998.

[26] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1:541–551, December 1989.

[27] Yann Lecun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[28] Nan Li, William W. Cohen, and Kenneth R. Koedinger. A computational model of accelerated future learning through feature recognition. In *ITS'10: Proceedings of 10th International Conference on Intelligent Tutoring Systems*, pages 368–370, 2010.

[29] Nan Li, Subbarao Kambhampati, and Sungwook Yoon. Learning probabilistic hierarchical task networks to capture user preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, Pasadena, CA, 2009.

[30] Noboru Matsuda, Andrew Lee, William W. Cohen, and Kenneth R. Koedinger. A computational model of how learner errors arise from weak prior knowledge. In *Proceedings of Conference of the Cognitive Science Society*, 2009.

[31] Tom Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[32] Tom M. Mitchell, Sridhar Mahadevan, and Louis I. Steinberg. Leap: a learning apprentice for vlsi design. In *Proceedings of the 9th international joint conference on Artificial intelligence*, pages 573–580, San Francisco, CA, 1985.

[33] Raymond J. Mooney. *A General Explanation-Based Learning Mechanism and its Application to Narrative Understanding*. Morgan Kaufmann, San Mateo, CA, 1990.

[34] Stephen Muggleton and Luc de Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19:629–679, 1994.

[35] Allen Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

[36] Alexandru Niculescu-Mizil and Rich Caruana. Inductive transfer for bayesian network structure learning. In *Proceedings of the 11th International Conference on AI and Statistics*, 2007.

[37] Philip I. Pavlik, Hao Cen, and Kenneth 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*, pages 121–130, 2009.

[38] Timo Niemirepo Pentti Hietala. The competence of learning companion agents. *International Journal of Artificial Intelligence in Education*, 9:178–192, 1998.

[39] J. R. Quinlan. Learning logical definitions from relations. *Mach. Learn.*, 5(3):239–266, 1990.

[40] Rajat Raina, Andrew Y. Ng, and Daphne Koller. Constructing informative priors using transfer learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 713–720, New York, NY, 2006.

[41] Marc'Aurelio Ranzato, Fu J. Huang, Y. Lan Boureau, and Yann LeCun. Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:1–8, 2007.

[42] Matthew Richardson and Pedro Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, 2006.

[43] Lawrence K. Saul, Tommi Jaakkola, and Michael I. Jordan. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

[44] Alberto Segre. A learning apprentice system for mechanical assembly. In *Proceedings of the Third IEEE Conference on AI for Applications*, pages 112–117, 1987.

[45] Patrice Y. Simard, Dave Steinkraus, and John C. Platt. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition*, Washington, DC, USA, 2003. IEEE Computer Society.

[46] Niels A. Taatgen and Frank J. Lee. Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, 45(1):61–75, 2003.

[47] Kikumi K. Tatsuoka. Rule space: An approach for dealing with misconceptions based on item response theory. *Journal of Educational Measurement*, pages 345–354, 1983.

[48] Ivan Titov and James Henderson. Constituent Parsing with Incremental Sigmoid Belief Networks. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 632–639, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[49] L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Relational macros for transfer in reinforcement learning. In *Proceedings of the 17th Conference on Inductive Logic Programming*, Corvallis, Oregon, 2007.

[50] Kurt VanLehn. *Mind Bugs: The Origins of Procedural Misconceptions*. MIT Press, Cambridge, MA, USA, 1990.

[51] Kurt Vanlehn, Stellan Ohlsson, and Rod Nason. Applications of simulated students: an exploration. *Journal of Artificial Intelligence in Education*, 5:135–175, February 1994.

[52] Michael Villano. Probabilistic student models: Bayesian belief networks and knowledge space theory. In *Proceedings of the 2nd International Conference on Intelligent Tutoring Systems*, pages 491–498, Heidelberg, 1992.

MACHINE LEARNING
DEPARTMENT

Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA15213

Carnegie Mellon