Knowledge Tracing in Sequential Learning of Inflected Vocabulary

Adithya Renduchintala and **Philipp Koehn** and **Jason Eisner**
Department of Computer Science
Johns Hopkins University
{adi.r,phi,eisner}@jhu.edu

Abstract

We present a feature-rich knowledge tracing method that captures a student's acquisition and retention of knowledge during a foreign language phrase learning task. We model the student's behavior as making predictions under a log-linear model, and adopt a neural gating mechanism to model how the student updates their log-linear parameters in response to feedback. The gating mechanism allows the model to learn complex patterns of retention and acquisition for each feature, while the log-linear parameterization results in an interpretable knowledge state. We collect human data and evaluate several versions of the model.

1 Introduction

Knowledge tracing attempts to reconstruct when a student acquired (or forgot) each of several facts. Yet we often hear that "learning is not just memorizing facts." Facts are not atomic objects to be discretely and independently manipulated. Rather, we suppose, a student who recalls a fact in a given setting is demonstrating a *skill*—by solving a structured prediction problem that is akin to reconstructive memory (Schacter, 1989; Posner, 1989) or pattern completion (Hopfield, 1982; Smolensky, 1986). The attempt at structured prediction may draw on many cooperating feature weights, some of which may be shared with other facts or skills.

In this paper, for the task of foreign-language vocabulary learning, we will adopt a specific structured prediction model and learning algorithm. Different knowledge states correspond to model parameter settings (feature weights). Different learning styles correspond to different hyperparameters that govern the learning algorithm.[1] As we interact with each student through a simple online tutoring system, we would like to track their evolving knowledge state and identify their learning style. That is, we would like to discover parameters and hyperparameters that can explain the evidence so far and predict how the student will react in future. This could help us make good future choices about how to instruct this student, although we leave this reinforcement learning problem to future work. In this paper, we show that we can predict the student's next answer.

In short, we expand the notion of a knowledge tracing model to include representations for a student's (i) current knowledge, (ii) retention of knowledge, and (iii) acquisition of new knowledge. Our reconstruction of the student's knowledge state remains interpretable, since it corresponds to the weights of hand-designed features (sub-skills). Interpretability may help a future teaching system provide useful feedback to students and to human teachers, and help it construct educational stimuli that are targeted at improving particular sub-skills, such as features that select correct verb suffixes.

Our present paper considers a verb conjugation task, where a foreign language learner learns the verb conjugation paradigm by reviewing and interacting with a series of flash cards. This task is a good testbed, as it needs the learner to deploy sub-word features and to generalize to new examples. For example, a student learning Spanish verb conjugation might encounter pairs such as (`tú entras`, `you enter`), (`yo miro`, `I watch`). Using these examples, the student needs to recognize suffix patterns and apply them to new pairs seen such as (`yo entro`, `I enter`).

Vocabulary learning presents a challenging learning environment due to the large number of skills (words) that need to be traced. Learning vocabulary in conjunction with inflection further complicates the challenge due to the number of new sub-skills that are introduced. Huang et al. (2016) suggest that modeling sub-skill interaction is crucial to several knowledge tracing domains. For our domain, a log-linear formulation elegantly allows for arbitrary sub-skills via feature functions.

[1] In the present paper, we assume that all students share the same hyperparameters (same learning style), although each student will have their own parameters, which change as they learn.

Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017), pages 238–247,
Vancouver, Canada, August 3 - August 4, 2017.

2 Related Work

Bayesian knowledge tracing (Corbett and Anderson, 1994) (BKT) has long been the standard method to infer a student's knowledge from his or her performance on a sequence of task items. In BKT, each skill is modeled by an HMM with two hidden states ("known" or "not-known"), and the probability of success on an item depends on the state of the skill it exercises. Transition and emission probabilities are learned from the performance data using Expectation Maximization (EM). Many extensions of BKT have been investigated, including personalization (e.g., Lee and Brunskill, 2012; Khajah et al., 2014a) and modeling item difficulty (Khajah et al., 2014b).

Our approach could be called **Parametric Knowledge Tracing (PKT)** because we take a student's knowledge to be a vector of prediction parameters (feature weights) rather than a vector of skill bits. Although several BKT variants (Koedinger et al., 2011; Xu and Mostow, 2012; González-Brenes et al., 2014) have modeled the fact that related skills share sub-skills or features, that work does not associate a real-valued weight with each feature at each time. Either skills are still represented with separate HMMs, whose transition and/or emission probabilities are parameterized in terms of shared features with time-invariant weights; or else HMMs are associated with the individual sub-skills, and the performance of a skill depends on which of its subskills are in the "known" state.

Our current version is not Bayesian since it assumes deterministic updates (but see footnote 4). A closely related line of work with deterministic updates is deep knowledge tracing (DKT) (Piech et al., 2015), which applied a classical LSTM model (Hochreiter and Schmidhuber, 1997) to knowledge tracing and showed strong improvements over BKT. Our PKT model differs from DKT in that the student's state at each time step is a more interpretable feature vector, and the state update rule is also interpretable—it is a type of error-correcting learning rule. In addition, the student's state is able to predict the student's actual response and not merely whether the response was correct. We expect that having an interpretable feature vector has better inductive bias (see experiment in section 7.1), and that it may be useful to plan future actions by smart flash card systems. Moreover, in this work we test different plausible state update rules and see how they fit actual student responses, in orer to gain insight about learning.

Most recently, Settles and Meeder (2016)'s half-life regression assumes that a student's retention of a particular skill exponentially decays with time and learns a parameter that models the rate of decay ("half-life regression"). Like González-Brenes et al. (2014) and Settles and Meeder (2016), our model leverages a feature-rich formulation to predict the probability of a learner correctly remembering a skill, but can also capture complex spacing/retention patterns using a neural gating mechanism. Another distinction between our work and half-life regression is that we focus on knowledge tracing within a single session, while half-life regression collapses a session into a single data point and operates on many such data points over longer time spans.

3 Verb Conjugation Task

We devised a flash card training system to teach verb conjugations in a foreign language. In this study, we only asked the student to translate from the foreign language to English, not vice-versa.[2]

3.1 Task Setup

We consider a setting where students go through a series of interactive flash cards during a training session. Figure 1 shows the three types of cards: (i) *Example* (EX) cards simply display a foreign phrase and its English translation (for 7 seconds). (ii) *Multiple-Choice* (MC) cards show a single foreign phrase and require the student to select one of five possible English phrases shown as options. (iii) *Typing* (TP) cards show a foreign phrase and a text input box, requiring the student to type out what they think is the English translation. g Our system can provide feedback for each student response. (i) *Indicative Feedback*: This refers to marking a student's answer as correct or incorrect (Fig. 1c, 1d and 1h). Indicative feedback is always shown for both MC and TP cards. (ii) *Explicit Feedback*: If the student makes an error on a TP card, the system has a 50% chance of showing them the true answer (Fig. 1g). (iii) *Retry*: If the student makes an error on a MC card, the system has a 50% chance of allowing them to try again, up to a maximum of 3 attempts.

3.2 Task Content

In this particular task we used three verb lemmas, each inflected in 13 different ways (Table 1). The inflections included three tenses (simple past,

[2]We would regard these as two separate skills that share parameters to some degree, an interesting subject for future study.

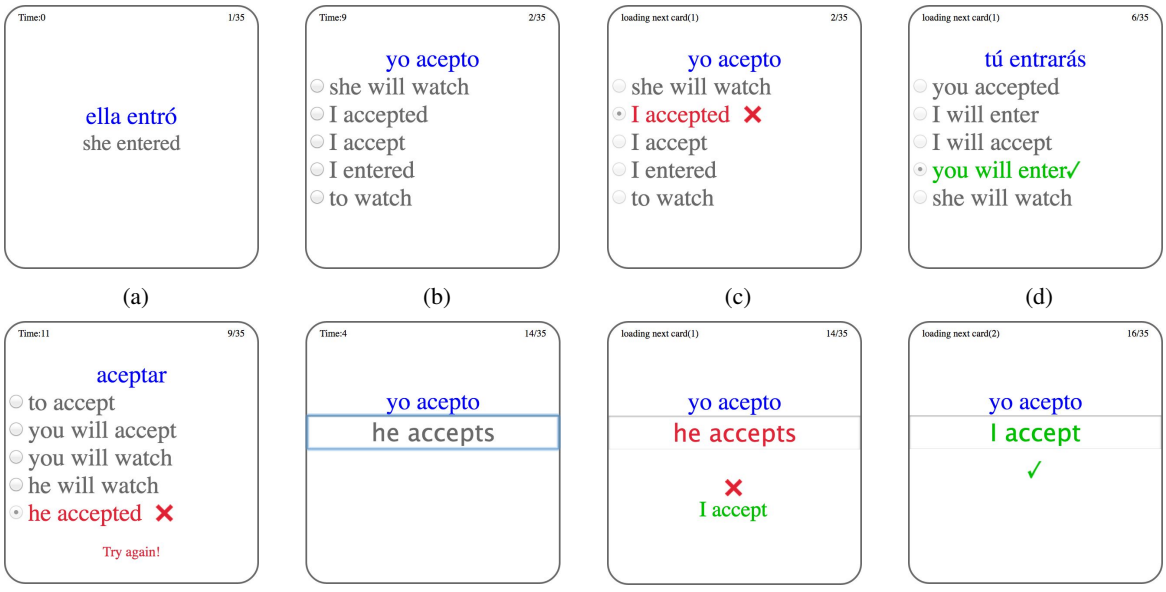


Figure 1: Screen grabs of card modalities during training. These examples show cards for a native English speaker learning Spanish verb conjugation. Fig 1a is an EX card, Fig 1b shows a MC card before the student has made a selection, and Fig 1c and 1d show MC cards after the student has made an incorrect or correct selection respectively, Fig 1e shows a MC card that is giving the student another attempt (the system randomly decides to give the student up to three additional attempts), Fig 1f shows a TP card where a student is completing an answer, Fig 1g shows a TP card that has marked a student answer wrong and then revealed the right answer (the reveal is decided randomly), and finally Fig 1h shows a card that is giving a student feedback for their answer.

Categories	Inf	SPre,1,N	SPre,2,N	SPre,3,M	SPre,3,F	SF,1,N	SF,2,N	SF,3,M	SF,3,F	SP,1,N	SP,2,N	SP,3,M	SP,3,F
Lemma	acceptar to accept	yo acepto I accept	tú aceptas you accept	él acepta he accepts	ella acepta she accepts	yo aceptaré I will accept	tú aceptarás you will accept*	él aceptará he will accept	ella aceptará she will accept	yo acepté I accepted*	tú aceptaste you accepted	él aceptó he accepted	ella aceptó she accepted
	entrar to enter	yo entro I enter	tú entras you enter	él entra he enters	ella entra she enters	yo entraré I will enter	tú entrarás you will enter	él entrará he will enter	ella entrará she will enter	yo entré I entered	tú entraste you entered	él entró he entered	ella entró she entered
	mirar to watch	yo miro I watch*	tú miras you watch*	él mira he watches*	ella mira she watches	yo miraré I will watch	tú mirarás you will watch*	él mirará he will watch	ella mirará she will watch	yo miré I watched	tú miraste you watched	él miró he watched*	ella miró she watched

Table 1: Content used in training sequences. Phrase pairs with * were used for the quiz at the end of the training sequence. This Spanish content was then transformed using the method in section 6.1.

present, and future) in each of four persons (first, second, third masculine, third feminine), as well as the infinitive form. We ensured that each surface realization was unique and regular, resulting in 39 possible phrases.[3] Seven phrases from this set were randomly selected for a quiz, which is shown at the end of the training session, leaving 32 phrases that a student may see in the training session. The student's responses on the quiz do not receive any feedback from the system.We also limited the training session to 35 cards (some of which may require multiple rounds of interaction, owing to retries). All of the methods presented in this paper could be applied to larger content sets as well.

4 Notation

We will use the following conventions in this paper. System actions a_t, student responses y_t, and feedback items a'_t are subscripted by a time $1 \leq t \leq T$. Other subscripts pick out elements of vectors or matrices. Ordinary lowercase letters indicate scalars (α, β, etc.), boldfaced lowercase letters indicate vectors ($\boldsymbol{\theta}$, $\mathbf{y}$, $\mathbf{w}^{\mathrm{zx}}$), and boldfaced uppercase letters indicate matrices ($\boldsymbol{\Phi}$, $\mathbf{W}^{\mathrm{hh}}$, etc.). The roman-font superscripts are part of the vector or matrix name.

[3]The inflected surface forms included explicit pronouns.

5 Student Models

5.1 Observable Student Behavior

A flash card is a structured object $a = (x, \mathcal{O})$, where $x \in \mathcal{X}$ is the foreign phrase and $\mathcal{O}$ is a set of allowed responses. For an MC card, $\mathcal{O}$ is the set of 5 multiple-choice options on that card (or fewer on a retry attempt). For a EX or TP card, $\mathcal{O}$ is the set of all 39 English phrases (the TP user interface prevents the student from submitting a guess outside this set).

For non-EX cards, we assume the student samples their response $y \in \mathcal{O}$ from a log-linear distribution parameterized by their knowledge state $\boldsymbol{\theta} \in \mathbb{R}^d$:

$$p(y \mid a; \boldsymbol{\theta}) = p(y \mid x, \mathcal{O}; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta} \cdot \boldsymbol{\phi}(x, y))}{\sum_{y' \in \mathcal{O}} \exp(\boldsymbol{\theta} \cdot \boldsymbol{\phi}(x, y'))} \quad (1)$$

where $\boldsymbol{\phi}(x,y) \in \mathbb{R}^d$ is a feature vector extracted from the (x,y) pair.

5.2 Feature Design

The student's knowledge state is described by the weights $\boldsymbol{\theta}$ placed on the features $\boldsymbol{\phi}(x,y)$ in equation (1). We assume the following binary features will suffice to describe the student's behavior.

- *Phrasal* features: We include a unique indicator feature for each possible (x,y) pair, yielding 39^2 features. For example, there exists a feature that fires iff $x = \texttt{yo\_miro} \wedge y = \texttt{I\_enter}$.
- *Word* features: We include indicator features for all (source word, target word) pairs: e.g., $\texttt{yo} \in x \wedge \texttt{enter} \in y$. (These words need not be aligned.)
- *Morpheme* features: We include indicator features for all (w,mc) pairs, where w is a word of the source phrase x, and m is a possible tense, person, or number for the target phrase y (drawn from Table 1). For example, m might be `1st` (first person) or `SPre` (simple present).
- *Prefix* and *suffix* features: For each word or morpheme feature that fires, 8 backoff features also fire, where the source word and (if present) the target word are replaced by their first or last i characters, for $i \in \{1,2,3,4\}$.

These templates yield about 4600 features in all, so the knowledge state has $d \approx 4600$ dimensions.

5.3 Learning Models

We now turn to the question of modeling how the student's knowledge state changes during their session. $\boldsymbol{\theta}_t$ denotes the state at the start of round t. We take $\boldsymbol{\theta}_1 = \mathbf{0}$ and assume that the student uses a deterministic update rule of the following form:[4]

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\beta}_t \odot \boldsymbol{\theta}_t + \boldsymbol{\alpha}_t \odot \mathbf{u}_t \tag{2}$$

where $\mathbf{u}_t$ is an *update vector* that depends on the student's experience (a_t,y_t,a'_t) at round t.

Why this form? First imagine that the student is learning by stochastic gradient descent on some L_2-regularized loss function $C \cdot \| \boldsymbol{\theta} \|^2 + \sum_t \mathcal{L}_t(\boldsymbol{\theta})$. This algorithm's update rule has the simplified form

$$\boldsymbol{\theta}_{t+1} = \beta_t \cdot \boldsymbol{\theta}_t + \alpha_t \cdot \mathbf{u}_t \tag{3}$$

where $\mathbf{u}_t = -\nabla \mathcal{L}_t(\boldsymbol{\theta})$ is the steepest-descent direction on example t, $\alpha_t > 0$ is the learning rate at time t, and $\beta_t = 1 - \alpha_t C$ handles the weight decay due to following the gradient of the regularizer.

Adaptive versions of stochastic gradient descent—such as AdaGrad (Duchi et al., 2011) and AdaDelta (Zeiler, 2012)—are more like our full rule (2) in that they allow different learning rates for different parameters.

In general, we can regard $\boldsymbol{\alpha}_t \in (0,1)^d$ as modeling the rates at which the learner updates the various parameters according to $\mathbf{u}_t$, and $\boldsymbol{\beta}_t \in (0,1)^d$ as modeling the rates at which those parameters are forgotten. These vectors correspond respectively to the *input gates* and *forget gates* in recurrent neural network architectures such as the LSTM (Hochreiter and Schmidhuber, 1997) or GRU (Cho et al., 2014). As in those architectures, we will use neural networks to choose $\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t$ at each time step t, so that they may be sensitive in nonlinear ways to the context at round t.

5.3.1 Schemes for the Update Vector $\mathbf{u}_t$

We assume that $\mathbf{u}_t$ is the gradient of some log-probability, so that the student learns by trying to increase the log-probability of the correct answer. However, the student does not always *observe* the correct answer y. For example, there is no output label provided when the student only receives feedback that their answer is incorrect. Even in such cases, the student can change their knowledge state.

In this section, we define schemes for defining $\mathbf{u}_t$ from the experience (a_t,y_t,a'_t) at round t. Recall that $a_t = (x_t, \mathcal{O}_t)$. We omit the t subscripts below.

Suppose the student is told that a particular phrase $y \in \mathcal{O}$ is the correct translation of x (via an EX card or via feedback on an answer to an MC or TP card). Then an apt strategy for the student would be to use the following gradient:[5]

$$\begin{aligned} \boldsymbol{\Delta}^{\checkmark} &= \nabla_{\boldsymbol{\theta}} \log p(y \mid x, \mathcal{O}; \boldsymbol{\theta}) \\ &= \boldsymbol{\phi}(x,y) - \sum_{y' \in \mathcal{O}} p(y' \mid x) \boldsymbol{\phi}(x,y') \end{aligned} \tag{4}$$

If the student is told that y is *incorrect*, an apt strategy is to move probability mass collectively to the other available options, increasing their total probability, since one of those options *must* be correct.

[4] Since learning is not perfectly predictable, it would be more realistic to compute $\boldsymbol{\theta}_t$ by a stochastic update—or equivalently, by a deterministic update that also depends on a random noise vector $\boldsymbol{\epsilon}_t$ (which is drawn from, say, a Gaussian). These noise vectors are "nuisance parameters," but rather than integrating over their possible values, a straightforward approximation is to optimize them by gradient descent—along with the other update parameters—so as to locally maximize likelihood.

[5] An objection is that for an EX or TP card, the student may not actually know the exact set of options $\mathcal{O}$ in the denominator. We attempted setting $\mathcal{O}$ to be the set of English phrases the student has seen prior to the current question. Though intuitive, this setting performed worse on all the update and gating schemes.

We call this the *redistribution gradient (RG)*:

$$\boldsymbol{\Delta}^{\text{✗}} = \nabla_{\boldsymbol{\theta}} \log p(\mathcal{O} - \{y\} \mid x, \mathcal{O}; \boldsymbol{\theta}) \quad (5)$$

$$= \sum_{y' \in \mathcal{O} - \{y\}} p(y' \mid x, y' \neq y) \boldsymbol{\phi}(x, y') \quad (6)$$
$$- \sum_{y' \in \mathcal{O}} p(y' \mid x) \boldsymbol{\phi}(x, y')$$

where $p(y' \mid x, y' \neq y)$ is a renormalized distribution over just the options $y' \in \mathcal{O} - \{y\}$. Note that if the student selects two wrong answers y_1, y_2 in a row on an MC card, the first update will subtract the average features of $\mathcal{O}$ and add those of $\mathcal{O} - \{y_1\}$; the second update will subtract the average features of $\mathcal{O} - \{y_1\}$ and add those of $\mathcal{O} - \{y_1, y_2\}$. The intermediate addition and subtraction cancel out if the same $\boldsymbol{\alpha}$ vector is used at both rounds, so the net effect is to shift probability mass from the 5 initial options to the 3 remaining ones.[6]

An alternate scheme for incorrect y is to use $-\boldsymbol{\Delta}^{\text{✓}}$. We call this *negative gradient (NG)*.

Since the RG and NG update vectors both worked well for handling incorrect y, we also tried linearly interpolating them *(RNG)*, with $\mathbf{u}_t = \boldsymbol{\gamma}_t \odot \boldsymbol{\Delta}^{\text{✗}} + (\mathbf{1} - \boldsymbol{\gamma}_t) \odot -\boldsymbol{\Delta}^{\text{✓}}$. The interpolation vector $\boldsymbol{\gamma}_t$ has elements in $(0,1)$, and may depend on the context (possibly different for MC and EX cards, for example).

Finally, the *feature vector (FG)* scheme simply adds the features $\phi(x, y)$ when y is correct or subtracts them when y is incorrect. This is appropriate for a student who pays attention only to y, without bothering to note that the alternative options in $\mathcal{O}$ are (respectively) incorrect or correct.

Recall from section 3.1 that the system sometimes gives both indicative and explicit feedback, telling the student that one phrase is incorrect and a different phrase is correct. We treat these as two successive updates with update vectors $\mathbf{u}_t$ and $\mathbf{u}_{t+1}$. Notice that in the FG scheme, adding this pair of update vectors resembles a perceptron update.

Table 2 summarizes our update schemes.

Update Scheme	Correct	Incorrect
redistribution (RG)	$\mathbf{u}_t = \boldsymbol{\Delta}^{\text{✓}}$	$\mathbf{u}_t = \boldsymbol{\Delta}^{\text{✗}}$
negative grad. (NG)	$\mathbf{u}_t = \boldsymbol{\Delta}^{\text{✓}}$	$\mathbf{u}_t = -\boldsymbol{\Delta}^{\text{✓}}$
feature vector (FG)	$\mathbf{u}_t = \boldsymbol{\phi}(x, y)$	$\mathbf{u}_t = -\boldsymbol{\phi}(x, y)$

Table 2: Summary of update schemes (other than RNG).

5.3.2 Schemes for the Gates $\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \boldsymbol{\gamma}_t$

We characterize each update t by a 7-dimensional context vector $\mathbf{c}_t$, which summarizes what the student has experienced. The first three elements in $\mathbf{c}_t$ are binary indicators of the type of flash card (EX, MC or TP). The next three elements are binary indicators of the type of information that caused the update: correct student answer, incorrect student answer, or revealed answer (via an EX card or explicit feedback). As a reminder, the system can respond with an indication that the answer is correct or incorrect, or it can reveal the answer. Finally, the last element of $\mathbf{c}_t$ is $1/|\mathcal{O}|$, the chance probability of success on this card. From $\mathbf{c}_t$, we define

$$\boldsymbol{\alpha}_t = \sigma(\mathbf{W}^{\alpha} \mathbf{c}_t \quad + b^{\alpha} \mathbf{1}) \qquad \in (0,1)^d \quad (7)$$

$$\boldsymbol{\beta}_t = \sigma(\mathbf{W}^{\beta} \mathbf{c}_{t-1} + b^{\beta} \mathbf{1}) \qquad \in (0,1)^d \quad (8)$$

$$\boldsymbol{\gamma}_t = \sigma(\mathbf{W}^{\gamma} \mathbf{c}_t \quad + b^{\gamma} \mathbf{1}) \qquad \in (0,1)^d \quad (9)$$

where $\mathbf{c}_0 = \mathbf{0}$. Each gate vector is now parameterized by a weight matrix $\mathbf{W} \in \mathbb{R}^{d \times 7}$, where d is the dimensionality of the gradient and knowledge state.

We also tried simpler versions of this model. In the *vector model (VM)*, we define $\boldsymbol{\alpha}_t = \sigma(\mathbf{b}^{\alpha})$, and $\boldsymbol{\beta}_t, \boldsymbol{\gamma}_t$ similarly. These vectors do not vary with time and simply reflect that some parameters are more labile than others. Finally, the *scalar model (SM)* defines $\boldsymbol{\alpha}_t = \sigma(b^{\alpha} \mathbf{1})$, so that all parameters are equally labile. One could also imagine tying the gates for features derived from the same template, meaning that some *kinds* of features (in some contexts) are more labile than others, or reducing the number of parameters by learning low-rank $\mathbf{W}$ matrices.

While we also tried augmenting the context vector $\mathbf{c}_t$ with the knowledge state $\boldsymbol{\theta}_t$, this resulted in far too many parameters to train well, and did not help performance in pilot tests.

5.4 Parameter Estimation

We tune the $\mathbf{W}$ and b parameters of the model by maximum likelihood, so as to better predict the students' responses y_t. The likelihood function is

$$p(y_1, \ldots y_T \mid a_t, \ldots a_T) = \prod_{t=1}^{T} p(y_t \mid a_{1:t}, y_{1:t-1}, a'_{1:t-1})$$
$$= \prod_{t=1}^{T} p(y_t \mid a_t; \boldsymbol{\theta}_t) \quad (10)$$

where we take $p(y_t \mid \cdots) = 1$ at steps where the student makes no response (EX cards and explicit

[6] Arguably, a zeroth update should be allowed as well: upon first viewing the MC card, the student should have the chance to subtract the average features of the full set of possibilities and add those of the 5 options in $\mathcal{O}$, since again, the system is implying that one of those 5 options *must* be correct.

feedback). Note that the model assumes that $\boldsymbol{\theta}_t$ is a sufficient statistic of the student's past experiences.

For each (update scheme, gating scheme) combination, we trained the parameters using SGD with RMSProp updates (Tieleman and Hinton, 2012) to maximize the regularized log-likelihood

$$\sum_{t,\tau_t=0} \log p(y_t \mid x_t;\boldsymbol{\theta}_t) - C \cdot \| \mathbf{W} \|^2 \qquad (11)$$

summed over all students. Note that $\boldsymbol{\theta}_t$ depends on the parameters through the gated update rule (2).

The development set was used for early stopping and to tune the regularization parameter C.[7]

6 Data Collection

We recruited 153 unique "students" via Amazon Mechanical Turk (MTurk). MTurk participants were compensated $1 for completing the training and test sessions and a bonus of $10 was given to the three top scoring students. In our dataset, we retained only the 121 students who answered all questions.

6.1 Language Obfuscation

Fig. 1 shows a few example flash cards for a native English speaker learning Spanish. Fig. 1 shows all our Spanish-English phrase pairs. In our actual task, however, we invented an artificial language for the MTurk students to learn, which allowed us to ignore the problem of students with different initial knowledge levels. We generated our artificial language by enciphering the Spanish orthographic representations. We created a mapping from the true source string alphabet to an alternative, manually defined alphabet, while attempting to preserve pronounceability (by mapping vowels to vowels, etc.). For example, `mirar` was transformed into `melil` and `tú aceptas` became `pi icedpiz`.

6.2 Card Ordering Policy

In the future, we expect to use planning or reinforcement learning to choose the sequence of stimuli for the student. For the present study of student behavior, however, we hand-designed a simple stochastic policy for choosing the stimuli.

The policy must decide what foreign phrase and card modality to use at each training step. Our policy likes to repeat phrases with which participants had trouble—in hopes that these already-taught phrases are on the verge of being learned. It also likes to pick out new phrases. This was inspired by the popular Leitner (1972) approach, which devised a system of buckets that control how frequently an item is reviewed by a student. Leitner proposed buckets with review frequency rates of every day, every 2 days, every 4 days and so on.

For each foreign phrase $x \in \mathcal{X}$, we maintain a *novelty score* v_x, which is a function of the number of times the phrase is exposed to a student and an *error score* e_x, which is a function of the number of times the student incorrectly responded to the phrase. These scores are initialized to 1 and updated as follows:[8]

$$\begin{aligned} v_x &\leftarrow v_x - 1 \text{ when } x \text{ is viewed} \\ e_x &\leftarrow \begin{cases} 2e_x \text{ when student gets } x \text{ wrong} \\ 0.5e_x \text{ when student gets } x \text{ right} \end{cases} \\ x &\sim \frac{g(\mathbf{v}) + g(\mathbf{e})}{2} \end{aligned} \qquad (12)$$

On each round, we sample a phrase x from either P_v or P_e (equal probability); these distributions are computed by applying a softmax $g(.)$ over the vectors $\mathbf{v}$ and $\mathbf{e}$ respectively (see Eq. 12). Once the phrase x is decided, the modality (EX, MC, TP) is chosen stochastically using probabilities $(0.2,0.4,0.4)$, except that probabilities $(1,0,0)$ are used for the first example of the session, and $(0.4,0.6,0)$ if x is not "TP-qualified." A phrase is TP-qualified if the student has seen both x's pronoun and x's verb lemma on previous cards (even if their correct translation was not revealed). For an MC card, the distractor phrases are sampled uniformly without replacement from the 38 other phrases.

7 Results & Experiments

We partitioned the students into three groups: 80 students for training, 20 for development, and 21 for testing. Most students found the task difficult; the average score on the 7-question quiz—was 2.81 correct, with maximum score of 6. (Recall from section 3.2 that the quiz questions were typing questions, not multiple choice questions.)

After constructing each model, we evaluated it on the held-out data: the 728 responses from the 21 testing students. We measure the log-probability under the model of each actual response ("cross-entropy"), and also the fraction of responses that

[7] We searched $C \in \{0.00025, 0.0005, 0.001, ..., 0.01, 0.025, 0.05, 0.1\}$ for each gating model and update scheme combination. $C = 0.0025$ gave best results for the CM models, 0.01 for VM and 0.0005 for SM.

[8] Arguably we should have updated e_x instead by adding/subtracting 1, since it will be exponentiated later.

were correctly predicted if our prediction was the model's max-probability response ("accuracy").

Table 3 shows the results of our experiment. All of our models were predictive, doing far better than a uniform baseline that assigned equal probability $1/|\mathcal{O}|$ to all options. Our best models are shown in the final two lines, RNG+VM and RNG+CM.

Which update scheme was best? Interestingly, although the RG update vector is principled from a *machine* learning viewpoint, the NG update vector sometimes achieved better accuracy—though worse perplexity—when predicting the responses of *human* learners.[9] We got our best results on both metrics by interpolating between RG and NG (the RNG scheme). Recall that the NG scheme was motivated by the notion that students who guessed wrong may not study the alternative answers (even though one is correct), either because it is too much trouble to study them or because (for a TP card) those alternatives are not actually shown.

Which gating mechanism was best? In almost all cases, we found that more parameters helped, with CM > VM > SM on accuracy, and a similar pattern on cross-entropy (with VM sometimes winning but only slightly). In short, it helps to use different learning rates for different features, and it probably helps to make them sensitive to the learning context.

Surprisingly, the simple FG scheme outperformed both RG and NG when used in conjunction with a scalar retention and acquisition gate. This, however, did not extend to more complex gates.

Fig. 2 shows a breakdown of the prediction accuracy measures according to whether the card was MC or TP, and according to whether the student's answer was correct (C) or incorrect (IC). Unsurprisingly, all the models have an easier time predicting the student's guess when the student is correct, since the predicted parameters $\boldsymbol{\theta}_t$ will often pick the correct answer. However, this is where the vector and context gates far outperform the scalar gates. All the models find predicting the incorrect answers of the students difficult. Moreover, when predicting these incorrect answers, the RG models do slightly better than the NG models.

The models obviously have higher accuracy when predicting student answers for MC cards than for TP cards, as MC cards have fewer options. Again, within both of these modalities, the vector and context gates outperform the scalar gate.

[9]Even the FG vector sometimes won (on both metrics!), but this happened only with the worst gating mechanism, SM.

Update Scheme	Gating Mechanism	accuracy	cross-ent.
(Uniform baseline)		0.133	2.459
FG	SM	0.239*	2.362
FG	VM	0.357†	2.130
FG	CM	0.401	2.025
RG	SM	0.135	3.194
RG	VM	0.397†	1.909
RG	CM	0.405	1.938
NG	SM	0.185*	4.674
NG	VM	0.394†	2.320
NG	CM	0.449†*	2.244
RNG (mixed)	SM	0.183	3.502
RNG (mixed)	VM	0.427	1.855
RNG (mixed)	CM	0.449	1.888

Table 3: Table summarizing prediction accuracy and cross-entropy (in nats per prediction) for different models. Larger accuracies and smaller cross-entropies are better. Within an update scheme, the † indicates significant improvement (McNemar's test, $p < 0.05$) over the next-best gating mechanism. Within g a gating mechanism, the * indicates significant improvement over the next-best update scheme. For example, NG+CM is significantly better than NG+VM, so it receives a †; it is also significantly better than RG+CM, and receives a * as well. These comparisons are conducted only among the pure update schemes (above the double line). All other models are significantly better than RG+SM ($p < 0.01$).

Finally, Fig. 3 examines how these models behave when making specific predictions over a training sequence for a single student. At each step we plot the difference in log-probability between our model and a uniform baseline model. Thus, a marker above 0 means that our model assigned the student's answer a probability higher than chance.[10] To contrast the performance difference, we show both the highest-accuracy model (RNG+CM) and the lowest-accuracy model (RG+SM). For a high-scoring student (Fig. 3a), we see RNG+CM has a large margin over RG+SM and a slight upward trend. A higher probability than chance is noticeable even when the student makes mistakes (indicated by hollow markers). In contrast, for an average student (Fig. 3b), the margin between the two models is less perceptible. While the CM+NG model is still above the SM+RG line, there are some answers where CM+NG does very poorly. This is especially true for some of the wrong answers, for example at training steps 25, 29 and 33. Upon closer inspection into the model's error in step 33, we found the prompt received at this training step was `ekki melü` as a MC card, which had been shown to the student on three prior occasions, and the student even answered correctly on one of these occasions. This explains why the model

[10]For MC cards, the chance probability is in $\{\frac{1}{5}, \frac{1}{4}, \frac{1}{3}\}$—depending on how many options remain—while for TP cards it is $\frac{1}{39}$.

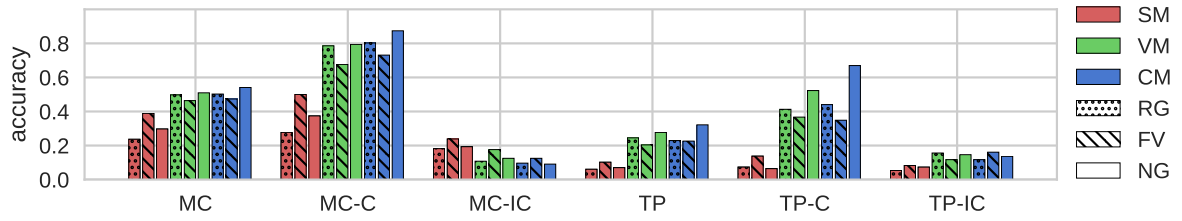
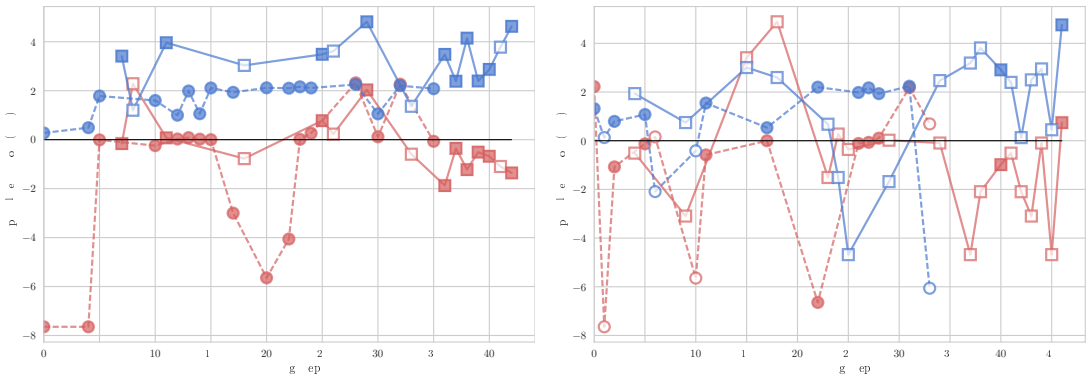


Figure 2: Plot comparing the models on test data under different conditions. Conditions MC and TP indicate Multiple-choice and Typing questions respectively. These are broken down to the cases where the student answers them correctly C and incorrectly IC. SM, VM, and CM represent scalar, vector, and context retention and acquisition gates (shown with different colors), respectively, while RG, NG and FG are redistribution, negative and feature vector update schemes(shown with different hatching patterns).



(a) a student with quiz score 6/7

(b) a student with quiz score 2/7

Figure 3: Predicting a specific student's responses. For each response, the plot shows our model's improvement in log-probability over the uniform baseline model. TP cards are the square markers connected by solid lines (the final 7 squares are the quiz), while MC cards—which have a much higher baseline—are the circle markers connected by dashed lines. Hollow and solid markers indicate correct and incorrect answers respectively. The RNG+CM model is shown in blue and the FG+SM model in red.

was surprised to see the student make this error.

7.1 Comparison with Less Restrictive Model

Our parametric knowledge tracing architecture models the student as a typical structured prediction system, which maintains weights for hand-designed features and updates them roughly as an online learning algorithm would. A natural question is whether this restricted architecture sacrifices performance for interpretability, or improves performance via useful inductive bias.

To consider the other end of the spectrum, we implemented a flexible LSTM model in the style of recent deep learning research. This alternative model predicts each response by a student (i.e., on an MC or TP card) given the entire history of previous interactions with that student as summarized by an LSTM. The LSTM architecture is formally capable of capturing update rules exactly like those of PKT, but it is far from limited to such rules.

Much like equation (1), at each time t we predict

$$p(y_t = y \mid a_t) = \frac{\exp(\mathbf{h}_t \cdot \boldsymbol{\psi}(y))}{\sum_{y' \in \mathcal{O}_t} \exp(\mathbf{h}_t \cdot \psi(y))} \qquad (13)$$

for each possible response y in the set of options $\mathcal{O}_t$, where $\boldsymbol{\psi}(y) \in \mathbb{R}^d$ is a learned embedding of response y. Here $\mathbf{h}_t \in \mathbb{R}^d$ denotes the hidden state of the LSTM, which evolves as the student interacts with the system and learns. $\mathbf{h}_t$ depends on the LSTM inputs for all times $< t$, just like the knowledge state $\boldsymbol{\theta}_t$ in equations (1)–(2). It also depends on the LSTM input for time t, since that specifies the flash card a_t to which we are predicting the response y_t.

Each flash card $a = (x, \mathcal{O})$ is encoded by a concatenation $\mathbf{a}$ of three vectors: a one-hot 39-dimensional vector specifying the foreign phrase x, a 39-dimensional binary vector $\boldsymbol{\mathcal{O}}$ indicating the possible English options in $\mathcal{O}$, and a one-hot vector indicating whether the card is EX, MC, or TP.

When reading the history of past interactions, the LSTM input at each time step t concatenates the vector representation $\mathbf{a}_t$ of the current flash card with vectors $\mathbf{a}_{t-1}, \mathbf{y}_{t-1}, \mathbf{f}_{t-1}$ that describe the student's experience in round $t-1$: these respectively encode the previous flash card, the student's response to it (a one-hot 39-dimensional vector), and the resulting feedback (a 39-dimensional binary vector that indicates the remaining options after feedback). Thus, if the student receives no feedback, then $\mathbf{f}_{t-1} = \boldsymbol{\mathcal{O}}_{t-1}$. Indicative feedback sets $\mathbf{f}_{t-1} = \mathbf{y}_{t-1}$

Model	Parameters	Accuracy(test)	Cross-Entropy
RNG+CM	$\approx$ 97K	0.449	1.888
LSTM	$\approx$ 25K	0.429	1.992

Table 4: Comparison of our best-performing PKT model (RNG+CM) to our LSTM model. On our dataset, PKT outperforms the LSTM both in terms of accuracy and cross-entropy.

or $\mathbf{f}_{t-1} = \boldsymbol{\mathcal{O}}_{t-1} - \mathbf{y}_t$, according to whether the student was correct or incorrect. Explicit feedback (including for an EX card) sets $\mathbf{f}_{t-1}$ to a one-hot representation of the correct answer. Thus, $\mathbf{f}_{t-1}$ gives the set of "positive" options that we used in the RG update vector, while $\boldsymbol{\mathcal{O}}_{t-1}$ gives the set of "negative" options, allowing the LSTM to similarly update its hidden state from $\mathbf{h}_{t-1}$ to $\mathbf{h}_t$ to reflect learning.[11]

As in section 5.4, we train the parameters by L_2-regularized maximum likelihood, with early stopping on development data. The weights for the LSTM were initialized uniformly at random $\sim U(-\delta, +\delta)$, where $\delta = 0.01$, and RMSProp was used for gradient descent. We settled on a regularization coefficient of 0.002 after a line search. The number of hidden units d was also tuned using line search. Interestingly, a dimensionality of just $d = 10$ performed best on dev data:[12] at this size, the LSTM has *fewer* parameters than our best model.

The result is shown in Table 4. These results favor our restricted PKT architecture. We acknowledge that the LSTM might perform better when a larger training set was available (which would allow a larger hidden layer), or using a different form of regularization (Srivastava et al., 2014).

Intermediate or hybrid models would of course also be possible. For example, we could predict $p(y \mid a_t)$ via (1), defining $\boldsymbol{\theta}_t$ as $\mathbf{h}_t^\top M$, a learned linear function of h_t. This variant would again have access to our hand-designed features $\boldsymbol{\phi}(x, y)$, so that it would know which flash cards were similar. In fact $\boldsymbol{\theta}_t \cdot \boldsymbol{\phi}(x,y)$ in (1) equals $\mathbf{h}_t \cdot (M\boldsymbol{\phi}(x,y))$, so M can be regarded as projecting $\boldsymbol{\phi}(x,y)$ down to the LSTM's hidden dimension d, learning how to weight and use these features. In this variant, the LSTM would no longer need to take a_t as part of its input at time t: rather, $\mathbf{h}_t$ (just like $\boldsymbol{\theta}_t$ in PKT) would be a pure representation of the student's knowledge state at time t, capable of predicting y_t for *any* a_t. This setup more closely resembles PKT—or the DKT LSTM of Piech et al. (2015). Unlike the DKT paper, however, it would still predict the student's specific response, not merely whether they were right or wrong.

[11] This architecture is formally able to mimic PKT. We would store $\boldsymbol{\theta}$ in the LSTM's vector of cell activations, and configure the LSTM's "input" and "forget" gates to update this according to (2) where $\mathbf{u}_t$ is computed from the input. Observe that each feature in section 5.2 has the form $\phi_{ij}(x,y) = \xi_i(x) \cdot \psi_j(y)$. Consider the hidden unit in $\mathbf{h}$ corresponding to this feature, with activation θ_{ij}. By configuring this unit's "output" gate to be $\xi_i(x)$ (where x is the current foreign phrase given in the input), we would arrange for this hidden unit to have output $\xi_i(x) \cdot \theta_{ij}$, which will be multiplied by $\psi_j(y)$ in (13) to recover $\theta_{ij} \cdot \phi_{ij}(x,y)$ just as in (1). (More precisely, the output would be sigmoid$(\xi_i(x) \cdot \theta_{ij})$, but we can evade this nonlinearity if we take the cell activations to be a scaled-down version of $\boldsymbol{\theta}$ and scale up the embeddings $\boldsymbol{\psi}(y)$ to compensate.)

[12] We searched $0.001, 0.002, 0.005, 0.01, 0.02, 0.05$ for the regularization coefficient, and $5, 10, 15, 20, 50, 100, 200$ for the number of hidden units.

8 Conclusion

We have presented a cognitively plausible model that traces a human student's knowledge as he or she interacts with a simple online tutoring system. The student must learn to translate very short inflected phrases from an unfamiliar language into English. Our model assumes that when a student recalls or guesses the translation, he or she is attempting to solve a structured prediction problem of choosing the best translation, based on salient features of the input-output pair. Specifically, we characterize the student's knowledge as a vector of feature weights, which is updated as the student interacts with the system. While the phrasal features memorize the translations of entire input phrases, the other features can pick up on the translations of individual words and sub-words, which are reusable across phrases.

We collected and modeled human-subjects data. We experimented with models using several different update mechanisms, focusing on the student's treatment of negative feedback and the degree to which the student tends to update or forget specific weights in particular contexts. We also found that in comparison to a less constrained LSTM model, we can better fit the human behavior by using weight update schemes that are broadly consistent with schemes used in machine learning.

In the future, we plan to experiment with more variants of the model, including variants that allow noise and personalization. Most important, we mean to use the model for planning which flash cards, feedback, or other stimuli to show next to a given student.

Acknowledgments

This material is based upon work supported by a seed grant from the Science of Learning Institute at Johns Hopkins University.

References

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, pages 1724–1734. http://www.aclweb.org/anthology/D14-1179.

Albert T Corbett and John R Anderson. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction* 4(4):253–278.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12(Jul):2121–2159.

José González-Brenes, Yun Huang, and Peter Brusilovsky. 2014. General features in knowledge tracing to model multiple subskills, temporal item response theory, and expert knowledge. In *Proceedings of the 7th International Conference on Educational Data Mining*. University of Pittsburgh, pages 84–91.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

J. J. Hopfield. 1982. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences of the USA*. volume 79, pages 2554–2558.

Yun Huang, J Guerra, and Peter Brusilovsky. 2016. Modeling skill combination patterns for deeper knowledge tracing. In *Proceedings of the 6th Workshop on Personalization Approaches in Learning Environments (PALE 2016)*. 24th Conference on User Modeling, Adaptation and Personalization, Halifax, Canada.

Mohammad Khajah, Rowan Wing, Robert Lindsey, and Michael Mozer. 2014a. Integrating latent-factor and knowledge-tracing models to predict individual differences in learning. In *Proceedings of the 7th International Conference on Educational Data Mining*.

Mohammad M Khajah, Yun Huang, José P González-Brenes, Michael C Mozer, and Peter Brusilovsky. 2014b. Integrating knowledge tracing and item response theory: A tale of two frameworks. In *Proceedings of Workshop on Personalization Approaches in Learning Environments (PALE 2014) at the 22th International Conference on User Modeling, Adaptation, and Personalization*. University of Pittsburgh, pages 7–12.

K. R. Koedinger, P. I. Pavlick Jr., J. Stamper, T. Nixon, and S. Ritter. 2011. Avoiding problem selection thrashing with conjunctive knowledge tracing. In *Proceedings of the 4th International Conference on Educational Data Mining*. Eindhoven, NL, pages 91–100.

Jung In Lee and Emma Brunskill. 2012. The impact on individualizing student models on necessary practice opportunities. *International Educational Data Mining Society* .

Sebastian Leitner. 1972. *So lernt man lernen: der Weg zum Erfolg*. Herder, Freiburg.

Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. 2015. Deep knowledge tracing. In *Advances in Neural Information Processing Systems*. pages 505–513.

Michael I Posner. 1989. *Foundations of cognitive science*. MIT press Cambridge, MA.

D. L. Schacter. 1989. Memory. In M. I. Postner, editor, *Foundations of Cognitive Science*, MIT Press, pages 683–725.

Burr Settles and Brendan Meeder. 2016. A trainable spaced repetition model for language learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 1848–1858. http://www.aclweb.org/anthology/P16-1174.

Paul Smolensky. 1986. Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press/Bradford Books, Cambridge, MA, volume 1: Foundations, pages 194–281.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15:1929–1958. http://jmlr.org/papers/v15/srivastava14a.html.

Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4(2).

Yanbo Xu and Jack Mostow. 2012. Comparison of methods to trace multiple subskills: Is LR-DBN best? In *Proceedings of the 5th International Conference on Educational Data Mining*. pages 41–48.

Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* .