dAFM: Fusing Psychometric and Connectionist Modeling for Q-matrix Refinement

Zachary A. Pardos
University of California, Berkeley
zp@berkeley.edu

Anant Dadu
University of Illinois Urbana-Champaign
adadu2@illinois.edu

We introduce a model which combines principles from psychometric and connectionist paradigms to allow direct Q-matrix refinement via backpropagation. We call this model dAFM, based on augmentation of the original Additive Factors Model (AFM), whose calculations and constraints we show can be exactly replicated within the framework of neural networks. In order to parameterize the Q-matrix definition in the model, the associations between questions and knowledge components (KC) need to be represented by adjustable weights. Furthermore, student KC opportunity counts, instead of serving as fixed inputs, need to be calculated dynamically as the Q-matrix changes during training. We describe our solutions to these two modeling challenges and evaluate several variants of our fully realized model on datasets from the Cognitive Tutor and ASSISTments. We compare learning the Q-matrix from scratch vs. refining an expert specified KC model and evaluate various procedures for refinement. In our quantitative predictive analysis, we find that dAFM learns a better generalizing Q-matrix than the original expert model in all our primary datasets. Using a development set, we also find that the dAFM Q-matrix is superior to KC representations extracted from trained Deep Knowledge Tracing and skip-gram models. Examples are shown of questions whose fit was improved by dAFM with depictions of their original and refined KC associations. We consistently find in our experiments that our dAFM variant which attempted to learn the Q-matrix from scratch underperformed models which started with an expert defined Q-matrix that was then refined. This observation continues a theme in EDM of utility found in the enduring value of expert domain knowledge enhanced through data-driven refinement.
Keywords: neural networks, IRT, Q-matrix, KC model, DKT, skip-gram, learning, measurement

1. INTRODUCTION

Psychometric instruments have been concerned with the measurement of well-defined dimensions of ability, assessed using responses to items designed to align with constructs corresponding to those dimensions. The alignment of constructs to items is often represented by way of a sparse matrix known as a Q-matrix (Tatsuoka, 1983), used by graphical models to estimate cognitive mastery based on student response sequences (Corbett, 2001). Neural networks have served as a connectionist alternative to classical graphical models of the mind (Fodor and Pylyshyn, 1988), encoding a cognitive state as a continuous, or distributed vector representation. In a neural network, individual node activations in the hidden layers do not directly correspond to any particular construct. Instead, a construct may be represented by another vector in the space having some relation to a cognitive state or may be distributed in its representation, much like the vector offset representation of lexical relations in word embeddings (Levy and Goldberg,

2014). This distributed representation underlies all deep-learning approaches to neural networks and their many advances (LeCun et al., 2015). A key to the success of connectionist representations has been their ability to be learned from data by way of the backpropagation algorithm (Rumelhart et al., 1986; Williams and Zipser, 1989). It is with this backdrop that we aim to fuse these two modeling paradigms. We will be using the continuous, connectionist representation fitting algorithm of backpropagation within the framework of neural networks to adjust the Q-matrix, while attempting to have each node in our Q-matrix representation maintain its local psychometric interpretation as a construct, or knowledge component (KC), per dimension.

Additive Factors Model (AFM; Pavlik Jr et al. 2009a) is our model of choice, a mainstay in educational data mining research (Pelánek, 2017; Koedinger et al., 2015; Pardos, 2017) and a derivative of the psychometric measurement model, Item Response Theory (IRT; Rasch 1961). AFM is a logistic model that predicts the probability of a student responding correctly to an item as a function of the student's ability, the sum of the difficulties of the KCs associated with the item, and the growth (learning) rate of each of the KCs multiplied by the number of times the student has encountered each KC. AFM has already been combined with a Q-matrix refinement algorithm named Learning Factors Analysis (LFA; Cen et al. 2006; Koedinger et al. 2012), which relies on A* search (Hart et al., 1968) to navigate the tree of operations (e.g., concatenate) that can be performed on KCs in combination with a secondary list of item factors, which itself can be an alternative KC model. In our approach, we translate AFM and the Q-matrix representation into the framework of neural networks where, instead of a distance heuristic determining the branches of the A* tree to explore, different Q-matrices can be expressed as existing on the loss surface of a neural network (Choromanska et al., 2015), traversed by gradient-based optimizers using backpropagation. This approach removes the outer loop of an item to KC association search and internalizes it with the other parameters of the model. It requires no additional list of item factors to conduct the search and focuses on improving the Q-matrix association of items with KCs, keeping the number of KCs fixed.

We introduce dAFM[1] as a neural network implementation of AFM, mathematically equivalent to its original form but with the added ability to refine, or learn from scratch, its Q-matrix from data. We will show how we address each of the major challenges in allowing for Q-matrix learning, namely (1) posing the Q-matrix as a weight coefficient matrix, adjustable by backpropagation, and (2) dynamically re-calculating the student KC opportunity counts with respect to the changing item to KC associations. We first train the dAFM model without modifying the Q-matrix to arrive at the standard AFM construction, then allow the Q-matrix, and other AFM parameters, to be fine-tuned. We then answer the following research questions:

- **RQ1**: Does dAFM refinement of a domain expert KC model (Clark et al., 2006) result in improved response prediction over the original expert model?
- **RQ2**: Does a KC model learned from the ground-up by dAFM (without the benefit of an expert model starting point) predict with accuracy equal to or greater than a domain expert KC model refined by dAFM?

After related work, we will present the details of the original AFM model and its conversion to a neural network, and then we will introduce the Q-matrix refinement capable dAFM model. Several variants of dAFM and comparison models will then be described, followed by

[1]The meaning of the first letter of our model's name is in reference to both the '**dynamic**' aspect of its Q-matrix (Psychometric), and '**deep**' neural networks (Connectionist)

presentation of initial prototyping results on a development dataset, then presentation of the main prediction results on our five datasets. Lastly, a brief qualitative analysis will be presented in which interpretation of the model refined Q-matrix is conducted, followed by a discussion of limitations and conclusions.

2. Related Work

The association of items to knowledge components in a Q-matrix directly affects how adaptive instruction, predicated on cognitive mastery (Corbett, 2001), will perform. The value proposition is that, with an improved Q-matrix will come a reduction in student over and under practice in an adaptive tutoring system (Cen et al., 2007). The phenomenon of expert blind spot (Nathan et al., 2001) is one theoretical reason why an expert's Q-matrix, architected through cognitive task analysis (Clark et al., 2006), might not agree with the earlier stage abstractions of knowledge components that students are learning. Large datasets of responses to items during the learning process can be leveraged towards reducing the dissonance between these two models.

Learning curve analysis (Ritter and Schooler, 2001; Martin et al., 2011) was the first data-driven technique used to modify a KC model in a learning context. The technique involves the manual process of observing the average performance of students on KCs by opportunity count and inspecting the item(s) associated with deviations from the expected inverse power shape of this error curve (corresponding to the power law of learning). AFM was used to score candidate KC models searched for using Learning Factors Analysis (Cen et al., 2006), an automated approach (Koedinger et al., 2012) to this curve smoothing idea. Aside from AFM, matrix factorization (Desmarais et al., 2011; Desmarais and Naceur, 2013) and Bayesian networks (González-Brenes and Mostow, 2012) have been frameworks in which Q-matrix learning has been explored. Non-negative matrix factorization has been used to learn the association of test items to broad and distinct subjects from the bottom-up. Application of this model has been limited to stationary, non-learning contexts since the two dimensions of matrix factorization (student and item) prohibit the addition of a necessary temporal term to denote the chronology of responses over time. Tensor factorization, incorporating time, has been applied to predicting performance in a learning context (Sahebi et al., 2016) but its use in imputing the Q-matrix has not been demonstrated and may not yet be tractable. Also from the stationary context, diagnostic classification models (DCM; Liu et al. 2012) and cognitive diagnosis models (CDM; Chiu 2013; Sun et al. 2014) have explored Q-matrix refinement with the motivation that an improved specification would also improve the accuracy and validity of examinee ability estimates. This benefit was seen by (Liu et al., 2016) when adding step-level misconception KCs to the Q-matrix in their AFM model. Related to the KC model improvement problem has been work exploring prerequisite relationships (KC to KC) from data (Scheines et al., 2014; Piech et al., 2015; Chen et al., 2016). Those works, which depart from the view of knowledge components as independent, rely on a valid, fixed Q-matrix (item to KC) specification as an input. Also related is work predicting the KC association of untagged items in a tutoring system based on semantics from the item content (Rosé et al., 2005) and the contexts in which the item has appeared (Pardos and Dadu, 2017).

With many modeling frameworks being employed across core psychometric and educational data mining fields, the genre of hybridizing models has gained in momentum. The most frequent hybridization has been combining Item Response Theory and its derivatives with Bayesian Knowledge Tracing (BKT; Khajah et al. 2014; González-Brenes et al. 2014), a Hidden Markov

based model (HMM) tracking cognitive mastery from temporal item responses in learning contexts. Deep Knowledge Tracing (Piech et al., 2015) is a recurrent neural network adaptation to this same context. While inspired by BKT, and both having a recurring temporal structure, it shares more in common with Performance Factors Analysis (PFA; (Pavlik Jr et al., 2009b)), another IRT derivative, in that it models correct and incorrect responses with independent coefficients and does not have an estimate of a student's KC mastery separate from its prediction of a student's performance on an item of that KC. IRT and BKT, in contrast, have estimates of a student's latent ability and item parameters which describe how responses to those items affect the ability estimate.

Neural networks have, for a time, been considered among the more opaque (Burrell, 2016) machine learning models, popularly referred to as a "black box." This negative perception of their interpretability has been, in part, due to inappropriately viewing neural networks as models of local instead of continuous representation. Choices in the design of model topology can aid interpretability. The max-pooling design of Convolutional Neural Network (CNN; (Krizhevsky et al., 2012)) layers necessitates that each successive layer represents an ever more abstracted representation of the features of the input image. The single hidden layer topology of a word2vec model and design choice to not use non-linear activations allows for a proper vector space to be created whereby embedded elements can be compared algebraically in the space (Mikolov et al., 2013). These are examples of how careful architecting of a neural network can bring about desired interpretability. We follow this paradigm of deliberate design in our construction of dAFM.

3. Additive Factors Model

Additive Factors Model (AFM) is a logistic model predicting a dichotomous item response as a function of the student's ability, the sum of KC difficulties associated with the item, and the KC growth terms associated with the item multiplied by their opportunity counts. Formally[2], this is the probability that a student j will answer an item i correctly based on the student's baseline proficiency (θ_j), the baseline difficulty (β_k) of the required KCs (q_{ik}), and the improvement (γ_k) in those KCs as the student accumulates practice opportunities (T_{jk}). Training the model maximizes the likelihood of the predicted response, formulated in Figure 1. In this statistical model, the discrete portion is represented by q_{ik}, which takes on a value of one if item i is associated with KC k (according to the Q-matrix) and a zero if it is not. The knowledge components associated with the item carry with them a measure of difficulty (β_k) and an estimate of learning or growth (γ_k) representing the amount gained by each practice opportunity (T_{jk}) to answer items associated with the knowledge component k. The opportunity count (T_{jk}) is the total number of times student j has previously attempted items of KC k associated with item i. Implicitly, T_{jk} is referring to student j's count immediately prior to answering item i.

[2]In the first incarnation of the Additive Factors Model (Cen et al., 2005), student is represented with a j and item with an i, consistent with the notation used to describe IRT. In a follow-up paper (Cen et al., 2006), this notation was swapped, with others following suit (Pavlik Jr et al., 2009a; Pavlik Jr et al., 2009b; Koedinger et al., 2012), but without justification for the change given. Bayesian Knowledge Tracing model notation has differed as well, denoting student with s (Corbett and Anderson, 1994) and more recent individualized approaches using u (Yudelson et al., 2013). Given the lack of standard convention in the learner modeling literature, we return to the original notation of j for student and i for item, used in the Psychometrics literature from which AFM was derived.

$$\ln \frac{p_{ij}}{1-p_{ij}} = \theta_j + \sum_{k=1}^{K} q_{ik}\beta_k + \sum_{k=1}^{K} q_{ik}\gamma_k T_{jk}$$

Figure 1: Additive Factors Model logistic formulation

3.1. AFM POSED AS A NEURAL NETWORK

In this section, we describe how a mathematically equivalent AFM model can be implemented in the form of a neural network architecture. We make this transition of the model into the framework of neural networks so that we may later augment the model and reap the benefits of backpropagation.

We will start by showing the equivalence of a standard logistic regression to a single-layer feed-forward neural network with sigmoid activation, also known as the single-layer perceptron. Let $p(x)$ be the probability that the data point having a feature vector given by x, is associated with a correct response. The β term is the weight coefficient vector and β_0 is the bias, or intercept. Formally, the logistic regression model (Eq. 1) and sing-layer perceptron (Eq. 2) are described as:

$$\ln \frac{p(x)}{1-p(x)} = \beta_0 + x.\beta \tag{1}$$

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2}$$

$$p(x) = \frac{e^{\beta_0 + x.\beta}}{1+e^{\beta_0 + x.\beta}} = \frac{1}{1+e^{-(\beta_0 + x.\beta)}} = \sigma(\beta_0 + x.\beta)) \tag{3}$$

Observing the equations above, the $p(x)$ is directly analogous to the sigmoid activation used in neural networks. In Equation 3, $x.\beta$ is the dot product of the weight coefficients and the input vector, which is equivalent to a perceptron in a neural network. Like a perceptron, the output of the single dense node is the sum of the incoming weights multiplied by their inputs with a bias added and pushed through the squashing function of a sigmoid activation.

The input to the model (Figure 2) will comprise of the following input layers: A one-hot vector representing student j, a multi-hot KC representation vector for the item i answered by student j, equivalent to that item's row in the Q-matrix, and a vector with the opportunity counts of the KCs associated with item i and all other values zeroed. The parameters β, γ, and θ are defined in terms of edge weights in the neural network topology. The student one-hot vector is mapped to a single node using the weight vector θ, containing the general ability of each student. The KC multi-hot vector is mapped using the weights β, the difficulty of each KC, and the opportunity counts vector for KCs associated with the item is mapped with edge weights γ, the growth rate of each KC. Finally, all three inputs are multiplied by their edge weight vectors, summed at the single output node with the scalar sum, then passed through the sigmoid activation to give p_{ij}, the probability of student j answering item i correctly.

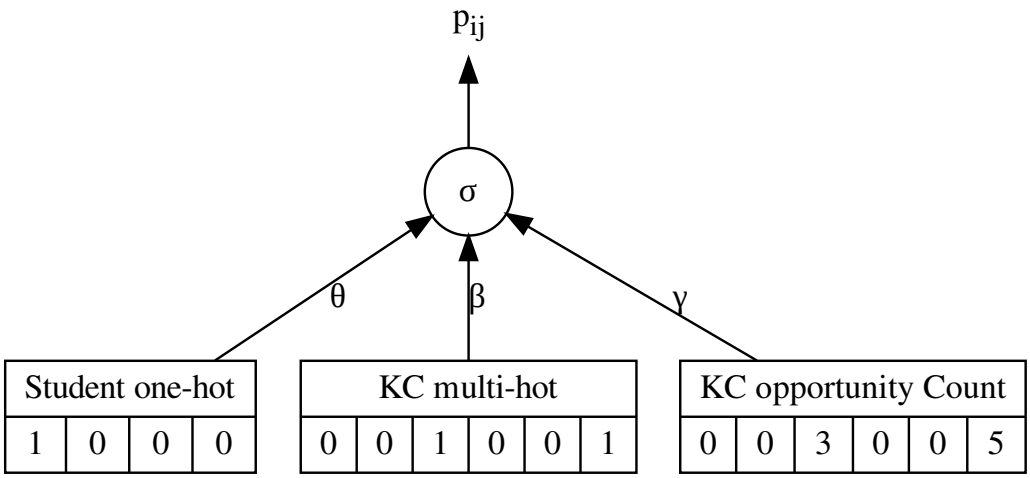

Figure 2: The AFM model posed as a feed-forward neural network with three separate input layers representing the student, the KC(s) of the item the student answered, and the student opportunity counts associated with the KC(s) of the item. The inputs are multiplied element-wise with their respective weight vectors, summed, and sent through a sigmoid activation function to predict correctness.

4. DAFM

With AFM posed as a neural network, we now introduce the Q-matrix learning augmented version of the model, called dAFM.

4.1. REPRESENTING THE Q-MATRIX AS A WEIGHT COEFFICIENT MATRIX

AFM and most BKT models concentrate on measurement at the KC level and therefore have not required there to be an identification of individual items as part of the input data. The Q-matrix look-up is already implicitly applied in most datasets, with each response row containing the KC tagged to the item being responded to.

In order to model the Q-matrix explicitly, a layer needed to be added (q_k) that represented associations of items to KCs, accepting the item i of student j as input in the form of a one-hot representation of the item (q_{ij}). As per the all-connected nature of adjacent layers in a connectionist model, this layer's respective weight matrix (W_{Qk}) represents the mapping of items to KCs and is initialized with binary values corresponding to the KC associations in an existing expert Q-matrix. Therefore, the architecture is now able to represent the Q-matrix as a weight matrix adjustable by backpropagation. Figure 3 illustrates this adjustable structure with an example item input. The KC representation layer (q_k) in this figure is representing the same information as the KC multi-hot input layer of AFM in Figure 2.

$$W_{Qk} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

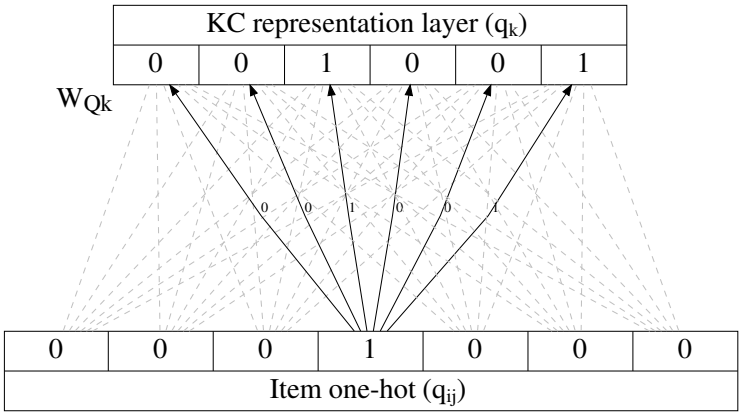

Figure 3: The added dAFM representation layers allowing for backpropagation-based adjustment of the Q-matrix. On the left is an example domain expert Q-matrix, with items as rows and KCs as columns. On the right is an example of how the weights of the model are initialized to represent the domain expert model item association to KC(s).

4.2. Dynamic calculation of opportunity counts

In AFM, the opportunity count of the student on the KCs involved in the item being predicted is often part of the input data. These counts can be found as feature columns in the datasets made available from Cognitive Tutors and can otherwise be manually generated. In dAFM, the model is making modifications to the association of items to KCs, which in turn, retrospectively changes the opportunity counts for every student. In this section, we describe how we addressed the problem of modeling dynamic opportunity counts in dAFM. This approach is predicated on the link between practice opportunity and growth (learning) as a desirable theoretical constraint to retain from the original model.

To dynamically calculate opportunity counts as the Q-matrix is changing, we utilized a recurrent neural network model in which we have fixed both the recurrent weight matrix $W_{\mathbf{TT}}$ and the input weight matrix $W_{\mathbf{QT}}$ to be the identity matrix (having all diagonal elements 1s and the rest 0s) to make the RNN work as a counter. Let q_k be the KC representation vector for the item the student gave a response to at time slice t. At $t = 0$, the output is initialized to zero (i.e., the student has not seen any items). At $t = 1$, the output vector T_k, which is of k dimensions, will increase the opportunity count for the KC(s) which are associated with the question attempted by the student at her first time slice. Figure 4 demonstrates the behavior of the above mentioned RNN model, calculating counts according to the formula shown in Equation 4 using example values for q_k as inputs. As can be seen by the example in Table 3, the identity weight matrices W_{QT} and W_{TT} (Eq. 5) are effectively a notational trick that allows element-wise addition between q_k and T_k so that T_k accumulates (i.e., sums) the q_k vectors over time. Thus, to solve the problem of updating the opportunity count with the change in Q-matrix, the KC representation layer q_k is the input to the RNN model. The opportunity count T_k at t is dependent on this representation and will therefore also change when the Q-matrix changes, starting back at $t = 0$ and calculating the opportunity counts forward in time according to the current Q-matrix mapping defined by W_{Qk}.

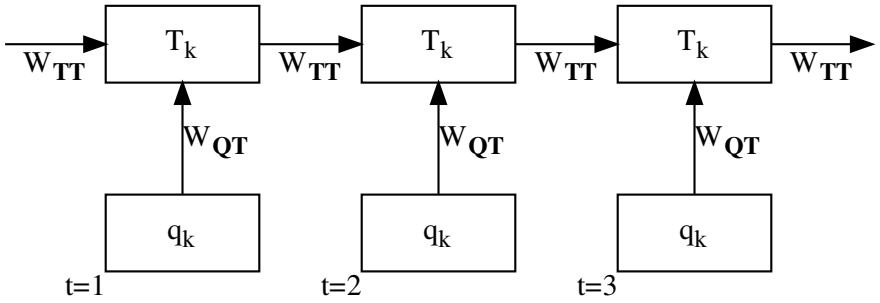

Figure 4: A Recurrent Neural Network acting as a simple opportunity counter

Table 1: Example opportunity counts resulting from the accumulation of KC representations of the items answered at each time slice

time slice	**KC multi-hot (q_k)**	**Opportunity counts (T_k)**
$t = 0$	—	$T_k = [0, 0, 0, 0, 0]$
$t = 1$	$q_k = [0, 0, 0, 1, 0]$	$T_k = [0, 0, 0, 1, 0]$
$t = 2$	$q_k = [0, 0, 1, 0, 0]$	$T_k = [0, 0, 1, 1, 0]$
$t = 3$	$q_k = [1, 0, 0, 1, 0]$	$T_k = [1, 0, 1, 2, 0]$
$t = 4$	$q_k = [0, 1, 0, 0, 0]$	$T_k = [1, 1, 1, 2, 0]$

$$T_k = q_k W_{\mathbf{QT}} + T_{k(t-1)} W_{\mathbf{TT}} \tag{4}$$

$$W_{\mathbf{QT}} = W_{\mathbf{TT}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{k \times k} \tag{5}$$

4.3. dAFM Full Definition

Traditional Recurrent Neural Networks (RNNs) map an input sequence x_1,...,x_t to an output sequence y_1,...,y_t. The architecture of dAFM incorporates an RNN to calculate opportunity counts of a student at every time slice t and therefore also conforms to this input and output format. In the dAFM model, the input sequence is q_{ij1},..,q_{ijt} which is the sequence of item i one-hots for a particular student j (depicted in Figure 3) corresponding to each of their t responses in the tutoring system. The y_1,...,y_t output or label sequence is the binary correctness of each of their t responses. The probability of a correct response to an item i at time slice t by student j is defined as p_{ijt}, used to predict the label.

A full depiction of the dAFM topology is shown in Figure 5. The dAFM model involves various parameters out of which a few are fixed (non-adjustable) while others are updateable by backpropagation to minimize the binary cross-entropy loss of the response predictions.

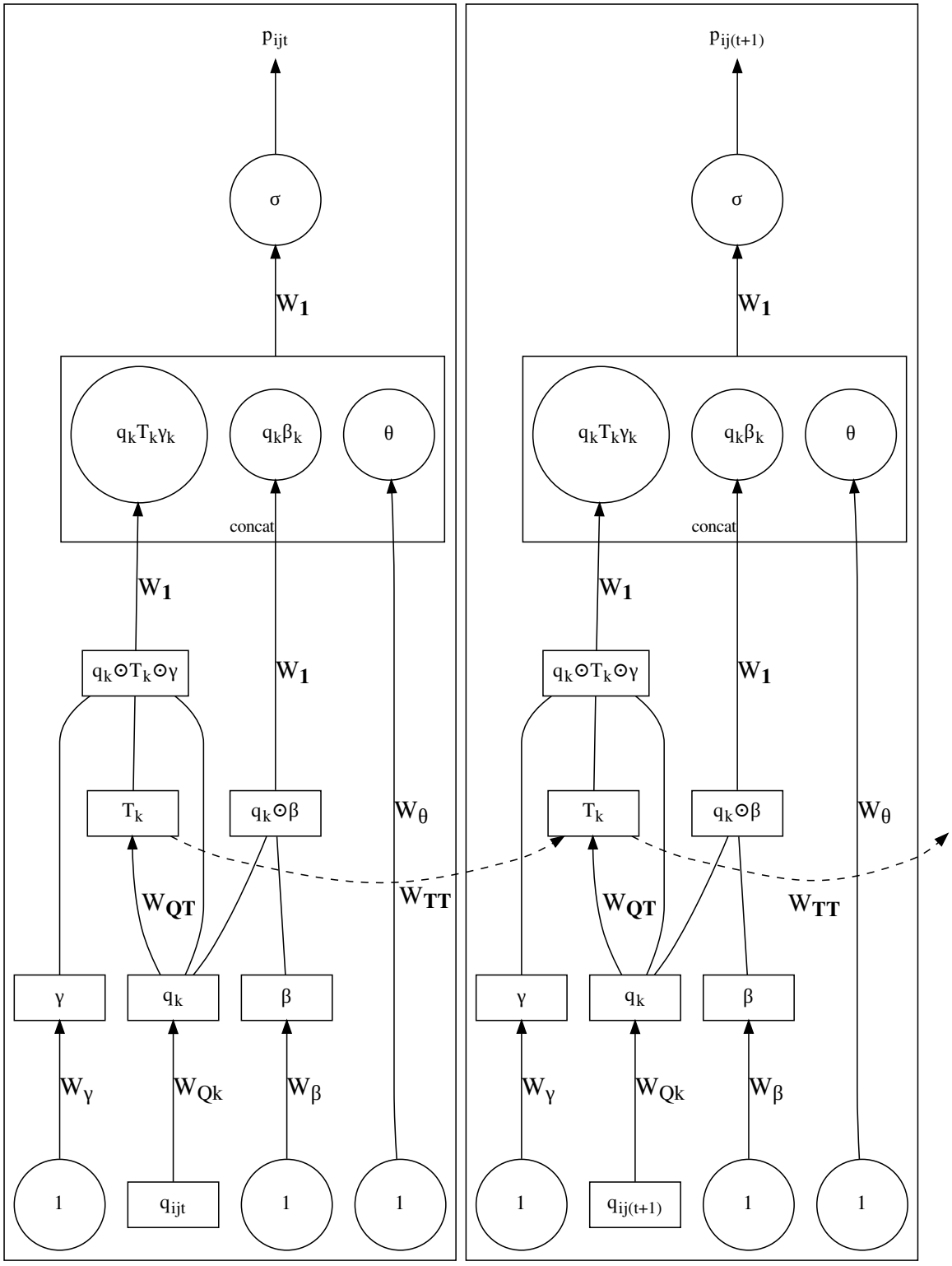

Figure 5: The dAFM model full topology specification across two time slices

The trainable parameters in the dAFM model are described below:

- W_{Qk}: The Q-matrix that is represented as a weight coefficient matrix, adjusted to improve an existing or randomly initialized Q-matrix.
- W_β: A vector of length k (number of KCs) containing the intercepts (difficulties) associated with each KC.
- W_γ: The vector containing the growth (or learning) rate for every KC.
- T_k: The vector containing the opportunity counts for each KC which dynamically updates during inference and during training as the Q-matrix W_{Qk} changes.
- W_θ: In dAFM, this is the ability average (or bias) for all students, represented by a single scalar value. In the original AFM, this represents the individual ability estimates for all students, which can be used when using item level cross-validation. In dAFM, we are focused on learning item to KC associations, and thus we validate at the student level, which does not utilize individual student abilities. Individualizing θ in the model is prototyped on a development set.

The non-trainable parameters in the dAFM model are:

- W_1: A vector of all 1s meant to act as a pass-through of the input values multiplied with it element-wise. It appears on several edges in the model and matches the size of the input, either k or 3 in our model.
- W_{QT} and W_{TT}: These matrices are a part of the RNN dynamic opportunity counting and are fixed to the identity to serve as a pass-through.

The step-wise procedure to compute the probability of student j answering an item i correct at time slice t using the dAFM model is formalized below:

$$
\begin{aligned}
q_k &= q_{ijt} W_{Qk} \\
\beta &= W_\beta \\
\gamma &= W_\gamma \\
T_k &= q_k W_{QT} + T_{k(t-1)} W_{TT} \\
q_k \beta_k &= q_k \odot \beta \\
q_k \gamma_k T_k &= q_k \odot \gamma \odot T_k \\
p_{ijt} &= \sigma(\theta + q_k \beta_k + q_k \gamma_k T_k)
\end{aligned}
$$

The final dAFM formulation is very similar to that of AFM (seen in Figure 1), reflecting our intent to adhere as closely as possible to the original model, augmenting only what is necessary to enable Q-matrix adjustment.

4.4. VARIANTS OF DAFM

We defined six variations of the dAFM model, five of which share the exact same topology shown in Figure 5 and differ only in how they are trained with respect to an expert Q-matrix. The sixth, QkDense, is the only variant to modify the topology. Variants which utilize an expert Q-matrix are:

- dafm-afm: Refers to the same model as the base AFM, with no Q-matrix modification but implemented in our neural network architecture. In this model, the W_β, W_γ, and single bias (θ) parameters are learned, while W_{Qk} is made non-adjustable.
- expert-init: In this model, all the parameters (including W_{Qk}) can train from the start with W_{Qk} initialized based on the original expert Q-matrix.
- fine-tuned (FT): The model is trained in two phases. It begins with training the base AFM parameters around the original Q-matrix (dafm-afm). In the second phase, it allows the Q-matrix (W_{Qk}) to be adjusted, along with all the other parameters (expert-init). The stopping criterion for both phases is when the validation loss no longer decreases.
- rounded fine-tuned: The same as fine-tuned except that the weights of the W_{Qk} matrix are rounded to 0 or 1 after training. This rounding of neural network weights to get binary KC associations is similar to the rounding of predicted KC imputations using non-negative matrix factorization (Desmarais and Naceur, 2013).
- QkDense: In this model, dafm-afm is augmented with an additional dense layer of size k placed just above the q_k layer. The weights leading into this dense layer are initialized to the identity to represent, initially, a pass-through of the KC association vector for the input item. During training, only the original AFM parameters and the weights associated with this layer are adjustable. This layer is meant to be able to learn skill-to-skill relationships. For example, the weights could change such that two skills merge into one.

The variant which does not utilize an expert Q-matrix is:

- random-init: This model is the same as expert-init, except that instead of using an expert Q-matrix as the starting weights for W_{Qk}, uniform random weights between 0 and 1 are used (a random initial Q-matrix).

4.5. OPTIMIZATION

The training objective is to minimize the binary cross-entropy loss of predicting the first attempt correctness for all the student responses in the training set. Let y be the target output, and $\hat{y}$ be the predicted probability of correct using the dAFM model. The loss for a single student is given by:

$$L = -\sum_{j=1}^{T}(y_j \log(\hat{y}_j) + (1 - y_j) \log(1 - \hat{y}_j) \tag{6}$$

This objective is minimized using RMSprop, Adam, and Adagrad as candidate optimizers in our evaluation, all of which are based on stochastic gradient descent.

Table 2: Dataset Description

Dataset	students	responses	items	KCs
Geometry	59	5,104	139	15
ASSISTments 2009-2010	6,806	331,774	12,609	319
ASSISTments 2012-2013	28,436	2,060,131	44,898	197
Cognitive Tutor Bridge 2006-2007	1,146	1,817,476	129,263	493
Cognitive Tutor Bridge 2008-2009	5,985	11,239,188	243,511	807

5. Datasets

We used five datasets, ranging in size, for the evaluation of dAFM; Cognitive Tutor for Geometry, ASSISTments 2009-2010, ASSISTments 2012-2013, Cognitive Tutor Bridge to Algebra 2006-2007, and Cognitive Tutor Bridge to Algebra 2008-2009. One unit from Bridge to Algebra 2008-2009, *INTRO-PERIM-AREA*, was used as a development set to prototype early approaches. Descriptive stats of the five datasets are shown in Table 2. All datasets used are publicly available.

ASSISTments (Heffernan and Heffernan, 2014) is an online web tutoring platform used primarily for middle and high school mathematics. Students' assignments are selected entirely by their teacher from a library of problem sets. Many ASSISTments problem sets implement a simple form of adaptivity whereby students are marked as having completed the set if they answer three items (or problems) correct in a row.

Cognitive tutors (Ritter et al., 2007) are intelligent tutoring systems with Carnegie Learning producing products based on this paradigm mostly for high-school mathematics curricula. Units are the top level of organization of a curriculum, followed by sections, then problems containing the items (or steps) that students respond to. Instead of the three correct in a row heuristic of ASSISTments, they employ Bayesian Knowledge Tracing to estimate when cognitive mastery has been achieved for a student on a particular KC. When mastery is estimated to have been achieved, the student no longer needs to answer items of that KC and can move on to the next section once the other KCs in the section have also been mastered. The two large datasets from the Cognitive Tutor Bridge to Algebra curriculum, with 1.8M and 11.2M responses, were made available for a data mining competition to predict student responses within the tutor (Stamper and Pardos, 2016). The much smaller Cognitive Tutor Geometry set (collected 1996-1997), with only 59 students and 15 KCs, will serve as our dataset for qualitative analysis because of the depictions of all the problems conveniently included in the dataset package.

6. Methods and Evaluation

6.1. Experiments

The largest Cognitive Tutor dataset was split up and trained by unit in order to keep within memory constraints. All other datasets were wholly trained with a single model. The dimensions of the layers in dAFM are all determined by the number of items in the dataset and the number of KCs defined in the expert model. There are, therefore, very few hyperparameters to adjust in the model. We conducted a hyperparameter search on three parameters: activation function,

optimizer, and learning rate. The activation functions applied to the q_k layer were: rectified linear unit (ReLU), linear (pass-through), and sigmoid. We searched learning rates of 0.01 and 0.1 for the ASSISTments sets and the Cognitive Tutor '06-'07 set, and 0.001, 0.01, and 0.1 on the individual units of the Cognitive Tutor '08-'09 set and on Cognitive Tutor Geometry. For optimizers, we chose RMSprop, Adam, and Adagrad. Conducting this search on each of the six dAFM variants resulted in 648 total models trained. The experiments were run on a machine with 1TB of system memory and four Xeon processors with a total of 48 cores. The mini-batch size was set to 16, 32, or 64, favoring batches (of entire student sequences), the largest of which would fit into memory. Training time for each model varied between 6 and 20 hours.

All models were implemented in python and described using the Keras neural network framework. The code for dAFM fine-tuned can be found in the appendix section. Full training examples and documentation for all model variants can be found in our code repo[3].

6.2. Evaluation

We split all datasets by student for evaluation, such that 80% of students were in train and 20% in test, with 20% of the students in train serving as a validation set. The training and validation sets were used to conduct all prototyping, and the test set was only allowed to be predicted once for the best hyperparameter model from each dAFM variant. The validation set was also used as a hill climbing set to determine the number of epochs to train for. A train/test hold-out strategy, as opposed to k-fold cross-validation, was preferable as it produces a single trained model, instead of k, for our qualitative study. Additionally, it made the training more manageable for our high computation cost experiments. We chose root mean square error (RMSE), an error metric which has been prescribed for skill model evaluation tasks like ours (Pelánek, 2015). It was applied to the predictions of first response attempts across all the students in the test set and calculated per student and then averaged across students to represent the reported metric.

To make the selection of which Q-matrix refined items would be used for qualitative study, we broke down RMSE by item and looked to see which items' RMSE most improved after dAFM refinement as compared to using the original Q-matrix. Improvement was calculated by both absolute RMSE decrease and decrease as a percent of original RMSE. This item improvement comparison was conducted for predictions made on the test set. The problem with the most improvement by percentage in the test set was chosen for presentation. We chose only among the best models using the ReLU activation function for q_k, as we wanted to avoid looking at negative weight values, the interpretation of which is not clear. ASSISTments provides preview links for problems linkable by their ASSISTment ID, found in the dataset; however, upon trying to look-up the best performing problems, we were unable to identify them with the preview link, and therefore only the Geometry set was qualitatively analyzed. Items from the other Cognitive Tutor datasets were not qualitative analyzed because their depictions are not publicly available.

6.3. Methods Explored Only on the Development Set

In this section, we present alternative approaches to neural network Q-matrix induction and other modifications to dAFM prototyped on our development set, *INTRO-PERIM-AREA*, from the Bridge to Algebra '08-'09 dataset. This unit was attempted by 438 students and had 98,768 responses to 3,706 items. There were a total 44 KCs associated with these items. These ap-

[3] https://github.com/CAHLR/dAFM

proaches showed less potential than the models used on our primary datasets, but nevertheless may inform future work.

6.3.1. Deriving the Q-matrix from Continuous Representations of Items

Before applying dAFM, we explored deriving a Q-matrix from the continuous representations extracted from the embedding of items from a Skip-gram applied to item sequences, utilizing the same models used for imputing the KCs of untagged items (Pardos and Dadu, 2017). We also explored deriving a Q-matrix from continuous representations found in various weight matrices of Deep Knowledge Tracing. Once the continuous vector representations of the items were extracted, k-means clustering was run on the vectors using various values of k with respect to the original number of KCs in the expert model. The assigned clusters were then taken as the KC, and this new Q-matrix was evaluated using dafm-afm, but with the cluster KC model used in place of an expert model.

Deep Knowledge Tracing (Piech et al., 2015) introduced an RNN model for predicting student responses to exercises based on their past responses. DKT maps an input sequence of feature vectors $(x_1, ..., x_t)$ to target sequence vectors $(a_1, ...a_t)$. In the DKT model, x_t is a one-hot encoding of the KC associated with items concatenated with the student's first attempt correctness on those items, and a_t is the correctness of the item, whose identifier is specified by another input, q_t. The mathematical formulation for DKT is shown in Equations 7 and 8. In the published DKT model, their input (x_t) was sequences of correct and incorrect responses represented at the KC level. In our adaptation for the purposes of learning an item embedding, we instead present sequences of correct and incorrect responses represented at the item level. Similarly, in the published DKT paper, the number of outputs appear to correspond to the number of KCs. In our adaptation, the number of outputs equals the number of unique items. In Equation 9, let $\delta(q_{t+1})$ be the one-hot encoding of the item answered at $t+1$ used to mask the correctness prediction outputs (y_t^T) presented to the binary cross-entropy loss function (ℓ).

$$h_t = tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \tag{7}$$

$$y_t = \sigma(W_{yh}h_t + b_y) \tag{8}$$

$$L = \sum_t \ell(y_t^T \delta(q_{t+1}), a_{t+1}) \tag{9}$$

In DKT, since x_t embeds the correct and incorrect token representing a response to an item, W_{xh} will have separate representations for each. After training the model, there are four ways to get the representation of each of the items out of the model:

- rnn-correct: Representations are taken from the W_{xh} matrix of only the correct attempt of the items.
- rnn-incorrect: Representations are taken from the W_{xh} matrix of only for the incorrect attempt of the items.
- rnn-correct-incorrect: Representations are taken from the W_{xh} matrix, and the correct and incorrect attempts of the items are concatenated.

- rnn-dense: Representations of items are taken from the output (W_{hy}) matrix.

Skip-gram was the second method we used in the development set, a three-layer neural network with an input layer, hidden layer, and output layer. The input is the item one-hot with a one-hot output for the items in context. The representations are provided by the weight matrix that maps an input layer to the hidden layer. Past work (Pardos and Dadu, 2017) used this embedding to impute the KC of an item based on its vector proximity to items with known KCs. We adapt this same model and the findings from that study as justification for why the position of items embedded in the space may suggest a KC association. There are two hyperparameters in this model that we have used in our experiments–window size and the size of hidden layer (the length of the representation vector). We used 20, 40, and 60 for the window size and 100 and 200 for the hidden layer size.

After extracting the question representation from the embeddings, we applied k-means clustering on their continuous vectors to assign them to clusters which serve as the KCs in this Q-matrix learned from scratch. We conducted our experiments by varying the number of clusters and distance measure. Let k be the total number of KCs in the current KC model. The different number of clusters tried were: k/2, k-k/10, k, k+k/10, and k*2. While the clustered items are completely lacking in semantic interpretability, they represent a reasonable alternative source for Q-matrix construction from other neural networks to compare with dAFM. Finally, the dafm-afm model is applied, using the clustering derived Q-matrix, only learning the growth and KC difficulty parameters, and then predicting students' responses on a validation set evaluated using RMSE.

6.3.2. Changing the Activation Function for q_k Layer

The activation function of an item's KC representation layer (q_k) plays an important role in the interpretation of a Q-matrix. If adhering to local representation and non-fuzzy, binary tagging, the output of that layer should have values tending towards either 0 or 1. Additionally, because the values of q_k are directly added to a running opportunity count total, negative values lead to a reduction in this count and would thus affect its intended interpretation. We tried achieving values in the range of 0 to 1 by using a sigmoid activation with a high discrimination term to force values towards the extremes. However, this suffered convergence problems during training. The closer to a binary function this output becomes (higher discrimination term), the less informative the gradient. This desired local binary representation at least appears to be at odds with model optimization. When using the sigmoid activation to constrain outputs, we made sure to initialize the $0s$ of W_{Qk} to -99 and $1s$ to 99 since a value of 0, representing no KC association when using a linear activation, would result in 0.50 association when passed through a sigmoid.

6.3.3. Including Individual Student Ability Estimates and Section Information

As mentioned previously, dAFM uses average student ability W_θ to make predictions rather than individual student ability. We here elaborate and motivate the decision to explore using individual ability for training while still using average ability for prediction. In the original AFM student model improvement work (Koedinger et al., 2012), cross-validation was done at the item level, allowing for the use of individual student ability estimates across items. Since dAFM focuses on learning refinements to item-KC associations, it would not be an appropriate evaluation to test on items excluded from the training process. While this reduces the utility of

Table 3: The table shows the error for the models that include student information.

Section	AFM	FT	QkDense	random-init	expert-init	rounded FT
Standard θ	0.3314	**0.3308**	0.3288	**0.3427**	**0.3314**	0.3312
Individualized θ_j	0.3314	0.3312	**0.3286**	0.3553	0.3376	0.3312

individualized ability estimates, and thus our choice not to use them outside of the development set, the non-individualized parameters may be better learned when individualized ability estimates are present and thus still benefit a student hold-out evaluation. This benefit was observed by (Yudelson et al., 2013) when learning individualized parameters in BKT validated at the student level. We tested this possibility by training dAFM expert-init with student individualized θ_j along with an additional bias term. When testing on the validation set, no student one-hot information was input. Instead, the bias term represented the average θ. The predictions from a model trained with this individualization were compared to regular dAFM expert-init.

In the Cognitive Tutor, the cognitive mastery estimate of a KC is reset every section. Because of this, the KCs provided can be thought of as pertaining specifically to each section, and thus could be tokenized as the concatenation of section and KC. Section information has proven useful in several of the top approaches to predicting responses in the Cognitive Tutor (Stamper and Pardos, 2016). We tested the utility of incorporating section information into dAFM expert-init by comparing the model with no section information to two variants; one which added a one-hot of section to the concatenation layer (where the bias and other terms are merged), and another where a KC associated with an item was concatenated with the section number the item appeared in, leading to a mild increase in the total number of KCs (only mild because many KCs are contained in only a single section). This was not a topology variation, rather an increase in the dimensionality of the layers to correspond to the new k.

7. Results

The results in this section report development set prototype model results as well as the degree to which the dAFM model variants generalized from the training to the test sets of our primary five datasets. Results of prediction on the validation set are also presented as well as an inspection of KC association refinements for items whose predictions were most improved by dAFM.

7.1. Development set prototyping results

Here we present the results of methods trialed only on the development set, *INTRO-PERIM-AREA*, from the Bridge to Algebra '08-'09 dataset. As with the other datasets, this unit was split into train, validation, and test sets; however, since these experiments were conducted during the development stage, only predictions on the validation set were made.

Individualized Student Ability θ_j: We explored if refining the Q-matrix while modeling individualized ability estimates lead to any improvements in the learning of the Q-matrix evident in the prediction results on the validation hold-out. Table 3 shows that this was largely not the case. The individualized parameters only lead to better prediction in one of the six models, QkDense, and in that model, reducing error by 0.0002.

Adding Section Information: Section information has helped previous performance prediction models in the response prediction task; however, Table 4 shows that both section one-hot

Table 4: The effect of section information on the predictive performance for dAFM models.

Section	AFM	FT	QkDense	random-init	expert-init	rounded FT
No Section	**0.3314**	**0.3308**	**0.3288**	**0.3427**	**0.3314**	**0.3312**
KC-section	0.3315	0.3310	0.3289	0.3422	0.3315	**0.3312**
Section on-hot	0.3317	0.3315	0.3299	0.3434	0.3317	0.3316

Table 5: Comparison of dAFM models with linear and sigmoid activation functions

Models	Linear	Sigmoid
AFM	0.3314	0.3314
fine-tuned	**0.3308**	0.3312
QkDense	**0.3288**	0.3341
random-init	0.3560	**0.3422**
expert-init	0.3373	**0.3314**
rounded FT	**0.3312**	0.3698

and KC-section token concatenation produced poorer predictions in all six dAFM models.

Alternative q_k Activation Function: A sigmoid activation for the q_k layer would restrict the range of values to between 0 and 1, and potentially lead to a more interpretable model than with a linear, pass-through function. Table 5 shows that the best model is dAFM fine-tuned with linear activation and that sigmoid is the better activation in two of the models (random and expert init).

Using Continuous Representation Models to Derive the Q-matrix: Both DKT and Skip-gram models can embed questions into a vector space. Given the pre-requisite relationships that were found in DKT (Piech et al., 2015) and the KC imputation ability of embedding with Skip-grams (Pardos and Dadu, 2017), it was conceivable that the embedding space might provide fertile representational grounds for constructing a Q-matrix. Clustering the space into different sizes of k using these models; however, did not produce a better predicting Q-matrix than the dAFM models (except for random-init), as seen in Table 6. The best size k for DKT and Skip-gram were the original KC model k (using rnn-incorrect item representations) and k-k/10, respectively. While this was a null result, as neither method beat even the base AFM model, outperforming the randomly initialized Q-matrix model of dAFM random-init suggests they may have better promise in learning a Q-matrix from the ground up than doing so within the framework of dAFM.

Table 6: Comparison of continuous representation learning approaches to dAFM models

DKT	skip-gram	AFM	FT	QkDense	random	expert	rounded-FT
0.3472	0.3475	0.3314	0.3312	**0.3286**	0.3553	0.3376	0.3312

Table 7: Results of dAFM and AFM model prediction performance with RMSE averaged across all students in the validation set for our primary datasets. The best model per dataset is in bold

Dataset	AFM	FT	QkDense	random	expert	round-FT
Geometry	0.4715	**0.4594**	0.4716	0.4712	0.4601	0.4709
ASSISTments 09-10	0.4008	**0.3757**	0.3965	0.3857	0.3787	0.4080
ASSISTments 12-13	0.2971	**0.2726**	0.2884	0.2839	0.2758	0.2883
CogTutor Bridge 06-07	0.3603	0.3651	**0.3596**	0.4104	0.3665	0.3714
CogTutor Bridge 08-09	0.3672	**0.3595**	0.3667	0.3918	0.3772	0.3692

Table 8: Results of dAFM and AFM model prediction performance with RMSE averaged across all students in the test set for our primary datasets. The best model per dataset is in bold

Dataset	AFM	FT	QkDense	random	expert	round-FT
Geometry	0.4252	**0.4129**	0.4307	0.4191	0.4207	0.4143
ASSISTments 09-10	0.4064	**0.3810**	0.4014	0.3913	0.3844	0.4127
ASSISTments 12-13	0.3020	**0.2797**	0.2940	0.2907	0.2824	0.2945
CogTutor Bridge 06-07	0.3502	0.3573	**0.3498**	0.3954	0.3590	0.3625
CogTutor Bridge 08-09	0.3629	**0.3552**	0.3626	0.3870	0.3687	0.3650

7.2. Main Prediction Results

Results of the six dAFM models' predictions on our five primary datasets are shown in Table 7 (validation) and Table 8 (test). The AFM model (dafm-afm) represents the baseline standard model with its expert Q-matrix, while expert (expert-init), FT (fine-tuned), and round-FT (rounded fine-tuned), represent dAFM's attempt at refining the expert model using different training regimes. Only random (random-init) represents a Q-matrix model learned from the ground-up in dAFM and QkDense represents the learning of a fuzzy (non-binary) skill merging. Comparing the base AFM model to random, AFM generalizes better than the ground-up learned Q-matrix model in the large Cognitive Tutor Bridge datasets, but AFM is worse in the other three in both the validation and test set predictions. Comparing a random to an expert initialized Q-matrix, the expert refined model outperforms the KC model learned from the ground-up in all cases except for on the validation set of Geometry. Like the random model, the expert-init dAFM model is better than AFM in three datasets, but also underperforms it in the large Cognitive Tutor datasets. If the learned AFM model, including its Q-matrix, is fine-tuned, the resultant model shows improvement in all datasets except for the CogTutor Bridge '06-'07 dataset and is the best model all-around in eight of the ten experiments (combining validation and test results). Rounding the weights of the fine-tuned model did not lead to improvement in any of the experiments. The performance of the base AFM model in the '06-'07 CogTutor dataset suggests a very strong expert KC model, also evidenced by having the largest difference in error between expert-init and random-init predictions. For this dataset, only the skill merging layer variant (QkDense) outperformed the base model, as it also did in all other datasets except for Geometry.

The relative results between models and datasets in the validation and test set were nearly identical, suggesting that our hyperparameter search and evaluation methodology did not overfit the validation set. The Adagrad optimizer produced the best scoring models for all datasets except for the much smaller Geometry, where Adam was best.

Table 9: Comparison of linear and ReLU activation for the Q_k layer on the Geometry dataset

Models	Linear (validation)	ReLU (validation)	Linear (test)	ReLU (test)
AFM	0.4715	0.4715	0.4252	0.4252
fine-tuned	**0.4556**	0.4594	**0.4114**	0.4129
QkDense	**0.4695**	0.4716	**0.4268**	0.4307
random-init	0.4717	**0.4712**	**0.4145**	0.4191
expert-init	**0.4582**	0.4601	0.4232	**0.4207**
rounded FT	**0.4694**	0.4709	0.4200	**0.4143**
Average	0.4660	0.4674	0.4202	0.4205

7.3. Qualitative Study of Q-matrix refinements

In this section, we look into the dAFM refinements made to the Q-matrix. We break down the dAFM fine-tuned predicted performance by problem on the test set and then sort by problems most improved by the Q-matrix refinement from the original expert model (Table 10). The most improved problem is selected from The Cognitive Tutor Geometry dataset, which has imagery of its 40 problems. We show a screenshot of the problem, its old and dAFM refined KC mapping, and reflect on the interpretability and plausibility of the refinement.

Since we are concerned with interpretability over predictive performance here, we consider an activation function for q_k not yet discussed. The linear (pass-through) activation function used in the dAFM models has the undesirable property (wrt. interpretation) of allowing for negative values in the Q-matrix. Avoiding negatives was the rationale for (Desmarais et al., 2011) in choosing non-negative matrix factorization to derive the Q-matrix in their testing scenario. In the development set, we explored using a sigmoid function to limit the range of outputs to between 0 and 1, but this led to convergence issues due to needing to set high magnitude positive and negative weight values to represent an initial Q-matrix, due to the infinite domain of the sigmoid function. To avoid this, we sought a function that would prohibit negatives, but have output linear with its input for values 0 and above. A rectified linear unit (ReLU) fit this description perfectly. We tested this activation with dAFM FT compared with linear to make sure we were not sacrificing too much predictive gain in replacing the linear activation. We found that very little was lost, with an average RMSE of 0.4205 compared to linear's 0.4202 on the test set (Table 9).

7.3.1. Geometry Problem Presentation

The problem *Triangle Rectangle* was reported in Table 10 as having the highest increase in predictive accuracy when predicted in the test set using the dAFM fine-tuned Q-matrix model. It was most improved in terms of both percentage and absolute point reduction in error and was therefore the chosen problem for presentation.

Figure 6 shows the *Triangle Rectangle* problem. The problem statement asks for the area of a scalene triangle, EBD, enclosed in a rectangle whose dimensions are provided. The table below the problem statement shows the three items (steps) associated with the problem. The last item represents the main question posed in the problem statement. The second item is filled in by the tutor, as its value is given in the problem statement. The first item asks for the length of the base of the scalene triangle. This item was originally tagged with the KC *compose-by-*

Table 10: Top-2 improved problems when comparing predictions using the dAFM fine-tuned model and the original expert model.

Evaluation Type	Most Improved Problem Name	Improvement Magnitude
Test set (Absolute)	Triangle Rectangle	0.2828
	Two Circles in a Square	0.1446
Test set (Percent)	Triangle Rectangle	38.29%
	Circle Radius	37.81%

multiplication. In dAFM, it has mainly been re-mapped to identifying *triangle-side* (weight value = 1.05), shown in Figure 7. Three other KCs are also involved in the mapping with weight values greater than 0: *compose-by-multiplication* (0.19), *trapezoid-height* (0.12), and *pentagon-side* (0.08). The last problem asks for the area of the scalene triangle, EBD, and was tagged with *triangle-area* in the original Q-matrix. In dAFM, this problem has been associated with *triangle-side* (0.29), *trapezoid-height* (0.22), *trapezoid-area* (0.18), and *triangle-area* (0.11). The full list of candidate KCs in the Geometry dataset is shown in Table 11.

These refinements suggested an emphasis on identification of the sides of the triangle and of the rectangle over computing the area in solving the problems. They also favored mapping to the trapezoid side and height KCs in favor of the more precise respective parallelogram KCs. An inspection of the tagging of parallelogram questions in other problems would be needed to derive a rationale for this. The *pentagon-side* association appears to be spurious, not strictly related to the problem but perhaps correlated to performance in it, and correspondingly had the lowest coefficient of association (0.08).

Table 11: List of 15 Knowledge Component in the Geometry dataset

circle-area	compose-by-multiplication	trapezoid-area
circle-circumference	parallelogram-area	trapezoid-base
circle-diameter	parallelogram-side	trapezoid-height
circle-radius	pentagon-area	triangle-area
compose-by-addition	pentagon-side	triangle-side

8. LIMITATIONS

The design of dAFM focused on refinement of item mappings to KCs and, separately, KC merging (with QkDense). It does not, as described, support the expansion of the number of KCs, though contraction is possible if all items become unmapped from a KC and no new mappings are made to it. The described model will also be limited in the predictive accuracy it is capable of. Like the original AFM model, dAFM never observes the correctness of a response to an item in its input, only as a label in its output. This is in contrast to Bayesian Knowledge Tracing, Performance Factors Analysis, and Deep Knowledge Tracing, which do observe correctness in their inputs. Correctness information can contribute to the estimation of individual abilities in AFM and subsequently improve prediction performance, but under student-level validation such as in real-world tracing scenarios, where re-training or online updates may not be practical or

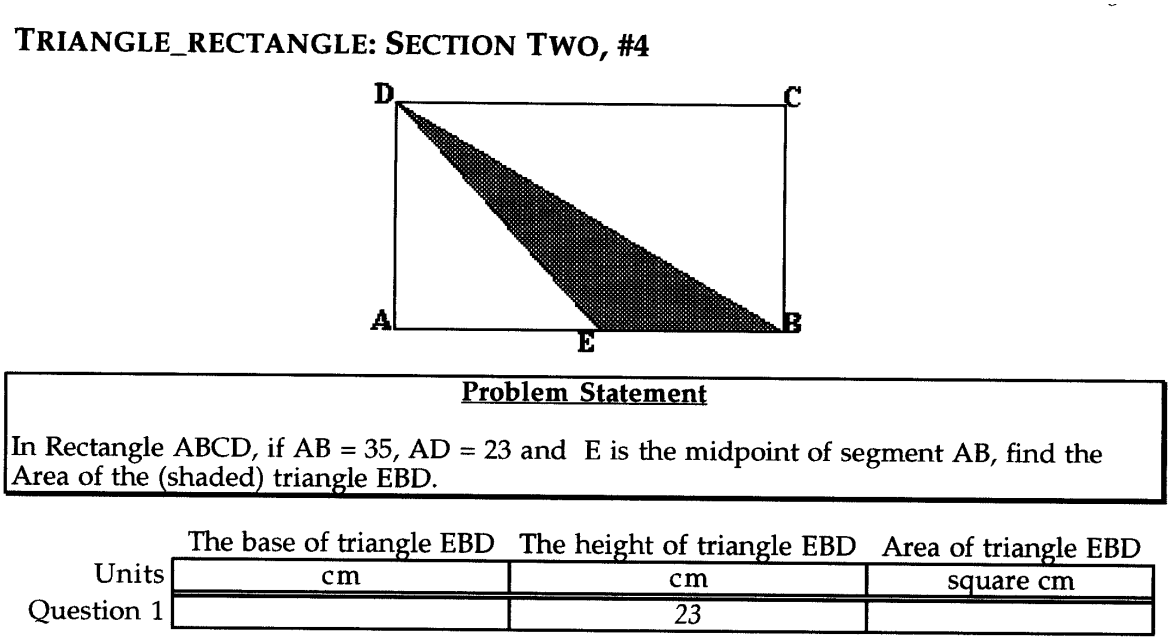

Figure 6: Screenshot of the top most improved Cognitive Tutor for Geometry problem (TRIANGLE RECTANGLE) on the test set. Prompts for answers to two items (or steps of 'Question 1') are shown in the table at the bottom of the screenshot.

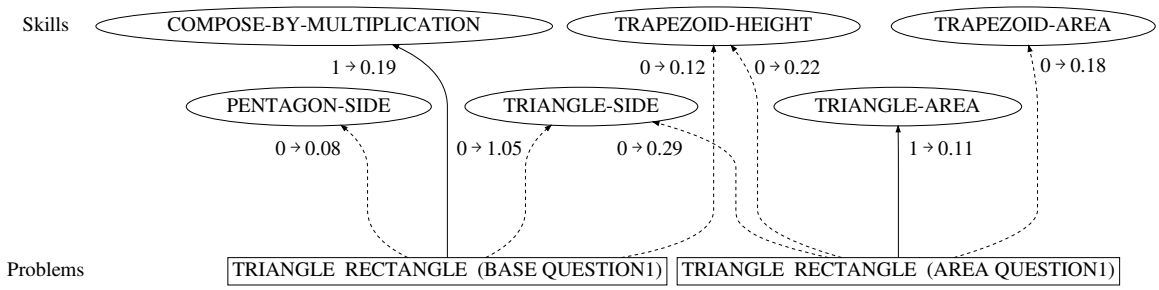

Figure 7: Plot showing expert and dAFM refined KC associations of items (bottom) to KCs (top) in the problem called TRIANGLE RECTANGLE in The Cognitive Tutor for Geometry. Solid black lines denote the original mapping, dotted lines denote the refined mapping, and the decimal values followed by an arrow and another decimal value represent the original KC association value and the dAFM refined value.

feasible, the model will predict according to the average growth curve modeled of the KCs. BKT will similarly project a performance curve, given only its four KC parameter values and no observed responses for inference.

It is unclear what class of problem Q-matrix refinement from observational data is. With elements of attribution, credit, and blame assignment, it may very well be a problem adjacent to or as hard as problems of causal inference (Shaver, 2012). With that in mind, refinements ought to be validated by other means in addition to predictive generalization before it is better understood what the conditions are under which valid inference can be made.

Lastly, we note that the prototype model results were run on a subset of the data (the development set), and thus the generalizability of these results is less supported than our more robust primary dataset results.

9. CONTRIBUTIONS

We contributed a new framework, dAFM, to the family of models used to study data-driven refinement of a Q-matrix. Through careful design, we reconstructed AFM as a neural network, having the same free parameters as the original model, but with the added ability to adjust the associations of items to KCs in its Q-matrix. Among our experiments, we found a two-phase training process to be the best practice for refinement of the Q-matrix. First, fitting the base parameters of AFM with the original expert Q-matrix unchanged, followed by fine-tuning the base parameters while allowing the Q-matrix associations to also be fit to data. In four of the five datasets from ASSISTments and Cognitive Tutors, this fine-tuning procedure produced a Q-matrix leading to better predictive generalization than the original (answering RQ1: does a dAFM refined Q-matrix outperform an expert model?). In the one dataset, Bridge to Algebra '06-'07, where Q-matrix fine-tuning did not improve generalization, a learned transformation among the KCs (i.e., merging via an added network layer) did produce predictions better than the original.

Our second research question was if a Q-matrix learned from the ground-up would generalize better than one refined from an existing expert model. We found this to be decidedly not the case, with the ground-up model (random-init) underperforming the fine-tuned model in all ten dataset experiments. The ground-up model did, however, outperform the (unmodified) original AFM expert model in three of the datasets (the two ASSISTments sets and Geometry). Its underperformance compared with expert models from the Bridge to Algebra datasets of the Cognitive Tutor speaks to the rigor applied in the specification of those KC models, including the internal refinements of the model conducted in that system.

We addressed the interpretation problem of negative Q-matrix values by adding a rectified linear unit (ReLU) activation, which kept values positive with very little hit to generalization performance. The sigmoid function, which would accomplish a similar goal, showed a greater hit to performance when prototyped in our development dataset. Also prototyped in the development set was the addition of individualized θ_i, which did not lead to learning of a more generalizable Q-matrix. The incorporation of curriculum structure information, in the form of sections in the Cognitive Tutor, also lead to no improvement. Lastly, we prototyped extracting a Q-matrix from models using continuous representations of items. These representations were sourced from different layers of DKT and Skip-gram embeddings of items. These representations, discretized through clustering, did not serve as a more generalizable Q-matrix than the original expert model; however, they did outperform the Q-matrix learned from the ground up (random-init) in dAFM.

ACKNOWLEDGEMENT

We thank Carnegie Learning and the ASSISTments Platform for their public sharing of datasets to advance learning sciences research. This material is based upon work supported by the National Science Foundation under Grant No. 1547055.

REFERENCES

BURRELL, J. 2016. How the machine thinks: Understanding opacity in machine learning algorithms. *Big Data & Society 3,* 1.

CEN, H., KOEDINGER, K., AND JUNKER, B. 2005. Automating cognitive model improvement by A* search and logistic regression. In *Proceedings of AAAI 2005 Educational Data Mining Workshop*, J. E. Beck, Ed.

CEN, H., KOEDINGER, K., AND JUNKER, B. 2006. Learning factors analysis: A general method for cognitive model evaluation and improvement. In *Proceedings of the 8th International Conference on Intelligent Tutoring Systems*, M. Ikeda, K. Ashley, and T.-W. Chan, Eds. Springer-Verlag, 164–175.

CEN, H., KOEDINGER, K. R., AND JUNKER, B. 2007. Is over practice necessary? Improving learning efficiency with the Cognitive Tutor through educational data mining. In *Proceedings of 13th International Conference on Artificial Intelligence in Education*, K. Ashley and T. van Engers, Eds. IOS Press, 511–518.

CHEN, Y., GONZÁLEZ-BRENES, J. P., AND TIAN, J. 2016. Joint discovery of skill prerequisite graphs and student models. In *Proceedings of the 9th International Conference on Educational Data Mining*, M. Chi and M. Feng, Eds. 46–53.

CHIU, C.-Y. 2013. Statistical refinement of the Q-matrix in cognitive diagnosis. *Applied Psychological Measurement 37,* 8, 598–618.

CHOROMANSKA, A., HENAFF, M., MATHIEU, M., AROUS, G. B., AND LECUN, Y. 2015. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*. 192–204.

CLARK, R., FELDON, D., VANMERRIENBOER, J., YATES, K., AND EARLY, S. 2006. Cognitive task analysis. *Handbook of research on educational communications and technology 3*.

CORBETT, A. 2001. Cognitive computer tutors: Solving the two-sigma problem. In *Proceedings of the 8th International Conference on User Modeling*, M. Bauer, P. J. Gmytrasiewicz, and J. Vassileva, Eds. Springer, 137–147.

CORBETT, A. T. AND ANDERSON, J. R. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction 4,* 4, 253–278.

DESMARAIS, M. ET AL. 2011. Conditions for effectively deriving a Q-matrix from data with non-negative matrix factorization. In *Proceedings of the 4th International Conference on Educational Data Mining*, C. Conati and S. Ventura, Eds. 41–50.

DESMARAIS, M. C. AND NACEUR, R. 2013. A matrix factorization method for mapping items to skills and for enhancing expert-based Q-matrices. In *Proceedings of the 6th International Conference on Artificial Intelligence in Education*, Y. B. Kafai, W. A. Sandoval, and N. Enyedy, Eds. Springer, 441–450.

FODOR, J. A. AND PYLYSHYN, Z. W. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition 28,* 1-2, 3–71.

GONZÁLEZ-BRENES, J., HUANG, Y., AND BRUSILOVSKY, P. 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*, J. Stamper and Z. A. Pardos, Eds. 84–91.

GONZÁLEZ-BRENES, J. P. AND MOSTOW, J. 2012. Dynamic cognitive tracing: Towards unified discovery of student and cognitive models. In *Proceedings of the 5th International Conference on Educational Data Mining*, K. Yacef and O. Zaiane, Eds. ERIC.

HART, P. E., NILSSON, N. J., AND RAPHAEL, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics 4,* 2, 100–107.

HEFFERNAN, N. T. AND HEFFERNAN, C. L. 2014. The ASSISTments ecosystem: Building a platform that brings scientists and teachers together for minimally invasive research on human learning and teaching. *International Journal of Artificial Intelligence in Education 24,* 4, 470–497.

KHAJAH, M., WING, R., LINDSEY, R., AND MOZER, M. 2014. Incorporating latent factors into knowledge tracing to predict individual differences in learning. In *Proceedings of the 7th International Conference on Educational Data Mining*, J. Stamper and Z. A. Pardos, Eds. 99–106.

KOEDINGER, K. R., D'MELLO, S., MCLAUGHLIN, E. A., PARDOS, Z. A., AND ROSÉ, C. P. 2015. Data mining and education. *Wiley Interdisciplinary Reviews: Cognitive Science 6,* 4, 333–353.

KOEDINGER, K. R., MCLAUGHLIN, E. A., AND STAMPER, J. C. 2012. Automated student model improvement. In *Proceedings of the 5th International Conference on Educational Data Mining*, K. Yacef and O. Zaiane, Eds. 17–24.

KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, L. B. F. Pereira, C. J. C. Burges and K. Q. Weinberger, Eds. 1097–1105.

LECUN, Y., BENGIO, Y., AND HINTON, G. 2015. Deep learning. *Nature 521,* 7553, 436–444.

LEVY, O. AND GOLDBERG, Y. 2014. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the 18th Conference on Computational Natural Language Learning*, M. Kay, Ed. 171–180.

LIU, J., XU, G., AND YING, Z. 2012. Data-driven learning of Q-matrix. *Applied psychological measurement 36,* 7, 548–564.

LIU, R., PATEL, R., AND KOEDINGER, K. R. 2016. Modeling common misconceptions in learning process data. In *Proceedings of the 6th International Conference on Learning Analytics & Knowledge*, S. Dawson, H. Drachsler, and C. P. Rosé, Eds. ACM, 369–377.

MARTIN, B., MITROVIC, A., KOEDINGER, K. R., AND MATHAN, S. 2011. Evaluating and improving adaptive educational systems with learning curves. *User Modeling and User-Adapted Interaction 21,* 3, 249–283.

MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G. S., AND DEAN, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds. 3111–3119.

NATHAN, M. J., KOEDINGER, K. R., AND ALIBALI, M. W. 2001. Expert blind spot: When content knowledge eclipses pedagogical content knowledge. In *Proceedings of the 3rd International Conference on Cognitive Science*, J. Mu, Q. Liang, W. Wang, B. Zhang, and Y. Pi, Eds. Beijing: University of Science and Technology of China Press, 644–648.

PARDOS, Z. A. 2017. Big data in education and the models that love them. *Current Opinion in Behavioral Sciences 18*, 107–113.

PARDOS, Z. A. AND DADU, A. 2017. Imputing KCs with representations of problem content and context. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, F. Cena and M. Desmarias, Eds. ACM, 148–155.

PAVLIK JR, P. I., CEN, H., AND KOEDINGER, K. R. 2009a. Learning factors transfer analysis: Using learning curve analysis to automatically generate domain models. In *Proceedings of the 2nd International Conference on Educational Data Mining*, T. Barnes and M. Desmarias, Eds. 121–130.

PAVLIK JR, P. I., CEN, H., AND KOEDINGER, K. R. 2009b. Performance factors analysis–a new alternative to knowledge tracing. In *Proceedings of the 14th International Conference on Artificial Intelligence in Education*, D. Dicheva and D. Dochev, Eds. IOS Press, 531–538.

PELÁNEK, R. 2015. Metrics for evaluation of student models. *Journal of Educational Data Mining 7,* 2, 1–19.

PELÁNEK, R. 2017. Bayesian knowledge tracing, logistic models, and beyond: an overview of learner modeling techniques. *User Modeling and User-Adapted Interaction 27,* 3-5, 313–350.

PIECH, C., BASSEN, J., HUANG, J., GANGULI, S., SAHAMI, M., GUIBAS, L. J., AND SOHL-DICKSTEIN, J. 2015. Deep knowledge tracing. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds. 505–513.

RASCH, G. 1961. On general laws and the meaning of measurement in psychology. In *Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability*, J. Neyman, Ed. Vol. 4. 321–333.

RITTER, F. E. AND SCHOOLER, L. J. 2001. The learning curve. *International Encyclopedia of the Social and Behavioral Sciences 13*, 8602–8605.

RITTER, S., ANDERSON, J. R., KOEDINGER, K. R., AND CORBETT, A. 2007. Cognitive Tutor: Applied research in mathematics education. *Psychonomic Bulletin & Review 14,* 2, 249–255.

ROSÉ, C. P., DONMEZ, P., GWEON, G., KNIGHT, A., JUNKER, B., COHEN, W. W., KOEDINGER, K. R., AND HEFFERNAN, N. T. 2005. Automatic and semi-automatic skill coding with a view towards supporting on-line assessment. In *Proceedings of the 12th International Conference on Artificial Intelligence in Education*, G. McCalla and C. Looi, Eds. IOS Press, 571–578.

RUMELHART, D. E., HINTON, G. E., AND WILLIAMS, R. J. 1986. Learning representations by back-propagating errors. *Nature 323*, 533–536.

SAHEBI, S., LIN, Y.-R., AND BRUSILOVSKY, P. 2016. Tensor factorization for student modeling and performance prediction in unstructured domain. In *Proceedings of the 9th International Conference on Educational Data Mining*, M. Chi and M. Feng, Eds. 502–506.

SCHEINES, R., SILVER, E., AND GOLDIN, I. M. 2014. Discovering prerequisite relationships among knowledge components. In *Proceedings of the 7th International Conference on Educational Data Mining*, J. Stamper and Z. A. Pardos, Eds. 355–356.

SHAVER, K. G. 2012. *The attribution of blame: Causality, responsibility, and blameworthiness*. Springer Science & Business Media.

STAMPER, J. AND PARDOS, Z. A. 2016. The 2010 KDD cup competition dataset: Engaging the machine learning community in predictive learning analytics. *Journal of Learning Analytics 3,* 2, 312–316.

SUN, Y., YE, S., INOUE, S., AND SUN, Y. 2014. Alternating recursive method for Q-matrix learning. In *Proceedings of the 7th International Conference on Educational Data Mining*, J. Stamper and Z. A. Pardos, Eds. 14–20.

TATSUOKA, K. K. 1983. Rule space: An approach for dealing with misconceptions based on item response theory. *Journal of educational measurement 20,* 4, 345–354.

WILLIAMS, R. J. AND ZIPSER, D. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation 1,* 2, 270–280.

YUDELSON, M. V., KOEDINGER, K. R., AND GORDON, G. J. 2013. Individualized Bayesian knowledge tracing models. In *Proceedings of 16th International Conference on Artificial Intelligence in Education*, H. Lane, K. Yacef, J. Mostow, and P. Pavlik, Eds. Springer, 171–180.

APPENDIX

This section presents the programmatics of the dAFM fine-tuned model specification. Full examples with documentation of this and the other dAFM variants in the paper can be found at http://github.com/CAHLR/dAFM.

The dAFM fine-tuned model trains in two phases. First, in which the Q-matrix (W_{Qk}) is non-trainnable while the other parameters are allowed to train and second, in which all the parameters can train without any restriction. The code, written in Keras, for the fine-tuned model is shown below:

Phase 1: W_{Qk} is non-trainable.

```
B_k = TimeDistributed(Dense(skills, activation='linear',
   use_bias=False, trainable=True), name="B_k")(virtual_input1)
t_k = TimeDistributed(Dense(skills, activation='linear',
   use_bias=False, trainable=True), name="t_k")(virtual_input1)
bias_layer = TimeDistributed(Dense(1, activation='linear',
   use_bias=False, kernel_initializer=initializers.Zeros(),
   trainable=True), name="bias")(virtual_input1)

Q_k = TimeDistributed(Dense(skills, activation=activation,
   kernel_initializer=self.f(Q_k_initialize), use_bias=False,
   trainable=False), name="Q_k")(step_input)
Qk_mul_Bk = multiply([Q_k, B_k])
sum_Qk_Bk = TimeDistributed(Dense(1, activation='linear',
   trainable=False, kernel_initializer=initializers.Ones(),
   use_bias=False), name="sum_Qk_Bk")(Qk_mul_Bk)

T_k = SimpleRNN(skills, kernel_initializer=initializers.Identity(),
   recurrent_initializer=initializers.Identity() , use_bias=False,
   trainable=False, activation='linear', return_sequences=True,
   name="T_k")(Q_k)

Qk_mul_Tk_mul_tk = multiply([Q_k, T_k, t_k])
sum_Qk_Tk_tk = TimeDistributed(Dense(1, activation='linear',
   trainable=False, kernel_initializer=initializers.Ones(),
   use_bias=False), name="sum_Qk_Tk_tk")(Qk_mul_Tk_mul_tk)

Concatenate = concatenate([bias_layer, sum_Qk_Bk, sum_Qk_Tk_tk])
output = TimeDistributed(Dense(1, activation="sigmoid",
   trainable=False, kernel_initializer=initializers.Ones(),
   use_bias=False), name="output")(Concatenate)

model1 = Model(inputs=[virtual_input1, step_input], outputs=output)
d_optimizer = {"rmsprop":optimizers.RMSprop(lr=learning_rate),
   "adam":optimizers.Adam(lr=learning_rate),
   "adagrad":optimizers.Adagrad(lr=learning_rate) }
model1.compile( optimizer = d_optimizer[optimizer],
                  loss = self.custom_bce)
```

Phase 2: W_{Qk} is trainable and the trained weights of the other parameters obtained from Phase 1 are used to intialize the respective parameters.

```
B_k = TimeDistributed(Dense(skills, activation='linear',
   kernel_initializer=self.f(model1.get_layer("B_k").get_weights()[0]),
   use_bias=False), name="B_k")(virtual_input1)
t_k = TimeDistributed(Dense(skills, activation='linear',
   kernel_initializer=self.f(model1.get_layer("t_k").get_weights()[0]),
   use_bias=False), name="t_k")(virtual_input1)
bias_layer = TimeDistributed(Dense(1, activation='linear',
   use_bias=False,
   kernel_initializer=self.f(model1.get_layer("bias").get_weights()[0]),
   trainable=True), name="bias")(virtual_input1)
Q_k = TimeDistributed(Dense(skills, activation=activation,
   kernel_initializer=self.f(Q_k_initialize), use_bias=False,
   trainable=False), name="Q_k")(step_input)
```