Towards Personalized Learning using Counterfactual Inference for Randomized Controlled Trials

by

Siyuan Zhao

A Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Computer Science

by

May 2018

APPROVED:

____

Professor Neil Heffernan, Advisor

____

Professor Joseph Beck, Committee member

____

Professor Jacob Whitehill, Committee member

____

Professor Adam Sales, External Committee Member, University of Texas, Austin

____

Dr. Adam Kalai, External Committee Member, Microsoft Research

Abstract

Personalized learning considers that the causal effects of a studied learning intervention may differ for the individual student (e.g., maybe girls do better with video hints while boys do better with text hints). To evaluate a learning intervention inside ASSISTments, we run a randomized control trial (RCT) by randomly assigning students into either a control condition or a treatment condition. Making the inference about causal effects of studies interventions is a central problem. Counterfactual inference answers What if questions, such as "Would this particular student benefit more if the student were given the video hint instead of the text hint when the student cannot solve a problem?". Counterfactual prediction provides a way to estimate the individual treatment effects and helps us to assign the students to a learning intervention which leads to a better learning.

A variant of Michael Jordan's "Residual Transfer Networks" was proposed for the counterfactual inference. The model first uses feed-forward neural networks to learn a balancing representation of students by minimizing the distance between the distributions of the control and the treated populations, and then adopts a residual block to estimate the individual treatment effect.

Students in the RCT usually have done a number of problems prior to participating it. Each student has a sequence of actions (performance sequence). We proposed a pipeline to use the performance sequence to improve the performance of counterfactual inference. Since deep learning has achieved a huge amount of success in learning representations from raw logged data, student representations were learned by applying the sequence autoencoder to performance sequences. Then, incorporate these representations into the model for counterfactual inference. Empirical results showed that the representations learned from the sequence autoencoder improved the performance of counterfactual inference.

Acknowledgements

First and foremost I would like to thank my advisor, Prof. Neil Heffernan, for his support of my Ph.D. study.

Besides my advisor, my thanks are also due to the rest of my committee members: Prof. Joseph Beck, Prof. Jacob Whitehill, Prof. Adam Sales, and Dr. Adam Kalai. Their insightful comments and the challenging questions incented me to work hard and widen my knowledge.

I would like to thank my lab mates at ASSISTments Lab. I had a good time working with them.

Last but not the least, I would like to express my gratitude to my family for their warm love and endless support. My parents always encouraged and supported me at every stage of my personal and academic life.

Contents

List of Figures

List of Tables

Part I

Using Deep Learning for Student Modeling

Chapter 1

Going Deeper with Deep Knowledge Tracing

Proper citation of this chapter is as follows:

Xiaolu Xiong, Siyuan Zhao, Eric Van Inwegen, and Joseph Beck. Going deeper with deep knowledge tracing. In *Proceedings of the 9th International Conference on Educational Data Mining, EDM 2016*

Over the last couple of decades, there have been a large variety of approaches towards modeling student knowledge within intelligent tutoring systems. With the booming development of deep learning and large scale artificial neural networks, there have been empirical successes in a number of machine learning and data mining applications, including student knowledge modeling. Deep Knowledge Tracing (DKT), a pioneer algorithm that utilizes recurrent neural networks to model student learning, reports substantial improvements in prediction performance. To help the EDM community better understand the promising techniques of deep learning, we examine DKT alongside of two well-studied models for knowledge modeling, PFA and BKT. In addition to sharing a primer on the internal computational structures of DKT, we also report on potential issues that arise from data formatting. We take steps to reproduce the experiments of Deep Knowledge Tracing by implementing a DKT algorithm using Google's TensorFlow framework; we also reproduce similar results on new datasets. We determine that the DKT findings don't hold an overall edge when compared to the PFA model, when applied to properly prepared datasets that are limited to main (i.e. non-help) questions. More importantly, during the investigation of DKT, we not only discovered a data quality issue in a public available data set, but we also detected a vulnerability of DKT at how it handles multiple skill sequences.

1.1 Introduction

Deep knowledge tracing (DKT), the recent adoption of RNN in the area of educational data mining, achieved dramatic improvement over well-known Bayesian knowledge tracing models and the results of it have been demonstrated to be able to discover the latent structure in skill concepts and can be used for curriculum optimization [PBH$^+$15]. One major question in curriculum design is the dependencies of skills. That is, should skill "A" be taught before skill "B", the other way around, or does it even matter? Human curriculum experts have their theories; educational psychologists may run trials to examine the dependencies. Knowledge estimate models (such as the well-known Bayesian knowledge tracing) are run with the human input of the skill dependencies. One of the powers of unsupervised learning systems such as Deep RNN's is that they can be run without the skill dependencies theories. As a natural part of model creation, the Deep RNN will create its own rules for skill dependencies. These dependencies can then be used for curriculum optimization. It has been well recognized that the power of deep learning comes from the fact that a deep learning algorithm is a particular kind of representation learning procedure that discovers multiple levels of representation, with higher-level features representing more abstract aspects of the data; DKT shows a key advantage that it does not require human expert annotations and can take advantage of any student input that can be vectorized.

Driven by both noble goals (testing the reproducibility of scientific findings) and some selfish ones (how did they do so much better at predicting student performance?!), we set out to take the theories, algorithms, and code from the DKT paper and apply them ourselves to the same data and more data sets. As to the goal of reproducing the findings, we were motivated by studies discussing the importance of reproducibility [C$^+$15]. In addition to applying DKT to the same data, we also tested the algorithm on a different new ASSISTments dataset (covers data in 2014-2015 school year), as well as the one of data sets from KDD Cup 2010 data set. In our experiments with the original DKT algorithm, we uncovered three aspects of the ASSISTments 2009-2010 dataset that, when accounted for, drastically reduce the effectiveness of the deep knowledge tracing algorithms. These can broadly be summarized as 1). an error in reporting the data (wherein rows of data were randomly duplicated). 2). an inconsistency of skill tagging, and 3). the use of information ignored by PFA and BKT. We will discuss these three inconsistencies and their impacts on the prediction accuracy in section 1.3.

1.2 Deep Knowledge Tracing and Other Student Modeling Techniques

Unlike the conventional understanding of using multiple processing layers so neural networks can be described as ”deep”, DKT algorithm uses recurrent neural networks that are ”deep” in time to take the task of modeling knowledge in sequential data sets. This family of models represents latent knowledge state, along with its temporal dynamics, using large vectors of artificial neurons, and allows the latent variable representation of student knowledge to be learned from data rather than hard-coded.

Typical RNNs suffer from the now famous problems of vanishing and exploding gradients. Figure 1.1 shows an unrolled RNN, there are loops at hidden layers, allowing information to retain. RNN can go deep in terms of time sequence. When standard activation functions, cumulative backpropagation error signals either shrink rapidly, or grow out of bounds. In fact, they decay exponentially in the number of layers, or they explode. Long short-term memory (LSTM) model [HS97] is introduced to deal with vanishing gradients problem and it also achieves remarkable results on many previously un-learnable tasks. LSTM, a variation of recurrent neural network, contains LSTM units in addition to regular RNN units. LSTM units have two unique gates: forget and input gates to determine when to forget previous information, and which current information is important to remember.

In the DKT algorithm, at a certain time step, the input to RNNs is the student performance on the skill that the student is currently working on. Since RNNs only accept a fixed length of vector as the input, we used one-hot encoding to convert student performance into a fixed length of vector whose all elements are 0s except for a single 1. The single 1 in the vector indicates two things: which

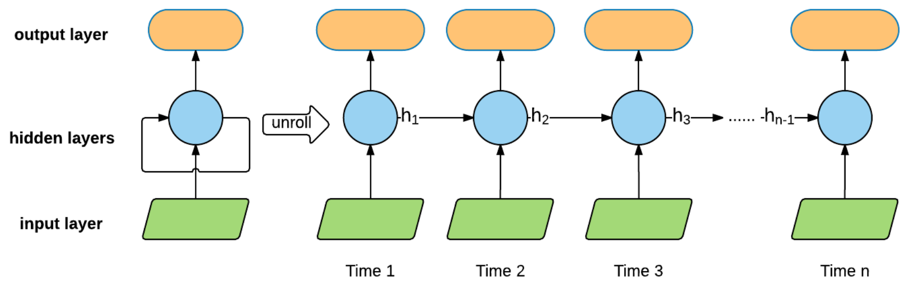

Figure 1.1: An illustration of baseline LSTM model for AES

skill was answered and if the skill was answered correctly. This data presentation draws a clear distinction between DKT and other student modeling methods, such as Bayesian Knowledge Tracing and Performance Factor Analysis.

The Bayesian Knowledge Tracing (BKT) model [CA94] is a 2-state dynamic Bayesian network where student performance is the observed variable and student knowledge is the latent. The model takes student performances and uses them to estimate the student level of knowledge on a given skill. The standard BKT model is defined by four parameters: initial knowledge and learning rate (learning parameters) and slip and guess (mediating parameters). The two learning parameters can be considered as: the likelihood the student knows the skill before he even starts on an assignment (initial knowledge, K0) and the probability a student will acquire a skill as a result of an opportunity to practice it (learning rate). The guess parameter represents the fact that a student may sometimes generate a correct response in spite of not knowing the correct skill. The slip parameter acknowledges that even students who understand a skill can make an occasional mistake. Guess and slip can be considered analogous to false positive and false negative. BKT typically uses the Expectation Maximization algorithm to estimate these four parameters from training data. Based on the estimated knowledge, student performance at a particular practice opportunity can be calculated except the very first one, which only apples the value of K0.

Skills vary in difficulties and amount of practices needed to master, so values for four BKT parameters are skill dependent. This lead to one major weakness of BKT [GBH10], it lacks the ability of handling multi-skill questions since it works by looking at historical observation of a skill and cannot accommodate all skills simultaneously. One simple workaround is treating the multiple skill combination as a new joint skill and estimate a set of parameters for this new skill. Another common solution of this issue is to associate the performance on multiple skill questions with all required skills, by listing the performance sequence repeatedly [HBM00]. This makes the model see this piece of evidence multiple times for each one of required skills. As a result, a multiple skill question is multiple single skill questions. Another simpler workaround is treating a multiple skill combination as a new joint skill and train a set of parameters for this joint skill. It is important to note that the different approaches of handling multiple skill questions is a debatable issue for DKT method too, as discussed in section 1.3.

Another popular student modeling approach is the Performance Factors Analysis Model (PFA) [PJCK09]. PFA is a variant of learning decomposition, and it based on reconfiguring Learning Factor Analysis. Unlike, BKT, it has the ability to handle multiple skill questions. Briefly speaking, it uses the form of standard logistic regression model with the student performance as dependent variable. It reconfigures LFA [CKJ06] on its independent variables, by dropping the student variable and replaces the skill variable with question identity. This model estimates parameters for each item's difficulty and also two parameters for each skill reflecting the effects of the prior correct and incorrect responses achieved for that skill. Previous work that compares KT and PFA have shown that PFA to be the superior one. One reason is due the richer feature set that PFA can utilize and the fact that learning decomposition models is ensured to reach global maxima while the typical fitting approach of BKT is no guarantee of finding a global, rather than a local maximum.

1.3 Methodology and Datasets

1.3.1 Implement DKT in Tensorflow

The original version of DKT (Lua DKT [1]) was implemented in Lua scripting language using Torch framework and its source code has been released to the public. In order to have comprehensive understanding of the DKT model, we decided to replicate and implement DKT model in Python and utilize Google's TensorFlow API to help us with building neural networks. TensorFlow is Google Brain's second generation machine learning interface, it is flexible and can be used to express a wide variety for algorithms, including training and inference algorithms for deep neural network models, and it has been used for conducting research and for deploying machine learning system into production across many areas.

Our implementation of DKT in TensorFlow (TF DKT [2]) can be described as a directed graph, which is composed of a set of nodes. The graph represents a dataflow computation, with extensions for allowing certain nodes to maintain and update persistent state and for branching and looking control, this is crucial for allowing RNN nodes to work on sequential data. In the directed graph, each node can has zero or more inputs and zero or more outputs, and represents the instantiation of an operation. An operation represents an abstract computation. Not only TensorFlow supports low level operations like element-wise mathematical ones to assemble custom algorithms, it also has high level neural net building blocks. In-

[1]https://github.com/chrispiech/DeepKnowledgeTracing

[2]https://github.com/siyuanzhao/2016-EDM

cluding SoftMax, RNN cells and LSTM cells, built in, which means we have the luxury of using high level neural net APIs and also have the ability to plug our own loss functions into our deep learning models.

For our implementation of DKT model, we adapted the loss function of original DKT algorithm. It has 200 fully-connected hidden nodes in the hidden layer. To speed up the training process, we used mini-batch stochastic gradient descent to minimize the loss function. The batch size for our implementation is 100. For one batch, we randomly select data from 100 students in our training data. After the batch finishes training, 100 students in the batch are removed from the training data. We continue to train the model on next batch until all batches are done.

Since a deep neural network has a large number of parameters, overfitting is a serious problem during training process. To overcome this problem, we applied Dropout [SHK+14] on each hidden layer. The idea of Dropout is to randomly drop nodes from the neural network during training. Dropout has a hyper parameter p, the probability of keeping a node in the network. A small value of p means that very few of hidden nodes from each hidden layer will be selected during the training. In our implementation, p is set to be 0.6. Dropout achieves two main system improvements. First, it forces the nodes within the system to learn important signal features independent of each other; second it creates some redundancy within the system, making it more robust. As stated at beginning of Section 2, vanishing and exploding gradients are two common issues with training recurrent neural network. LSTM model is utilized to solve vanishing gradients problem.

1.3.2 Student Level Cross Validation

The next important detail is how folds of cross validation are created. In his 1995 paper [K+95] comparing cross-validation and bootstrapping, Kohavi concluded that somewhere in the range of 10-20 fold is a valid design. However, there is a unique aspect of ITS data that requires a bit more care in the selection of the folds. A given student may have a particular pattern of learning (e.g. getting the first problem totally wrong to see all of the help, but then mastering the content within the next few instances). If student data is allowed to exist in multiple folds, the model may achieve higher apparent correctness by learning a student's patterns of learning (i.e. overfitting). If the folds are created such that all of a student's data exists in only one fold, the user model can't benefit from learning a specific student's pattern of learning and is forced to generalize; the generalized user model that comes out of student-level cross validation is more likely to model unseen users better than a model that does not constrain a student to one fold. Both Lua DKT's and our research use student-level cross validation.

1.4 Datasets

1.4.1 ASSISTments 2009-2010 Dataset

The original DKT paper conducted one of three of experiments using the ASSISTments 2009-2010 skill builder data set [3]. This data set was gathered from ASSISTments' skill builder problem sets, in which a student achieves mastery by working on similar (often isomorphic) questions until they can correctly answer n right in a row (where n is usually 3). After mastery, students do not commonly rework the same skill. This dataset contains 525,535 rows of student responses; there are 4,217 student ID's and 124 skills. Lua DKT achieved an AUC of 0.86 and noticeably outperformed BKT (AUC = 0.67) on this data set. However, during our investigation

[3]https://sites.google.com/site/assistmentsdata/home/assistment-2009-2010-data/skill-builder-data-2009-2010

on the DKT source code and application, we believe we discovered three issues that have unintentionally inflated the performance of Lua DKT. These issues are:

Duplicated Records

To our surprise and dismay, we found that the ASSISTments 2009-2010 data set has a serious issue of quality: large chunks of records are duplications that should not be there for any reason (e.g., see records of order id 36369610). These duplicated rows have same information but only differ on the "opportunity" and "opportunity_original"; these two features record the number of opportunities a student have practiced on a skill and the number of practices on main problems of a skill respectively. It is impossible to have more than one "opportunity" counts for a single order id. This is definitely an error in the data set and these duplicated records should not be used in any analysis or modeling studies. We counted there are 123,778 rows of duplications out of 525,535 in the data set (23.6%). The existence of duplicated data is an avoidable oversight and ASSISTments team has acknowledged this error on their website. All new experiments in this work and following discussions exclude data of these duplications.

Mixing Main Problems with Scaffolding Problems

A mastery learning problem sets normally contains over a hundred of main problems, and each main problem may have multiple associated scaffolding problems. Scaffolding problems were designed to help student acquire an integrated set of skills through processes of observations and guided practice; they usually tagged with different skills and have different designs from the main problems. Because of the difference in usage, scaffolding questions should not be treated as the same as main problems. Student modeling methods such as BKT and PFA exclude scaffolding features. The experiment conducted by Lua DKT did not filter out scaffolding problems. This means that Lua DKT had the advantage of additional information; thus the prediction results cannot be compared fairly with BKT. There are 73,466 rows of records of scaffolding problems.

Repeated Response Sequences with Different Skill Tagging (Duplication by Skill Tag)

The 2009-2010 skill builder dataset was created as a subset from the 2009-2010 full dataset. The full dataset from 2009-2010 includes student work from both skill builder assignments (where a student works until a mastery threshold is reached) and more traditional assignments (where a student has a fixed number of problems). Any problem (or assignment) can be tagged with any number of skill tags. Typically, problems have just one skill tag; they seldom are tagged with two skills; they are very rarely tagged with three or more. Depending on the design of the content creator, a problem set may have multiple skill tags; many assignments - especially skill builders - will have the same skill tag for all problems. When the full dataset was decomposed into only mastery style assignments, the problems and assignments that were tagged with multiple skills were included with a single tag, but repeated for each skill. This means that the sequence of action logs from one student working on one assignment were now repeated once per skill. For models such as RNNs that operate over sequences of vectors and memory on entire history of previous inputs, the issue of duplicated sequences is going to add additional weight on to the duplicated information; this will have undesired effects on RNN models.

For example, suppose we have a hypothetical scenario that a student answers two problems which have been tagged with skill ”A” and ”B”, he answers first one correctly and the next one incorrectly. Table 1.1 shows the data set where responses

Index ID	Skill ID	Problem ID	Correctness
1	A	3	1
1	B	3	1
2	A	4	0
2	B	4	0

Table 1.1: An example of repeated multiple-skill sequence

Index ID	Skill ID	Problem ID	Correctness
1	A, B	3	1
2	A, B	4	0

Table 1.2: An example of joint skills on multiple-skill problems

have been repeated on skill ”A” and ”B”. This format of data can be used in BKT models, since BKT can build two models for skill ”A” and ”B” separately. When applying this sequential data set to DKT, we believe DKT can recognize the pattern that a problem tagged with skill ”B” follows a problem tagged with ”A”; the skill ”B” problem has extremely high chance to repeat skill ”A” problem's response correctness. Note that skill ID can be mapped to skill names, but the order of skill ID is completely arbitrary.

One approach to change the way of how multiple-skill problems are handled is to simply use the combination of skills as a new joint skill. Table 1.2 shows the data set which uses a joint skill of A and B. In this case, DKT no longer has access to repeated information. PFA and BKT can also adapt this format of data too.

In order to understanding the impact of having scaffolding problems and two approaches of dealing with multiple-skill problems, we generate three different data

	09-10 (a)	09-10 (b)	09-10 (c)
Has duplicated records	No	No	No
Has scaffolding problems	Yes	No	No
Repeated multiple-skill sequences	Yes	Yes	No
Joint skills from multiple-skill	No	No	Yes

Table 1.3: Three variants of ASSISTments 2009-2010 Datasets

sets (namely 09-10 (a), 09-10 (b), 09-10 (c)) derivate from the ASSISTments 2009-1010 data set, as summarized in Table 1.3.

1.4.2 ASSISTments 2014-2015 Dataset

Even without the issue of duplicate rows, 2009-2010 skill builder set has lost its timeliness and certainly cannot represent the latest student data in an intelligent tutoring system. So we gathered another data set that covers 2014-2015 school years' student response records [4]. In this experiment, we randomly selected 100 skills from this year's data records. This data set contains 812,334 rows of records, each record represent a response to a main problem in a mastery learning problem set. Each problem set has only one associated skill and we take caution to make sure there is no duplicated row in this data set. We suspect this new data set contains different information that covers student learning patterns, item difficulties and skill dependencies.

1.4.3 KDD Cup 2010 Dataset

Our last data set comes from the Cognitive Algebra Tutor 2005-2006 Algebra system [5]. This data were provided as a development dataset in the KDD Cup 2010 competition. Although both ASSISTments and Cognitive Algebra Tutor involve using mathematics skills to solve problems, they are actually rather different from each other. ASSISTments serves primarily as computer-assisted practice for students' nightly homework and review lessons, while the Cognitive Tutor is part of an integrated curriculum and has more support for learners during the problem-solving process. Another difference in terms of content structure is that the Cognitive Tutor

[4]https://sites.google.com/site/assistmentsdata/home/assistment-2009-2010-data/skill-builder-data-2009-2010

[5]http://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp

	# Records	# Students	# Skills
09-10 (a)	401,757	4,217	124
09-10 (b)	328,292	4,217	124
09-10 (c)	275,459	4,217	146
14-15	812,334	19,457	100
KDD	607,026	574	436

Table 1.4: Dataset Statistics

presents a problem to a student that consists of questions (also called steps) of many skills. The Cognitive Tutor uses Knowledge Tracing to determine when a student has mastered a skill. A problem in the tutor can consist of questions of different skills, once a student has mastered a skill, as determined by KT, the student no longer needs to answer questions of that skill within a problem but must answer the other questions which are associated with the un-mastered skills. The number of skills in this dataset is substantially larger than the ASSISTments dataset [PH11]. One issue of using KDD data on PFA is how to estimate item difficulty feature. In this work, we use a concatenation of problem name and step name. However many such pairs are only attempted by 1 student and the difficulty values of these items are either 1.0 or 0.0, leading to both over-fitting and data leakage. To fix that, we replace difficulty values of these items with skills' difficulty information. Filtering out rows with missing values resulting in 607,026 rows of data with students responded correctly at 75.5% of the time. This data set has 574 students worked on 436 skills in mathematics. The complete statistic information of five data sets can be found in table 1.4.

	Torch DKT	TF DKT	PFA	BKT
09-10 (a)	0.79	0.81	0.7	0.6
09-10 (b)	0.79	0.82	0.73	0.63
09-10 (c)	0.73	0.75	0.73	0.63
14-15	0.7	0.7	0.69	0.64
KDD	0.79	0.79	0.71	0.62

Table 1.5: AUC Results

1.5 Results

Student performance predictions made by each model are tabulated and the accuracy was evaluated in terms of Area Under Curve (AUC) and the square of Pearson correlation (r2). AUC and r2 provide robust metrics for evaluation predictions where the value being predicted is either a 0 or 1 also represents different information on modeling performance. An AUC of 0.50 always represents the scored achievable by random chance. A higher AUC scores represents higher accuracy. r2 is the square of Pearson correlation coefficient between the observed and predicted values of dependent variable. In the case of r2, it is normalized relative to the variance in the data set and it is not a directly a measure of how good the modeled values are, but rather a way of measuring the proportion of variance we can explain using one or more variables. r2 is similar to root mean squared error (RMSE), but is more interpretable. For example, it is unclear whether an RMSE of 0.3 good or bad without knowing more on the data set. However, an r2 of 0.8 indicates the model is account for most of variability in the data set. Neither AUC nor r2 method is a perfect evaluation metric, but, when combined, they account for different aspects of model and provide us a basis for evaluating our models.

Experiments on every data set have been 5-fold student level cross validated and all parameters are learned on training data. We used EM to train BKT and the limit of iteration was set to 200. Besides the number of hidden nodes of size of

	Lua DKT	TF DKT	PFA	BKT
09-10 (a)	0.22	0.29	0.11	0.04
09-10 (b)	0.22	0.31	0.14	0.07
09-10 (c)	0.14	0.18	0.14	0.07
14-15	0.1	0.1	0.09	0.06
KDD	0.21	0.21	0.1	0.05

Table 1.6: R^2 Results

mini-batch parameters we have discussed, we set the number of epochs of DKT to 100.

The cross-validated model predictions results are shown in Table 1.5 and Table 1.6. As can be seen, DKT clearly outperforms BKT on all data sets, but the results are no longer overwhelmingly in favor of DKT (both implementations) with DKT only markedly beating PFA in 3 of 5 data sets. Note that Lua DKT uses RNN, TF DKT uses LSTM.

On the ASSISTments datasets, average DKT prediction performance across two implementations is better than PFA and it is not affected from removing scaffolding, as we change dataset from 09-10 (a) to 09-10 (b). On the other hand, PFA's performance increases from 0.70 to 073 in AUC and 0.11 to 0.14 in r2 (p 0.05), we believe that removing scaffolding helps reducing noise from data and providing PFA with a dataset with lower variance. When we switch to dataset 09-10 (c) where multiple skills were combined into joint skills, the performance of DKT suffers a noticeable hit, average AUC and average r2 drop from 0.81 to 0.74 and from 0.30 to 0.18 respectively. This observation confirms our suspicion on repeated response sequence inflates the performance of DKT models. On the 09-10 (c) dataset and 14-15 dataset where no repeated response sequences and scaffolding problems, we notice that PFA perform as good as DKT.

A deeper way of looking at the impact of repeated response sequences on data set (d) is splitting the prediction results into two, the predictions of leading records and repeated data points. We see that predictions on repeated data points (e.g. skill "B" problems in Table 4) have nearly perfect performance metrics (AUC = 0.97, r2 = 0.74). On the other hand, the leading records (e.g. skill A problems in Table 4) have much lower prediction results (AUC = 0.77, r2 = 0.23). That said, we also notice these numbers are still higher than 09-10 (c)'s results, which uses joint skill tags to avoid repeated sequences. One can explain this as making DKT to model skills individually can cause data duplications but it also can have benefits on building skill dependences over time and use such information to make better predictions.

On the KDD dataset, the performance results of two DKT implementations are definitely better than both BKT and PFA (p 0.05). There are a few possible reasons for this performance gap between PFA and DKT. First of all, as we have mentioned, we have to adjust item difficulty values for many problems in order to avoid over-fitting and data leakage, which leads to lower predictive power of that feature and lower PFA performance. Another possible explanation of DKT is winning on KDD data set is that DKT can better exploit step responses. The structure of KDD data set made it is difficult to distinguish "main problems" and "scaffolding problems", thus PFA is unable to have a more unified data set for this part of experiment. That said, the advantage of DKT shows its power on complicated and realistic datasets.

1.6 Discussion and Contribution

Within this paper, we have compared two well-studied knowledge modeling methods with the emerging Deep Knowledge Tracing algorithm. We have compared these models in terms of their power of predict student performance in 5 different data sets.

Contrary to our expectation, the DKT algorithm did not achieve overwhelmingly better performance when compared to PFA model on ASSISTments data sets, when they are properly prepared. DKT appears to perform much better on KDD dataset, but we believe this is due to PFA model undermined by inaccurate item difficulty estimation.

A second interesting finding is that when DKT is fed repeated response sequences derived from the transformation of problems tagged with multiple skills, the overall performance of DKT is certainly better than PFA and BKT. Our explanation is that DKT's implementation backbone, RNNs, has the power of remembering exact patterns of sequential data and could thus inflate prediction performance on responses tagged with multiple skills and repeated per skill. More discussion and special attention are required when handling multiple skill problems in DKT algorithm. In fact, we are very interested in what kinds of high level features DKT can detect from data, so we explore the relationship between skill difficulties between predicted performance values across skills on the 14-15 data set. Skill difficulties is not an independent variable for all three models and our assumption is that the correlation between it and the predicted performance (average of predicted correctness) of a model can reflect that model's ability of detecting such hidden information from a dataset.

Chapter 2

Incorporating Rich Features into Deep Knowledge Tracing

Proper citation of this chapter is as follows:

Liang Zhang, Xiaolu Xiong, Siyuan Zhao, Anthony Botelho and Neil T. Heffernan. Incorporating Rich Features into Deep Knowledge Tracing. In *Proceedings of the Fourth ACM Conference on Learning @ Scale, L@S 2017*

The desire to follow student learning within intelligent tutoring systems in near real time has led to the development of several models anticipating the correctness of the next item as students work through an assignment. Such models have included Performance Factors Analysis (PFA), Bayesian Knowledge Tracing (BKT), and more recently with developments in deep learning, Deep Knowledge Tracing (DKT). This DKT model, based on the use of a recurrent neural network, exhibited promising results. Thus far, however, the model has only considered the knowledge components of the problems and correctness as input, neglecting the breadth of other features collected by computer-based learning platforms. This work seeks to improve upon the DKT model by incorporating more features at the problem-level. With this higher dimensional input, an adaption to the original DKT model structure is also proposed, incorporating an auto-encoder network layer to convert the input into a low dimensional feature vector to reduce both the resource requirement and time needed to train. Experiment results show that our adapted DKT model, observing more combinations of features, can effectively improve accuracy.

2.1 Introduction

Models that attempt to follow the progression of student learning often represent student knowledge as a latent variable. As students work on new problems, these models update their estimates of student knowledge based on the correctness of responses. The problem emerges to be time series prediction, as student performance on previous items is indicative of future performance. Models then use the series of questions a student has attempted previously and the correctness of each question to predict the students performance on a new problem. Two well-known models, Bayesian Knowledge tracing (BKT) [CA94] and performance factor analysis (PFA) [PJCK09] have been widely explored due to their ability to capture this progression of knowledge with reliable accuracy. Both of these models, exhibiting success in terms of predictive accuracy, use differing algorithms to estimate student knowledge. BKT, for example, uses a bayesian network to learn four parameters per knowledge component, or skill, while the PFA model uses a logistic regression over aggregated performance to determine performance for each skill. The concept to treat each skill individually is perhaps a leading factor in the success of these models, as they understand that students will exhibit different learning behaviors depending on content.

Deep learning is an emerging approach which has proved to yield promising results in a range of areas including pattern recognition, natural language processing and image classification[18]. The deep aspect of deep learning refers to the multiple levels of transformation that occur between input nodes and output nodes; these levels are usually referred to as layers, with each layer consisting of numerous nodes. The hidden nodes are used to extract high level features from previous layers and pass that information on to the next layer. However, the features extracted by deep learning is largely uninterpretable due to the complexity. This complexity makes it infeasible to explain the meaning behind every parameter learned by the model, unlike BKT and PFA which attempt to incorporate interpretability with its estimates.

Many deep learning algorithms like recurrent neural network (RNN) and convolutional neural networks (CNN) have been proposed in recent years to benefit machine learning systems with complex, yet more accurate representative models. Such an attempt in the field of learning analytics is that of Deep Knowledge Tracing (DKT). Building from the promising results of that model, this work seeks to make better use of the complex nature of deep learning models to incorporate more features to improve predictive accuracy. We also explore how other deep learning structures can help reduce these high dimensional inputs into smaller representative feature vectors.

2.2 Deep Learning in Education

Deep knowledge tracing (DKT), introduced by Piech et al. [PBH$^+$15], applies a RNN for this educational data mining task of following the progression of student knowledge. Similar to BKT, this adaptation observes knowledge at both the skill level, observing which knowledge component is involved in the task, and the problem level, observing correctness of each problem. The input layer of the DKT model is described as an exercise-performance pair of a student, $\{(X_{st1,1}, Y_{st1,1}), (X_{st1,2}, Y_{st1,2}), \ldots, (X_{st1,T}, Y_{st1,T})\}$, while the output layer is $Y_{st1,2}, Y_{st1,3}, \ldots, Y_{st1,T+1}$. The term $X_{st1,1}$ refers to the feature combination of question (or skill) and correctness of student 1 on a problem of skill 1. $Y_{st1,1}$ refers to the correctness of a problem from skill 1 for student 1. In other words, the skill and correctness of each item is used to predict the correctness of the next item, given that problems skill.

The DKT algorithm uses a recurrent neural network to represent latent knowledge state, along with its temporal dynamics. As a student progresses through an assignment, it attempts to utilize information from previous timesteps, or problems, to make better inferences regarding future performance. A popular variant of RNN, also used in the DKT model, is that of long short-term memory (LSTM) networks. The key difference of LSTMs to traditional RNNs is the internal node structure, that acts like a conveyor belt in determining how to modify information within each recurrent node. The LSTM variant uses three gates to remove or add information to the cell states, determining how much information to remember from previous timesteps and also how to combine that memory with information from the current timestep.

The recurrent hidden nodes are trained to identify and retain the relevant aspects of the input history as it pertains to student performance. The appearance of DKT drew attention by the educational data mining community due to the claimed dramatic improvement over BKT, claiming about 25% gain in predictive performance using the ASSISTments 2009 benchmark dataset. At the 2016 Educational Data Mining Conference, three papers [WKHE16, KLM16, XZIB16] were published to compare DKT with traditional probabilistic and statistical models. They argue that traditional models and variants still perform as well as this new method with better interpretability and explanatory power.

Due to the recency of the DKT model, it is not as deeply researched as other established methods. We believe that DKT is a promising approach due to its comparable performance, and with the emergence of new neural network optimization algorithms, the structure has space for improvement. Thus far only question (or skill) and correctness are considered as input to the DKT model, but the network can easily consider more features. In this paper, we explore the inclusion of more features to improve the accuracy of prediction. However, the incorporation of new features can quickly increase the input layer dimensionality, requiring careful consideration to avoid model overfitting and also to ensure the feasibility of training such a model within reasonable hardware requirements.

While a simple feed-forward neural network can be trained relatively quickly depending on the number of nodes and size of the dataset, RNNs are considerably more computationally expensive due to the comparatively larger number of parameters. In such models fitting procedures often take hours or days to run on large data sets. For example, training a LSTM DKT model with 50 skills and 200 hidden nodes needs to learn 250,850 parameters. In our environment, the training of DKT models on the ASSISTments 2009 benchmark dataset takes 3.5 minutes per epoch, equating to more than 14 hours when using a 5 fold cross validation run over 50 epochs. In contrast, BKT is able to train on the same dataset within 10 minutes.

In this way, training time and the number of parameters are considered as an important metric of comparison; such models need to provide significant gains to predictive performance to justify their usage over simpler models. To this extent, the network structure of DKT may benefit from reduced dimensionality, particularly if this can be achieved without sacrificing performance. An auto-encoder is one such approach to this problem. Auto-encoders are multi-layer neural networks with a small central layer that can convert high dimensional data to low dimensional representative embeddings that can be used to reconstruct the high dimensional input vectors; in this way dimensionality is reduced without the loss of important information. This technique is an unsupervised learning algorithm that applies backpropagation, much like a traditional feed-forward neural network, observing the input vector as the training output. Using a smaller number of nodes in the hidden layer, therefore, finds a smaller number of values that can reconstruct the input. Once trained, the output layer can be removed, and the hidden layer can connect to another network layer. Auto-encoders may be stacked in this way,s but t each layer must be trained one at a time. Like other neural network, the gradient descent method is used to train the weight values of the parameters.

2.3 Improving DKT with More Features

Intelligent tutoring systems often collect additional features about the interaction of students including information on problems, instructional aids, and time spent on individual tasks. Models and algorithms that make use of this additional information have been proposed. For example, hint usage and the number of attempts need to find the problem answer are adopted to predict the performance in the sequence of actions (SOA) model [DZWH13]; partial credit history acquired based on the number of hints used and the number of attempts are used to predict the probability of that students getting the next question correct [VIAWH15].

As previously described, it is easy to incorporate useful information such as this into the input layer of a neural network. However, the key consideration is how feature engineering is performed on these features. Feature engineering played a vital

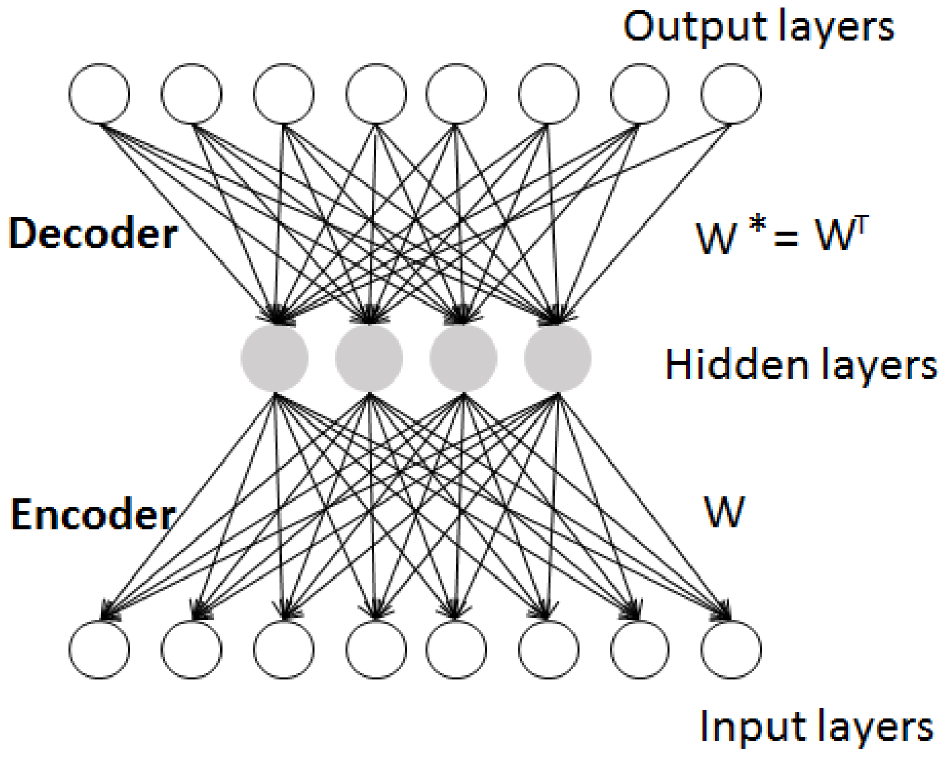

Figure 2.1: A one layer auto-encoder neural network; the weights of the decoder is simply the matrix transpose of the encoders. The hidden layer becomes a dense feature vector representative of the input layer.

role for the NTU team [YLH+10] who won the KDD competition in 2010. They incorporated a large number of features and cross-features into a vector-space model and then trained a traditional classifier. They also identified some useful feature combinations to improve the performance. Cross features were used in the original DKT work as well, utilizing a one-hot encoding to represent an correct and incorrect response for each skill separately as a vector of 2 times the number of skills; alternatively, such information could be represented separately, with a one-hot encoding representing skills, and just one binary metric to indicate correctness equating to a vector of the number of skills plus 1. In wide-and-deep learning proposed by Google [CKH+16], sparse features and cross features are selected for wide part, while the continuous columns and the embedding dimension for each categorical column are selected for deep part. These exemplary models use the engineering of features to improve model accuracy helping to motivate the methodology of this work.

2.3.1 Feature Process

In order to train the RNN model on student-tutor interaction data, the information must be converted into a sequence of fixed-length input vectors. Several features are selected for our modeling experiment; they are exercise (skill) tag, correctness, time (the time in seconds before the student's first response), hint usage (total number of hints requested by the student), attempt count (the number of attempts made to answer correctly on this problem), and problem view (total number of times the student encountered the problem so far). The exercise tag feature is used to identify the content of a problem, acting as the skill-level tag. In different datasets, the skill level tag can exhibit differing representations, described by either a numeric skill id or the name of the knowledge component.

Numerical features like time, hint usage, attempt count and problem views can be bucketed into categorical features which can be used to construct cross features in order to reduce the complexity of the model. This process simplifies the input without losing much information, as small differences in numeric values are often less important than large differences. For example, if a student finishes exercise a within 10 seconds while the other student is 300 seconds in the same exercise, the time difference represents their different mastery in exercise. Meanwhile, comparing a student who finishes in 10 seconds to a student who finishes in 11 seconds demonstrate similar, if not arguably the same level of understanding. Bucketing still captures this information while significantly reducing model complexity. The numeric features in this paper are bucketed across all skills and the result is represented by a one-hot encoding.

Cross features such as the tuple of exercise and correctness, are represented as one integer represented by a one-hot encoder. The advantage of using cross features has been shown to improve model performance [YLH+10] while models representing features separately exhibit degraded performance [PBH+15]. However, the disadvantage of using cross features is the rapid increase of the dimensionality of the input vector. As the dimensionality increases, it is hard for the model to converge to the global optimal. At the same time, computational resources may become exhausted due to the large number of parameters. Dimensionality reduction, and the extraction of key features, is critical to guarantee the running of such models. Here, an auto-encoder is used to accomplish this task. In our experiment, the dimension is successfully reduced to a quarter of the input size. We train the initial weights using an auto-encoder, and hold them constant while training the remainder of the model.

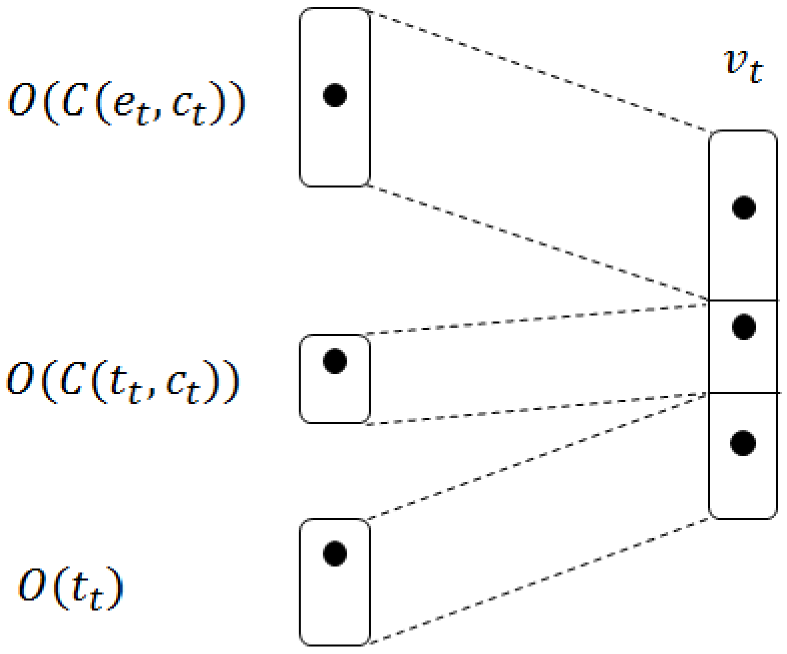

Figure 2.2: Feature concatenation

2.3.2 Model

The input vector of our model is constructed by concatenating one-hot encodings for separate features as illustrated in figure 2.2, where v_t represents the resulting input vector of each student exercise. The term e_t refers to the exercise tag, while $c_t$refers to correctness, and t_t represents time before the first response. Concatenation is described in the formulas below.

$$v_t = O(C(e_t, c_t)) +\prime O(C(t_t, y_t)) +\prime O(t_t) \tag{2.1}$$

$$C(e_t, c_t) = e_t + (max(e) + 1) * c_t \tag{2.2}$$

In these, $O(\cdot)$ is the one-hot encoder format, $C(\cdot, \cdot)$ is the cross feature, and the $+\prime$ operator is used to denote concatenation, not addition in 2.1. In 2.2, 1 is added

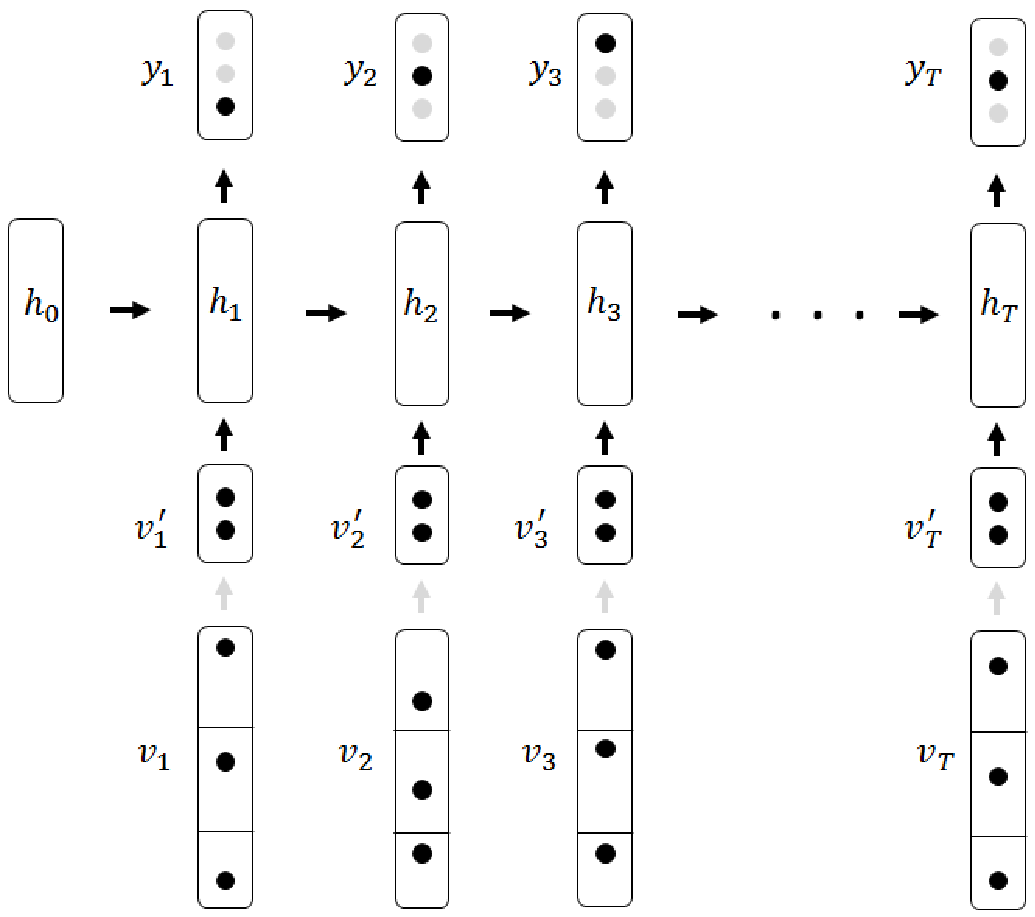

Figure 2.3: Feature concatenation

in the expression due to the unincluded exercise.

Figure 2.3 depicts the resulting model representation utilizing an auto-encoder layer to support the added features. In Figure 3, $v_1\prime$ represents the feature vector extracted from v_1 by auto-encoder; after training, this is simply the output of the hidden layer for each input vector. The gray arrows mean that weights between the two layers are held constant, so the auto-encoder is trained separately in advance. From our experiments, we noticed that the fine tuning of auto-encoder weights, if trained with the RNN together, would lead to overfitting due to the increase of parameters. Therefore, the pre-trained weights in encoder are fixed to prevent overfitting.

$$v_{t}\prime = tanh(W_{ae}v_t + b_{ae}) \tag{2.3}$$

$$h_t = \sigma(W_{hx}v_{t}\prime + W_{hh}h_{t-1} + b_h) \tag{2.4}$$

$$y_t = \sigma(W_{hy}h_t + b_y) \tag{2.5}$$

The model predicts performance in every exercise but just one prediction is selected at each time step because just one label exists at that time. The loss function was defined to use cross-entropy, as is common in other RNN models.

2.4 Datasets and Environment

Three educational datasets are tested in this paper. Each of these datasets comes from a system in which students interact with an intelligent tutor system for math content. Area under the curve (AUC) and r-squared metrics are measured for each. The original DKT model with inputs that include only exercise tag and correctness is used as a model for comparison. Since it is a time-series algorithm, students whose records are less than 2 are not considered.

2.4.1 ASSISTments 2009-2010 Datasets

ASSISTments is a computer-based learning system that simultaneously teaches and assesses students. This dataset was gathered from ASSISTments skill builder problem sets, which are assignments in which a student works on similar questions until he/she can correctly answer n consecutive problems correctly (where n is usually 3). After completion, students do not commonly rework the same skill. Xiong et al

[XZIB16] discovered three issues that have unintentionally inflated the performance of DKT in the original version, so the updated version of this dataset is adopted here.

Unlike other datasets, the records of a student may not be consecutive. That is why some previous works [PBH+15] report 15,391 students while others [XZIB16] report 4,217. In our model, all records that belong to one student are concatenated. The exercise tag is defined as the skill id.

In order to simplify the model and use cross features between time and others, the time is bucketed according to boundaries [-1, 60, 300, 1200, 3600, INF]. The boundary of hint count is defined as [-1, 0, 2, 4, INF], and [-1, 1, 20, 100, INF] for attempt count.

After preprocessing, this dataset consists of 4,217 students, 124 exercise tags and 338,000 records in total.

2.4.2 ASSISTments 2014-2015 Datasets

In addition to the 2009-2010 skill builder set we felt it appropriate to include a more recent representation of student data within the ASSISTments platform. We also used another dataset from ASSISTments that covers student response records from the 2014-2015 school year.

The process of feature processing with the exception of handling skill ids in this datasets is same as in the previous dataset. Unlike the ASSISTments 2009 dataset, some assignments have no mapped skill id so use the sequence id to represent skill id directly. Since the sequence level is finer than skill level, this process would introduce the noise to the dataset. The new skill id is mapped to the same pattern.

After pre-processing, the dataset consists of 19,103 students, 85 exercise tags, and 707,866 records.

2.4.3 KDD Cup 2010 Datasets

KDD Cup 2010 is an education data mining competition organized by an ACM Special Interest Group on Knowledge Discovery and Data Mining (KDD) to predict student algebraic problem performance given information regarding past performance. The dataset came from Carnegie Learning's Cognitive Tutor in Algebra from years 2005-2009.

Unlike the ASSISTments platform, the Cognitive Algebra Tutor is part of an integrated curriculum and has more support for the learner during the problem-solving process. It provides a much finer representation of the concepts assessed by an individual item. Each step a student takes to answer problem is counted as a separate interaction, with each step potentially assessing different knowledge components (KCs). We use each interaction (step) as the finest problem for prediction, over 438 knowledge components representing skill. The exercise tag is a numerated knowledge component derived from the text description. A skill composed of several sub-components is considered as a separate knowledge component. Time is bucketed according to boundaries [-1, 10, 60, 150, 300, INF]. Hint usage is bounded by [-1, 2, 5, 10, INF]. As there is no attempt count field in this dataset, problem view is instead used and bucketed according the boundaries [-1, 2, 5, 10, INF].

After processing, the data set consist of 574 students, 438 exercise tags and 809,684 records.

2.5 Result

The prediction is evaluated in terms of Area under curve (AUC) and the square of Pearson correlation (R^2). Experiment undergo 5-fold student level cross validation. There are a lot of possible feature and cross feature selection methods, but here

Model	2009	2014	KDD
DKT: exercise/correct	0.829	0.714	0.799
DKT + time/correct	0.857	0.725	0.806
AE(DKT + time/correct)	0.855	0.721	0.803
DKT + time/correct + time + hint + attempt	0.859	0.728	**0.808**
AE(DKT + time/correct + time + hint + attempt)	0.857	0.716	0.794
AE(DKT + time/correct + exercise/time + time + hint + attempt)	**0.863**	**0.731**	**0.808**

Table 2.1: The results of each of the explored models. The + operator denotes concatenation. The attempt feature in KDD data refers to the problem view feature.

Model	2009	2014	KDD
DKT: exercise/correct	0.323	0.115	0.234
DKT + time/correct	0.387	0.129	0.245
AE(DKT + time/correct)	0.387	0.124	0.239
DKT + time/correct + time + hint + attempt	0.388	0.133	0.25
AE(DKT + time/correct + time + hint + attempt)	0.393	0.119	0.221
AE(DKT + time/correct + exercise/time + time + hint + attempt)	0.403	0.135	0.25

Table 2.2: R^2 results

we just explore few of them. AUC and R^2 provide robust metrics for evaluation predictions where the value being predicted is either a 0 or 1 also represents different information on modeling performance. An AUC of 0.50 always represents the scored achievable by random chance. A higher AUC score represents higher accuracy. R^2 is the square of Pearson correlation coefficient between the observed and predicted values of dependent variable.

On all three datasets, models with incorporated features outperform the original DKT model. In the ASSISTments 2009 dataset, AUC value is improved to 0.857 from 0.829 after adding the cross feature of exercise and time. However, the AUC value just increases 0.2% when adding more features such as time, hint usage and attempt count into the input vectors. Even adding a cross feature of exercise and time shows no further improvement.

The adoption of the auto-encoder when compared to models using the same features shows degraded performance of about 0.2%. In the ASSISTments 2014 dataset, it decreases to 0.716 from 0.728 while 0.808 to 0.794 in KDD data set. However, the auto-encoder is essential if more features are to be considered. For example, the input dimension of the last model, AE(DKT + time/correct + exercise/time + time + hint + attempt), in KDD dataset is 3,079, which exhausted the GPU resources in our environment without an auto-encoder even when using small batch sizes. From the above results, the improvement of prediction is mainly contributed by incorporation of cross features.

2.6 Conclusion

The feature transformation and feature combination, when properly selected, can be used to improve the prediction accuracy. Although the parameters are difficult to interpret, such RNN models are adopted due to the performance gains.

The improvement here is attributed to the incorporation of cross features. The auto-encoder allows for the support of larger input vectors, making it possible to explore such combinations represented in one-hot encodings.

The work of extending these models has several potential directions to pursue. One such direction can explore even more features, engineered in different manners, such as tokening the words of knowledge components for different exercise representations. Similarly, a wide and deep approach can be explored in how the features are represented within model training.

The numerical data like time and hint usage can also be revisited in future work. Bucketed according to the distribution within each exercise rather than across all exercises will likely improve the representation of those features. For example, because skill B is harder than skill A, most students may answer skill A in 20 seconds while the same student requires 300 seconds in skill B.

Because of flexible structure of deep learning, another research direction is to use similar RNN model structures to make other predictions regarding concepts like wheel spinning, student dropout, or hint usage.

Part II

Application of Memory Networks

Chapter 3

Condensed Memory Networks for Clinical Diagnostic Inferencing

Proper citation of this chapter is as follows:

Aaditya Prakash, Siyuan Zhao, Sadid A. Hasan, Vivek V. Datla, Kathy Lee, Ashequl Qadir, Joey Liu and Oladimeji Farri. Condensed Memory Networks for Clinical Diagnostic Inferencing. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI 2017*

Diagnosis of a clinical condition is a challenging task, which often requires significant medical investigation. Previous work related to diagnostic inferencing problems mostly consider multivariate observational data (e.g. physiological signals, lab tests etc.). In contrast, we explore the problem using free-text medical notes recorded in an electronic health record (EHR). Complex tasks like these can benefit from structured knowledge bases, but those are not scalable. We instead exploit raw text from Wikipedia as a knowledge source. Memory networks have been demonstrated to be effective in tasks which require comprehension of free-form text. They use the final iteration of the learned representation to predict probable classes. We introduce condensed memory neural networks (C-MemNNs), a novel model with iterative condensation of memory representations that preserves the hierarchy of features in the memory. Experiments on the MIMIC-III dataset show that the proposed model outperforms other variants of memory networks to predict the most probable diagnoses given a complex clinical scenario.

3.1 Introduction

Clinicians perform complex cognitive processes to infer the probable diagnosis after observing several variables such as the patient's past medical history, current condition, and various clinical measurements. The cognitive burden of dealing with complex patient situations could be reduced by having an automated assistant provide suggestions to physicians of the most probable diagnostic options for optimal clinical decision-making.

Some work has been done in building Artificial Intelligence (AI) systems that can support clinical decision making [LKEW15a, CBS$^+$16a, CBS$^+$16b]. These works have primarily focused on the use of various biosignals as features. EHRs typically store such structured clinical data (e.g. physiological signals, vital signs, lab tests etc.) about the patients clinical encounters in addition to unstructured textual notes that contain a complete picture of the associated clinical events. Structured clinical data generally contain raw signals without much interpretation, whereas unstructured free-text clinical notes contain detailed description of the overall clinical scenario.

In this paper, We also explore the use of an external knowledge source like

Wikipedia from which the model can extract relevant information, such as signs and symptoms for various diseases. Our goal is to combine such an external clinical knowledge source with the free-text clinical notes and use the learning capability of memory networks to correctly infer the most probable diagnosis.

Memory Networks (MemNNs) [WCB14, SSWF15] are a class of models which contain an external memory and a controller to read from and write to the memory. Memory Networks read a given input source and a knowledge source several times (hops) while updating an internal memory state. The memory state is the representation of relevant information from the knowledge base optimized to solve the given task. This allows the network to remember useful features. The notion of neural networks with memory was introduced to solve AI tasks that require complex reasoning and inferencing. These models have been successfully applied in the Question Answering domain on datasets like bAbi [WBC$^+$15], MovieQA [TZS$^+$16], and WikiQA [SSWF15, MFD$^+$16]. Memory networks are harder to train than traditional networks and they do not scale easily to a large memory. End-to-End Memory Networks [SSWF15] and Key-Value Memory Networks (KV-MemNNs) [MFD$^+$16] try to solve these problems by training multiple hops over memory and compartmentalizing memory slots into hashes, respectively.

When the memory is large, hashing can be used to selectively retrieve only relevant information from the knowledge base, however not much work has been done to improve the information content of the memory state. If the network were trained for factoid question answering, the memory state might be trained to represent relevant facts and relations from the underlying domain. However, for real world tasks, a large amount of memory is required to achieve stateof-the-art results. In this paper, we introduce Condensed Memory Networks (C-MemNNs), an approach to efficiently store condensed representations in memory, thereby maximizing the utility of limited memory slots. We show that a condensed form of memory state which contains some information from earlier hops learns efficient representation. We take inspiration from human memory for this model. Humans can learn new information and yet remember even very old memories as abstractions. We also experiment with a simpler form of knowledge retention from previous hops by taking a weighted average of memory states from all the hops (A-MemNN). Even this simpler alternative which does not add any extra parameter is able to outperform standard memory networks. Empirical results on the MIMIC-III dataset reveal that C-MemNN improves the accuracy of clinical diagnostic inferencing over other classes of memory networks. To the best of our knowledge, this is the first empirical study to classify diagnosis from EHR free-text clinical notes using memory networks

3.2 Related Work

3.2.1 Memory Networks

Memory Networks (MN) [WCB14] and Neural Turnin Machines (NTM) [GWD14a] are two classes of neural networks models with external memory. MN store all information (e.g. knowledge base, background context) into external memory, assign a relevance probability to each memory slot using content-based addressing schemes, and read contents from each memory slot by taking the their weighted sum with relevance probabilities. End-to-End Memory Networks introduced multi-hop training [SSWF15] and do not require strong supervision unlike MemNN. Key-value Memory Networks [MFD$^+$16] have a key-value paired memory and is built upon MemN2N. Key-value paired structure in memory is a generalized way of storing content in memory. The contents in key memory are used to calculate the relevance probabilities. The contents in value memory are read into the model to help make the final prediction.

NTM form another family of neural networks models with external memory. The NTM controller uses both content and location-based mechanism to access the memory. On the other hand, MN only uses content-based mechanism. The fundemental difference between these two models is the MN do not have a mechanism to change the content in memory, while the NTM can modify the content of the memory in each episode. This leads to the fact that MN is easier to be trained in practice.

Another related class of models is attention-based neural networks. These models are trained to learn attention mechanism so that they can focus on important information on given input. Applying attention mechanism on machine reading comprehension task [HKG$^+$15, DLCS16, CCW$^+$16, SBB16] has shown promising results.

3.2.2 Neural Networks for Clinical Diagnosis

[LKEW15b] trained Long-short Term Memory (LSTM) to classify 128 diagnoses from 13 frequently but irregularly sampled clinical measurements extracted from patient's Electronic Health Record (EHR). Similary to their work, we formulate the problem as multilabel classification, since each medical note might be associated with multiple diagnoses. There are two differences between these two work. In our work, patient's EHR come from discharge summary, which are unstructured texts and do not contain time series information, while their data set are time series and each time series has fixed number of clinical measurements. We trained the model on the whole patient's EHR and did not extract hand-engineered features from these EHR, while they resampled all time series to an hourly rate and filled gaps created by window-based resampling in clinical measurements. We applied Memory Networks models instead of LSTM to classify diagnoses. Since memory component provides the flexibilities to store knowledge base, we collected related Wikipedia page for each diagnoses and embed these pages in memory.

3.3 Dataset

MIMIC-III (Multiparameter Intelligent Monitoring in Intensive Care) [JPS+16] is a large freely-available clinical database. It contains physiological signals and various measurements captured from patient monitors, and comprehensive clinical data obtained from hospital medical information systems for over $58K$ Intensive Care Unit (ICU) patients. We use the noteevents table from MIMIC-III: v1.3, which contains the unstructured free-text clinical notes for patients. We use 'discharge summaries', instead of 'admission notes', as former contains actual ground truth and free-text. Since discharge summaries are written after diagnosisdecision, we sanitize the notes by removing any mention of class-labels in the text.

As shown in Table 3.1, medical notes contain several details about the patient but the sections are not uniform. We do not separate the sections other than the *DIAGNOSIS*, which is our label. There are multiple labels (diagnoses) for a given note, and a note can belong to multiple classes of diagnoses, thus we formulate our task as a multiclass-multilabel classification problem. The number of diagnoses per note is also not consistent and shows a long tail (Figure 3.1). We have taken measures to counteract these issues, which are discussed in the *Memory addressing* section.

Some diagnoses are less frequent in the data set. Without enough training instances, a model is not able to learn to recognize these diagnoses. Therefore, we experiment with a varying number of labels in this work (see details in the Experi-

Medical Note (partially shown)
Date of Birth: [**2606-2-28**] Sex: M Service: Medicine CHIEF COMPLAINT: Admitted from rehabilitation for hypotension (systolic blood pressure to the 70s) and decreased urine output. HISTORY OF PRESENT ILLNESS: The patient is a 76-year-old male who had been hospitalized at the [**Hospital1 3007**] from [**8-29**] through [**9-6**] of 2002 after undergoing a left femoral-AT bypass graft and was subsequently discharged to a rehabilitation facility. On [**2682-9-7**], he presented again to the [**Hospital1 3087**] after being found to have a systolic blood pressure in the 70s and no urine output for 17 hours.
Final Diagnosis
Cardiorespiratory arrest. Non-Q-wave myocardial infarction. Acute renal failure.

Table 3.1: An example of MIMIC-III

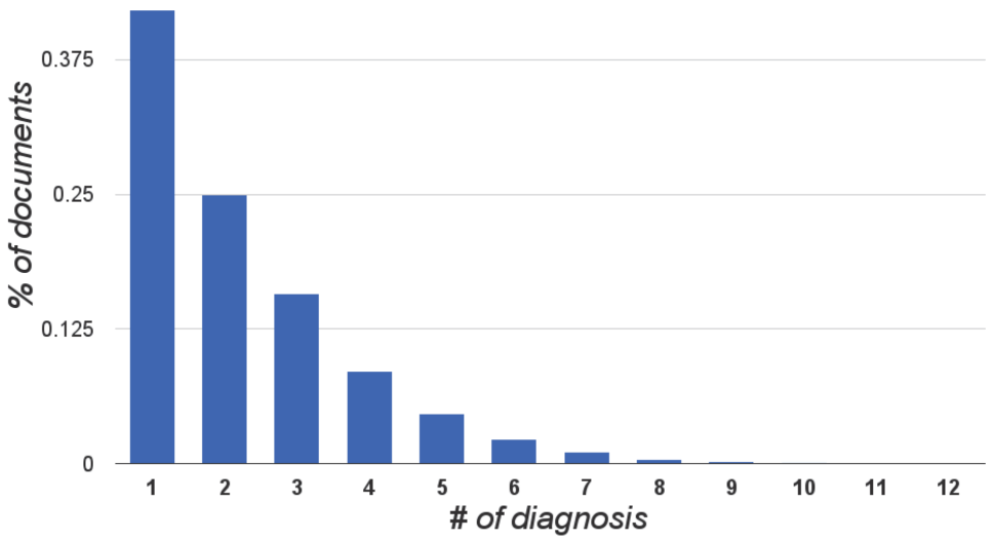

Figure 3.1: Distribution of number of diagnosis in a note.

Cardiac arrest (partially shown)
Cardiac arrest is a sudden stop in effective blood circulation due to the failure of the heart to contract effectively or at all.[1] A cardiac arrest is different from (but may be caused by) a myocardial infarction (also known as a heart attack), where blood flow to the muscle of the heart is impaired such that part or all of the heart tissue dies. ... **Sign and symptoms** Cardiac arrest is sometimes preceded by certain symptoms such as fainting, fatigue, blackouts, dizziness, chest pain, shortness of breath, weakness, and vomiting. The arrest may also occur with no warning. ...

Table 3.2: Partially shown example of a relevant Wikipedia page.

ments section).

3.3.1 Knowledge Base

We use Wikipedia pages (see Table 3.2) corresponding to the diagnoses in the MIMIC-III notes as our external knowledge source. WikiProject Medicine is dedicated to improving the quality of medical articles on Wikipedia and the information presented in these pages are generally shown to be reliable [Tre11]. Since some diagnosis terms from MIMIC-III dont always match a Wikipedia title, we use the Wikipedia API with the diagnoses as the search terms to find the most relevant Wikipedia pages. In most cases we find an exact match while in the rest we pick the most relevant page. We use the first paragraph and the paragraphs corresponding to the Signs and symptoms sections for our experiments. In cases where such a section is not available, we use the second and third paragraphs of the page. This happens for the obscure diseases, which have a limited content.

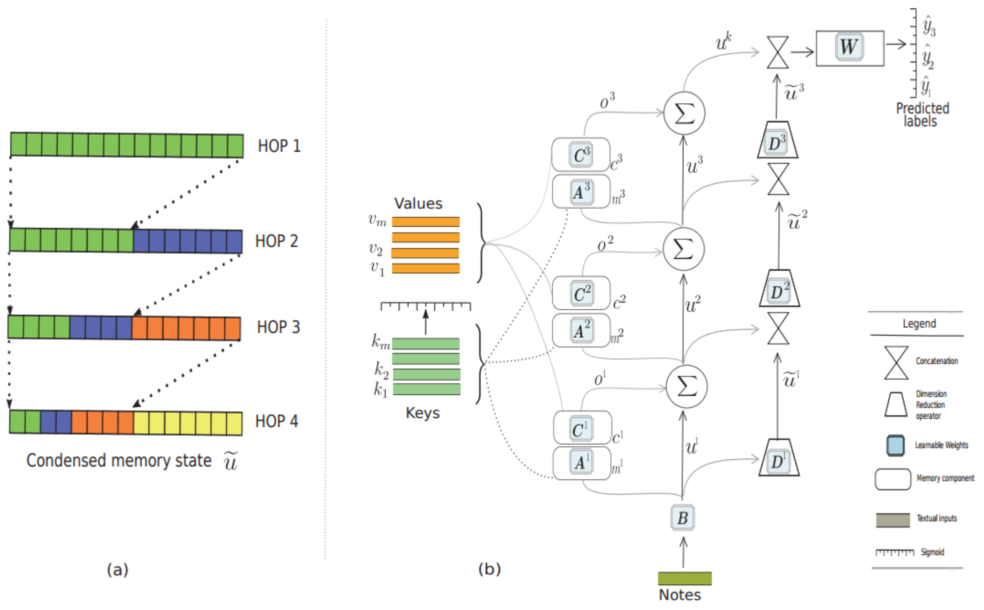

Figure 3.2: Distribution of number of diagnosis in a note.

3.4 Condensed Memory Networks

The basic structure of our model is inspired by MemNN. Our model tries to learn memory representation from a given knowledge base. Memory is organized as some number of slots $m_1, \ldots, m_t$. For the given input text i.e. medical notes $x_1, \ldots, x_n$, the external knowledge base (wiki pages, wiki titles) $(k_1, v_1), (k_2, v_2), \ldots, (k_m, v_m)$, and the diagnoses of those notes y, we aim to learn a model $\mathcal{F}$ such that

$$\mathcal{F}(x_n, (k_m, v_m)) = \hat{y} \rightarrow y \tag{3.1}$$

We break down this function $\mathcal{F}$, in four parts I, G, O, R which are the standard components of Memory Networks.

- **I: Input memory representation** is the transformation of the input x to some internal representation u using learned weights B. This is the internal state of the model and is similar to the hidden state of RNN-based models.

In this paper, we propose the addition of a condensed memory state $\tilde{u}$, which is obtained via the iterative concatenation of successively lower dimensional representations of the input memory state u.

- **G: Generalization** is the process of updating the memory. MemNN updates all slots in the memory, but this is not feasible when the size of the knowledge source is very large. Therefore, we organize the memory as key-value pairs as described in [MFD$^+$16]. We use hashing to retrieve a small portion of keys for the given input.

- **O: Output memory representation** is the transformation of the knowledge (k, v) to some internal representation m and c. While End-to-End MemNN uses an embedding matrix to convert memories to learned feature space, our model uses a two-step process because we represent wikititles and wikipages as different learned spaces. We learn matrix A to transform wikipages (keys) and C to transform wikititles (values). Our choice of wikipages as keys and wikititles as values is deliberate the input "medical notes" more closely match the text of the wikipages and the diagnoses more closely match the wiki-titles. This allows for a better mapping of features; our empirical results validate this idea.

  Let k represent the hop number. The output memory representation is obtained by:

$$o^k = \sum_i Addressing(u^k, m^k) \cdot c_i^k \tag{3.2}$$

  where $Addressing$ is a function which takes the given input memory state u and provides the relevant memory representation m.

- **R: Response** combines the internal state u, internal condensed state $\tilde{u}$ and the output representation o to provide the predicted label $\hat{y}$. We sum u and o and then take the dot product with another learned matrix W. We then concatenate this value with condensed memory state $\tilde{u}$. This value is then passed through sigmoid to obtain the likelihood of each class. We use sigmoid instead of softmax in order to obtain multiple predicted labels, $\hat{y}_1, \ldots, \hat{y}_r$ among possible R labels.

$$u^{k+1} = u^k + o^k \tag{3.3}$$

$$\tilde{u}^k + 1 = u^{k+1} \oplus D_1 \cdot \tilde{u}^k \tag{3.4}$$

where $\oplus$ denotes concatenation of vectors.

Our major contribution to memory networks is the use of condensed memory state $\tilde{u}$ in combination with input memory state u to do the inference. As shown in Figure 3.2(a), $\tilde{u}$ is transformed to include the information of previous hops, but in lower dimensional feature space. This leads to a longer term memory representation, which is better able to represent hierarchy in memory.

$$\hat{y}_r = \arg\max_{r \in R} \frac{1}{1 + e^{-1*(\tilde{u}^{k+1}) \cdot W}}$$

3.4.1 Network Overview

Figure 3.2(b) shows the overview of the structure. The input x is converted to internal state u^1 using transformation matrix B. This is combined with memory key slots m^1 using matrix A. Memory addressing is used to retrieve the corresponding memory value c^1. This value is transformed using matrix C to output memory representation o^1. In parallel, memory state u is condensed to half of its original dimension using the transformation matrix D. If u is of size $1K$ then D is of size $K\frac{K}{2}$. We call this reduced representation of u the condensed memory state, $\tilde{u}$. This is the end of first hop. This process is then repeated for a desired number of hops. After each hop, the condensed memory state $\tilde{u}$ becomes the concatenation of its previous state and its current state, each reduced to half its original dimension.

3.4.2 Average Memory Networks

In C-MemNN, the transformation of $\tilde{u}$ at every hop adds more parameters to the model, which is not always desirable. Thus, we also present a simpler alternative model, which we call A-MemNN, to capture hierarchy in memory representation without adding any learned parameters. In this alternative model, we compute the weighted average of $\tilde{u}$ across multiple hops. Instead of doing concatenation of previous $\tilde{u}$ values, we simply from different hops, we simply maintain an exponential moving average.

$$\tilde{u}^{k+1} = \tilde{u}^k + \frac{\tilde{u}^{k-1}}{2} + \frac{\tilde{u}^{k-1}}{4} + \dots \tag{3.5}$$

where, the starting condensed memory state is same as input memory state $\tilde{u}^1 = u^1$.

3.4.3 Memory Addressing

Key-Value addressing as described in KV-MemNN uses softmax on the product of question embeddings and retrieved keys to learn a relevance probability distribution over memory slots. The representation obtained is then the sum of the output memory representation o, weighted by those probability values. KV-MemNN was designed to pick the single most relevant answer given a set of candidate answers.

Using softmax significantly decreases the estimated relevance of all but the most probable memory slot. This presents a problem for multi-label classification in which several memory slots may be relevant for different target labels. We experimented with changing softmax to sigmoid to alleviate this problem, but this was not sufficient to allow the incorporation the condensed form of the internal state u arising from earlier hops. Thus, we explore a novel alternate addressing scheme, which we call gated addressing. This addressing method uses a multi-layer feed-forward neural network (FNN) with a sigmoid output layer to determine the appropriate weights for each memory slot. The network calculates a weight value between 0 and 1 for each memory slot, and a weighted sum of memory slots is obtained as before.

3.4.4 Document Representation

There are a variety of models to represent knowledge in key-value memories, and the choice of model can have an impact on the overall performance. We use a simple bag-of-words (BoW) model which transforms each word w_{ij} in the document $d_i = w_{i1}, w_{i2}, w_{i3}, \ldots, w$ in to embeddings, and sums these together to obtain the vectors $\Phi(d_i) = \sum_j Aw_{ij}$, with A being the embedding matrix. Medical notes, memory keys and memory values are all represented in this way.

3.5 Experiments

The distribution of diagnoses in our training data has a very long tail. There are 4,186 unique diagnosis in all of MIMIC-III. However, many diagnoses (labels) occur in only a single note. This is not sufficient to efficiently train those labels. The 50 most-common labels cover 97% of the notes and the 100 most-common labels cover 99.97%. Thus, we frame this task as multi-label classification for top-N labels. We

		# classes = 50			# classes = 100		
# Hops	Model	AUC (macro)	Average Precision @5	Hamming Loss	AUC (macro)	Average Precision @5	Hamming Loss
3	End-to-End	0.759	0.32	0.06	0.664	0.23	0.15
	KV MemNN	0.761	0.36	0.05	0.679	0.24	0.14
	A-MemNN	0.762	0.36	0.06	0.675	0.23	0.14
	C-MemNN	**0.785**	**0.39**	**0.05**	**0.697**	**0.27**	**0.12**
4	End-to-End	0.760	0.33	0.04	0.672	0.24	0.15
	KV MemNN	0.776	0.35	0.04	0.683	0.24	0.13
	A-MemNN	0.775	0.37	0.03	0.689	0.23	0.11
	C-MemNN	**0.795**	**0.42**	**0.02**	**0.705**	**0.27**	**0.09**
5	End-to-End	0.761	0.34	0.04	0.683	0.25	0.14
	KV MemNN	0.775	0.36	0.03	0.697	0.25	0.11
	A-MemNN	0.804	0.40	0.02	0.720	0.29	0.11
	C-MemNN	**0.833**	**0.42**	**0.01**	**0.767**	**0.32**	**0.05**

Table 3.3: Evaluation results of various memory networks on MIMIC-III dataset

present experiments for both the 50 most-common and 100-most common labels. For all experiments, we truncate both notes and wiki-pages to 600 words. We reduce the trained vocabulary to 20K after removing common stop-words. We use a common dimension of 500 for all embedding matrices. We use a memory slot of dimension 300. A smaller embedding of dimension 32 is used to represent the wiki-titles.

We presents experiments for end-to-end memory networks , Key-Value Memory Networks (KV-MemNNs) and our models, Condensed Memory Networks (C-MemNN) and Averaged Memory Networks (A-MemNN). We separately train models for three, four and five hops. The strength of our model is the ability to make effective use of several memory hops, and so we do not present results for one or two hops.

3.5.1 Results and Analysis

We present experiments in which performance is evaluated using three metrics: the area under the ROC curve (AUC), the average precision over the top ten predictions, and the hamming loss. The AUC is calculated by taking unweighted mean of the AUC values for each label - this is also known as the macro AUC. Average precision over the top ten predictions is reported because it is a relevant metric for real world applications. Hamming loss is reported instead of accuracy, because it is a better measure for multi-label classification.

As shown in Table 3.3, C-MemNN is able to exceed the results of various other memory networks across all experiments. The improvement is more pronounced with a higher number of memory hops. This is because of the learning saturation of vanilla memory networks over multiple hops. While A-MemNN has better results for higher hops it does not improve upon KV-MemNN at lower hops. The strength of our model lies at higher hops, as the condensed memory state $\tilde{u}$ after several hops contains more information than the same size input memory state u. Across all models, results improve as the number of hops increases, although with diminishing returns. The AUC value of C-MemNN with five memory hops for 100 labels is higher than the AUC value for End-to-End models trained only for three hops, which shows efficient training of higher hops produces good results.

Most documents do not have five labels (Figure 3.1) and thus precision obtained for five predictions is poor across all models. Hamming Loss correlates very well with other metrics along with the cross-entropy loss function, which was used for training.

3.6 Conclusions and Future Work

[WCB14] discussed the possibility of a better memory representation for complex inferencing tasks. We achieved a better memory representation by condensing the previous hops in a novel way to obtain a hierarchical representation of the internal memory. We have shown the efficacy of the proposed memory representation for clinical diagnostic inferencing from raw textual data. We discussed the limitations of memory networks for multi-label classification and explored gated addressing to achieve a better mapping between the clinical notes and the memory slots. We have shown that training multiple hops with condensed representation is helpful, but this is still computationally expensive. We plan to investigate asynchronous memory updating, which will allow for faster training of memory networks. In the future, we will explore other knowledge sources and recently proposed word vectors for medicine words, BioNLP [CCKP16].

Chapter 4

Automated Essay Scoring using Neural Memory Model

Proper citation of this chapter is as follows:

Siyuan Zhao, Yaqiong Zhang, Xiaolu Xiong, Anthony Botelho and Neil T. Heffernan. A Memory-Augmented Neural Model for Automated Grading. In *Proceedings of the Fourth ACM Conference on Learning @ Scale, L@S 2017*

The need for automated grading tools for essay writing and open-ended assignments has received increasing attention due to the unprecedented scale of Massive Online Courses (MOOCs) and the fact that more and more students are relying on computers to complete and submit their school work. In this paper, we propose an efficient memory networks-powered automated grading model. The idea of our model stems from the philosophy that with enough graded samples for each score in the rubric, such samples can be used to grade future work that is found to be similar. For each possible score in the rubric, a student response graded with the same score is collected. These selected responses represent the grading criteria specified in the rubric and are stored in the memory component. Our model learns to predict a score for an ungraded response by computing the relevance between the ungraded response and each selected response in memory. The evaluation was conducted on the Kaggle Automated Student Assessment Prize (ASAP) dataset. The results show that our model achieves state-of-the-art performance on this dataset.

4.1 Introduction

Automated grading is a critical part of Massive Open Online Courses (MOOCs) system and any intelligent tutoring systems (ITS) at scale. Essay writing is usually a common student assessment process in schools and universities. In this task, students are required to write essays of various length, given a prompt or essay topic. Some standard tests, such as Test of English as a Foreign Language (TOEFL) and Graduate Record Examination (GRE), assess student writing skills. Manually grading these essay will be time-consuming. Thus automated essay scoring (AES) systems has been used in these tests to reduce the time and cost of grading essays. Moreover, as massive open online courses (MOOCs) become widespread and the number of students enrolled in one course increases, the need for grading and providing feedback on written assignments are ever critical.

AES has employed numerous efforts to improving its performance. AES uses statistical and Natural Language Processing (NLP) techniques to automatically predict a score for an essay based on the essay prompt and rubric. Most existing AES systems are built on the basis of predefined features, e.g. number of words, average word length, and number of spelling errors, and a machine learning algorithm [CH13]. It is normally a heavy burden to find out effective features for AES. Moreover, the performance of the AES systems is constrained by the effectiveness of the predefined features. Recently another kind of approach has emerged, employing neural network models to learn the features automatically in an end-to-end manner [TN16]. By this means, a direct prediction of essay scores can be achieved without performing any feature extraction. The model based on long short-term memory (LSTM) networks in [TN16] has demonstrated promise in accomplishing multiple types of automated grading tasks.

Neural Networks have achieved promising results on various NLP tasks, including machine translation [BCB14, CvMG$^+$14], sentiment analysis [dSG14], and question answering [KIO$^+$16, WCB14, MFD$^+$16, SSWF15]. Neural Network models, in terms of NLP tasks, use word vectors to learn distributed representations from text. The advantages are that these models do not require hand-engineered features and can be trained to solve tasks in an end-to-end fashion.

Recent work [TN16] has exploited several Recurrent Neural Network (RNN) models to solve AES tasks. The results show that neural-based models outperform even strong baselines. Memory Networks (MN) [WCB14, MFD$^+$16, SSWF15] have been recently introduced to deal with complex reasoning and inferencing NLP tasks and have been shown to outperform RNNs on some complex reasoning tasks [SSWF15]. MN is a class of models which contains an external scalable memory and a controller to read from and write to that memory. The notion of neural networks with memory was introduced to solve complex reasoning and inferring AI-tasks which require remembering external contexts.

To our knowledge, no study has been conducted to investigate the feasibility and effectiveness of MN applied in automated grading tasks. In this study, we develop a generic model for such tasks using Memory Networks inspired by their capability to store rich representations of data and reason over that data in memory. For each essay score, we select one essay exhibiting the same score from student responses as a sample for that grade. All collected sample responses are loaded into the memory of the model. The model is trained with the rest of student responses in a supervised learning manner on these data to compute the relevance between the representation of an ungraded response and that of each sample. The intuition is that as a part of a scoring rubric, a number of sample responses of variable quality are usually provided to students and graders to help them better understand the rubric. These collected responses are characterized with expectations of quality described in the rubric. The model is expected to learn the grading criteria from these responses. We evaluate our model on a publicly available essay grading data set from the Kaggle Automated Student Assessment Prize (ASAP) competition [1]. Our experiments show that our model achieves state-of-the-art results on this dataset.

The rest of the paper is organized as follows. Section 2 gives an overview of related work in this research area. Section 3 provides detailed information of our model. Section 4 describes the ASAP dataset and evaluation metrics used to test our framework. Furthermore, it contains the details of our implementation and experimental setup to help other researchers replicate our work. In section 5, we present the results of our our model and compare them with other models. Finally, we discuss the results and conclude the paper.

4.2 Related Work

4.2.1 Automated Grading

MOOCs were introduced in 2008 and become more popular recently. Most MOOCs systems provide automated grading as their important features to prove the efficiency of their interaction with massive number of online users. Some specific as-

[1]https://www.kaggle.com/c/asap-aes

signment types have been adopted for automated grading since the correct answers of these kinds of assignments have some simple fixed-forms, such as multi-choice questions. Programming assignments are the represents of these kinds of assignments with simple form answer such as ”yes” or ”no” [FW65, Hel07]. Not satisfied with providing answers for one specific assignment, more efforts have been devoted to providing feedback on many different assignments according to the shared features of the programming codes [NPHG14, PHN$^+$15].

However, many assignment types cannot be responded well only with simple feedback. Some studies have been conducted with the attempt to fixing this problem by using semi-automatic grading approach. This kind of approach aims to optimize the collaboration between humans and machines and provide short-answers [MPRo13, BBJV14]. Another approach is to provide prediction directly. One research direction of this approach is to apply information extraction techniques to constructing specific answer patterns manually or to training from large training dataset with strong supervision support. Another direction is to compare the students' answers with a established standard answer with an unsupervised text-similarity approach [MM09].

Most studies mentioned above are dealing with simple fixed-form answers or short-answers assignments. Some complex assignments have long form answer instead of short, simple one. Essay writing with a given topic is a typical assignment with long form answers and AES has become one important research branch of automated grading system.

AES is generally treated as a machine learning problem. We can group the existing AES solutions from different points of view. Most developed AES system is based on a number of predefined features. These features include essay length, number of words, lexicon and grammar, syntactic features, readability, text coherence, essay organization, and so on [CH13]. Recently, there emerges another trial to treat the whole essays as inputs and learn the features automatically in an end-to-end manner [TN16]. Without pres-working on features extraction, work burden was lightened. Moreover, the predicting accurate is improved by removing the dependency of effectiveness of predefined features.

Based on learning techniques utilized in existing solutions, we divide them into three categories: regression based approach, classification based approach and preference ranking based approach. PEG-system and E-rater are two examples that belong to regression based approach. Specifically, when the scores range of the essays is wide, the regression based approach is normally adopted since it treats the essay score as a continuous value.

Besides essay writing, some complex assignments such as medicinal assignments utilized regression model as well [GZF16]. Some work such as [RL02, Lar98] treated the AES task as a classification problem. Each possible score is converted into a class label. By using classic classification algorithms, AES system predicts which class an essay should belongs to. Since it treats each score as a class label, this kind of approach is not suitable for a very large range of scores. Recently preference ranking based approach was also proposed by [YBM11].

According to the prompts or essay topics the AES system deals with, the existing solutions can be divided into two groups: prompt-specific and generic. The prompt-specific approach train the AES system with essays from one specific topic. This kind of AES system normally has excellent performance on the specific topic it was trained. Most of existing works belongs to this prompt-specific approach. Generic approach train the AES system with essays from different prompts. As an example, the work of [CH13] proposed a domain adaptation technique which is based on Bayesian linear ridge regression, to achieve a generic prompt adaption AES system.

This kind of approach normally neglects the prompt related features but focus on writing quality.

4.2.2 Memory Networks

Memory Networks (MN) [WCB14] and Neural Turing Machines (NTM) [GWD14b] are two classes of neural networks models with external memory. MN store all information (e.g. knowledge base, background context) into external memory, assign a relevance probability to each memory slot using content-based addressing schemes, and read contents from each memory slot by taking the their weighted sum with relevance probabilities. End-to-End Memory Networks (MemN2N) [SSWF15] can be trained end-to-end compared to MN, and hence require less supervision. Key-value Memory Networks [MFD$^+$16] have a key-value paired memory and is built upon MemN2N. Key-value paired structure in memory is a generalized way of storing content in memory. The contents in key memory are used to calculate the relevance probabilities. The contents in value memory are read into the model to help make the final prediction.

4.3 Model

An illustration of our model is given in Figure 4.1, which is inspired by the work of memory networks applied in question answering [MFD$^+$16, SSWF15]. Our model consists of four layers: input representation layer, memory addressing layer, memory reading layer, and output layer. Input representation layer is responsible for generating a vector representation for a student response. Memory addressing layer loads selected samples of student responses to memory, and assigns a weight to each memory piece. Afterward memory reading layer gathers the content from memory

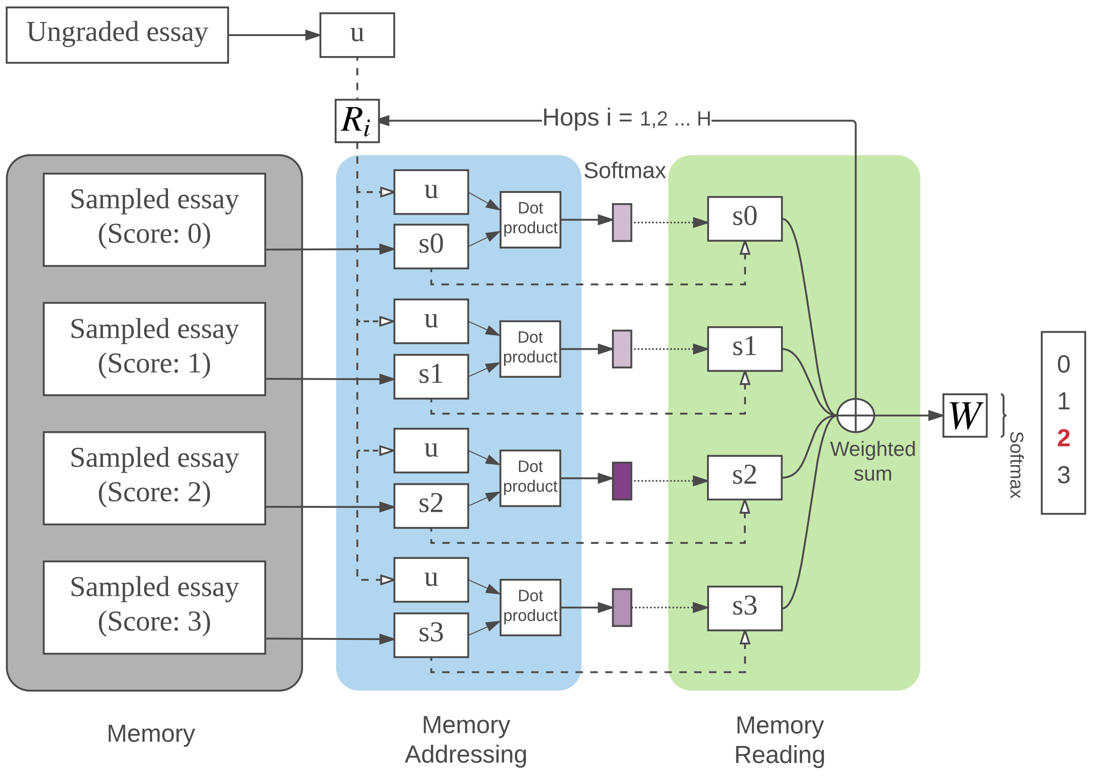

Figure 4.1: An illustration of memory networks for AES. The score range is 0 - 3. For each score, only one sample with the same score is selected from student responses. There are 4 samples in total in memory. Input representation layer is not included.

by taking weighted sum of each memory piece based on the weights calculated from previous layer, and produces a resulting state. Finally the output layer makes the prediction on the basis of the resulting state. Neural networks models are usually featured with multiple computational layers to learn a more abstract representation of the input. Our model is extended to have the structure of multiple layers (hops) by stacking memory addressing layer and memory reading layer repeatedly.

4.3.1 Input Representation

Each student response is represented as a vector in our model. Given a student response $x = \{x_1, x_2, x_3, ..., x_n\}$, where n is the length of the response, we map each word into a word vector $w_i = Wx_i$. All word vectors come from a word embedding matrix $W \in R^{d \times V}$, where d is the dimension of word vector and V is the vocabulary size. To represent an essay in a vector, we selected position encoding (PE) described in [SSWF15]. By the scheme of PE, the vector representation of a response is calculated by $m = \sum_j l_j \cdot Wx_{ij}$, where $\cdot$ is an element-wise multiplication. l_j is a column vector with the structure $l_{kj} = (1 - j/J) - (k/d)(1 - 2j/J)$ (assuming 1-based indexing), where J is the total number of words in the response, d is the dimension of word vector, and k is the embedding index. PE is a simple and efficient way to represent a response, and does not need to learn extra parameters.

Alternative way to represent a response is to feed each word vector from a response into Recurrent Neural Networks (RNN) [TN16]. Compared to traditional forward neural networks, hidden states of RNN are able to retain the sequential information. By feeding a response into RNN, all the information which are useful for the grading ideally should be stored in the hidden states. Instead of taking the last hidden state as the essay representation, it is recommended to calculate the mean of all hidden states to retrieve the representation for a long response.

4.3.2 Memory Addressing

After generating the representation of the responses, we select a sample from student response for every possible score, which is graded with the same score. The selected samples work as a representation of the criteria in the rubric for all possible scores. Expert knowledge can be used here to choose most representative sample for each score or even generate a number of ideal samples. The motivation is that the model is highly likely to distinguish the difference within the criteria for each score with these representative samples. For our experiment, we randomly pick a sample from student responses for each score, which is graded with that score.

All sampled responses are loaded into the memory as an array of vectors $m_1, m_2, ..., m_h$, where h is the total number of sampled essays. An ungraded response is denoted as x. The basic idea of memory addressing is that it assigns a weight/importance to each sampled response m_i by calculating a dot product between x and m_i followed by a softmax.

$$p_i = Softmax(xA^T \cdot m_i B^T) \tag{4.1}$$

where $Softmax(y_i) = \mathrm{e}^{y_i} / \sum_j \mathrm{e}^{y_j}$, A is a $k \times d$ matrix and so is B. Defined in this way p is a weight vector over all sampled responses. A and B are learned matrices used to transfer the response representation to a d-dimensional features space. The intuition is that the responses with the same grade are highly likely to have the similar representation in the feature space.

4.3.3 Memory Reading

After weight vector p is calculated, the output of the memory is computed as a weighted sum of each piece of memory in m:

$$o = \sum_i p_i m_i C^T \tag{4.2}$$

where C is a $k \times d$ matrix used to transfer the response representation to the feature space. The $k \times d$ matrix C may be identical to A, but from our experiment, we found that training a separate C leads to a better performance. From the equation, we can see that weight vector p controls the amount of content that is read from each memory piece.

4.3.4 Multiple Hops

The success of neural networks is due to its ability of learning multiple layers of neurons and each layer can transform the representation at previous level into a higher level of abstract representation. Inspired by this idea, we stack multiple memory addressing steps and memory reading steps together to handle multiple hops operations.

After receiving the output o from equation 4.2, the ungraded response u is updated with:

$$u_2 = Relu(R_1(u + o)) \tag{4.3}$$

where R_1 is a $k \times k$ matrix, $u = xA^T$ and $Relu(y) = max(0, y)$. Then memory addressing step and reading memory step are repeated, using a different matrix R_j on each hop j. The memory addressing step is modified accordingly to use the updated representation of the ungraded response.

$$p_i = Softmax(u_j \cdot m_i B) \tag{4.4}$$

4.3.5 Output Layer

After a fixed number H hops, the resulting state u_H is used to predict a final score over the possible scores:

$$\hat{s} = Softmax(u_H W + b) \tag{4.5}$$

where W is $k \times r$ matrix, r is the number of possible scores and b is the bias value. Note that the number of output nodes equals to the length of score range. We calculate a distribution over all possible scores and select most probable score as the prediction.

The whole network is trained in end-to-end fashion without any hand-engineered features, and the matrices A, B, C, W and $R_1, ..., R_H$ are learned through backpropagation and stochastic gradient descent by minimizing a standard cross entropy loss between the predicted score $\hat{s}$ and the actual score s.

4.4 Experimental Setup

4.4.1 Dataset

Dataset used in this study comes from Kaggle Automated Student Assessment Prize (ASAP) competition sponsored by William and Flora Hewlett Foundation (Hewlett). There are 8 sets of essays and each set is generated from a single prompt. All responses collected in the dataset were written by students ranging from grade 7 to grade 10. Score range varies on essay sets. All essays were graded by at least

Set	**# Essays**	**Avg. len.**	**Max len.**	**Min score**	**Max score**	**Mean score**
1	1,783	350	911	2	12	8
2	1,800	350	118	1	6	3
3	1,726	150	395	0	3	1
4	1,772	150	383	0	3	1
5	1,805	150	452	0	4	2
6	1,800	150	489	0	4	2
7	1,569	250	659	0	30	16
8	723	650	983	0	60	36

Table 4.1: Selected Details of ASAP dataset

2 human graders. The average length of the essays differs for each essay set, ranging from 150 words to 650 words. Selected details for each essay set is shown in Table 4.1.

4.4.2 Evaluation Metric

Quadratic weighted Kappa (QWK) is used to measure the agreement between the human grader and the model. We choose to use this metric because it is the official evaluation metric of the ASAP competition. Other work such as [CH13, TN16, PCN15] that uses the ASAP dataset also uses this evaluation metric. QWK is calculated using

$$k = 1 - \frac{\sum_{i,j} w_{i,j} O_{i,j}}{\sum_{i,j} w_{i,j} E_{i,j}} \tag{4.6}$$

where matrices O, w and E are the matrices of observed scores, weights, and expected scores respectively. Matrix $O_{i,j}$ corresponds to the number of student responses that receive a score i by the first grader and a score j by the second grader (the model in our experiment). The weight matrix are $w_{i,j} = (i-j)^2/(N-1)^2$, where N is the number of possible scores. Matrix E is calculated by taking the outer product between the score vectors of the two graders, which are then normalized to have the same sum as O.

4.4.3 Implementation Details

To help other researchers to replicate our results, the details of implementing and training the model are articulated below. We plan to publish our source code once the paper is accepted. The model was implemented using Tensorflow framework [MAP+15]. We used Adaptive Moment Estimation (Adam) stochastic gradient descent [KB14] for optimizing the learned parameters. Compared to normal stochastic gradient descent, Adam computes adaptive learning rate for each parameter and empirical results has shown that Adam achieves better outcomes. The learning rate was set to 0.002 and batch size for each iteration to 32 for all models. As final prediction layer, we used a fully connected layer on top of output from memory reading layer with a softmax activation function. The model learned the parameters by minimizing a standard cross-entropy loss between predicted score and the correct score.

For regularization we used L2 loss on all learned parameters with lambda set to 0.3 and limited the norm of the gradients to be below 10. Moreover, we added gradient noise sampled from a Gaussian distribution with mean 0 and variance 0.001 when training the memory networks.

We used the publicly available pre-trained Glove word embeddings [PSM14], which was trained on 42 billion tokens of web data, from Common Crawl [2]. The dimension of each word vector is 300. Word2vec [MSC+13] is another popular word embedding algorithm and pre-trained word embeddings are also publicly available from this algorithm. As results shown in [PSM14], Glove outperforms word2vec on word analogy, word similarity, and named entity recognition tasks.

[2]http://commoncrawl.org

5-fold cross validation was used to evaluate our model. For each fold, the data was split into two parts: 80% of the data as the training data and 20% as the testing data. The sampled response for each score is selected from the training data. A model was trained on each essay set due to the fact that score range varies among 8 essay sets.

4.4.4 Baselines

We compare our model with Enhanced AI Scoring Engine (EASE), an open-source AES system, to demonstrate the improvements on performance. EASE, like traditional NLP techniques, requires fine-grained hand-engineered features and builds a regression model on top of these features. The reason we use this system as one of baselines is that it achieved best QWK scores among all open-source systems participated in ASAP competition. [PCN15] described a set of reliable features and reported the results of two models using these features: support vector regression (SVR) and Bayesian linear ridge regression (BLRR).

[TN16] examined several neural networks models, e.g. RNN and Convolutional Neural Networks (CNN), on ASAP dataset. In their experiments, Long Short Term Memory networks (LSTM), a variant of RNN, achieved the best performance. LSTM is designed to have three gates in each hidden node: input gate, forget gate, and output gate. By controlling these three gates, LSMT has the capability of attaining long-term dependencies. Applying LSTM to ASAP dataset leads to the better performance compared to EASE. LSTM becomes a stronger baseline for our model. The structure of the LSTM model described in [TN16] is presented in Figure 4.2.

As mentioned above, LSTM is an alternative approach of learning the representation of an essay. But LSTM is more computationally expensive than position encoding (PE) is. We run our model with LSTM instead of PE on ASAP dataset

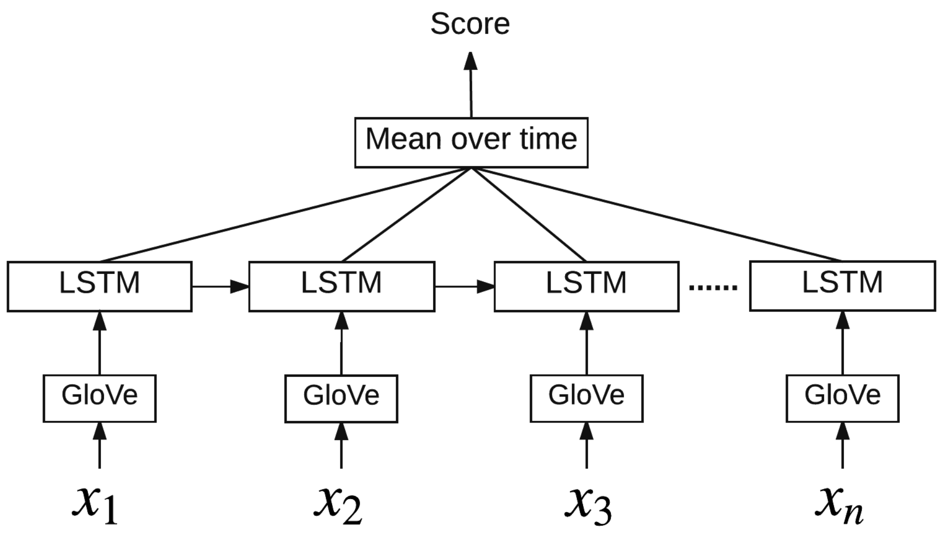

Figure 4.2: An illustration of baseline LSTM model for AES

and find that the performance suffers in the case of LSTM. In this paper we only report the results of our model with PE.

To verify the efficacy of GloVe word embeddings and external memory, we developed a simple multi-layer forward neural networks (FNN) model, which is similar to our model with respect to the model structure, but without an external memory. We refer this baseline model as FNN for the rest of paper for convenience. As shown in Figure 4.3, each word of a student response is first converted to a continuous vector using GloVe word embeddings. The vector representation for the response is obtained by applying PE on all word vectors from the response. Afterward the representation is fed into 4 hidden layers, each of which has 100 hidden nodes. Apply a softmax operation on the resulting states of last hidden layer at output layer to predict the final score. The model is also trained using Adam Optimizer by minimising the standard cross entropy between $\hat{s}$ and truth score s. FNN is properly

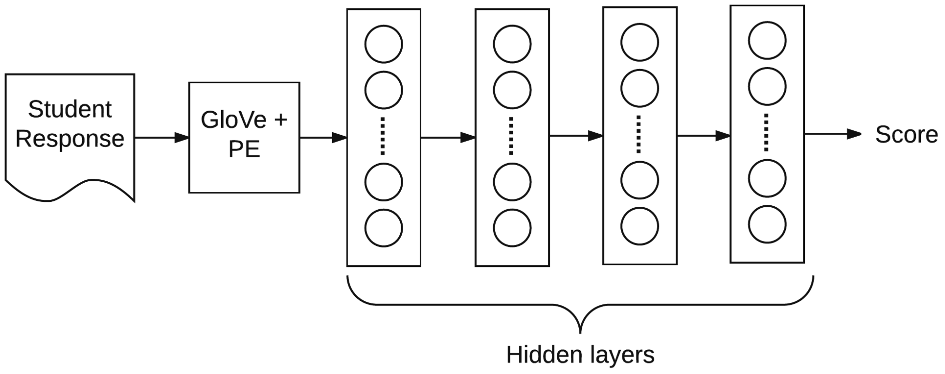

Figure 4.3: An illustration of baseline FNN. Use GloVe with PE to represent a student response. The representation is fed into 4-layer networks and each layer has 100 hidden nodes.

defined by the equations below:

$$h_0 = Relu(A^T x) \tag{4.7}$$

$$h_i = Relu(R_i h_{i-1}),\ for\ i \geq 1 \tag{4.8}$$

$$\hat{s} = Softmax(h_H W) \tag{4.9}$$

where x is the representation generated by GloVe with PE for a student response. h_i is the output of hidden layer i. H is the total number of hidden layers. A, R_i ,and W are weight matrices. The bias vectors are omitted in the equations.

4.5 Results

We first report the QWK scores of the ASAP dataset by varying the number of hops in our model in Table 4.2. We see that the average of the QWK scores across 8 sets remains the same from 1 hop to 3 hops and the average decreases a little for 4 hops

Set	**# Hops**				
	1	2	3	4	5
1	0.83	0.83	0.83	0.84	0.71
2	0.72	0.72	0.71	0.69	0.67
3	0.73	0.73	0.72	0.72	0.72
4	0.81	0.82	0.81	0.81	0.81
5	0.83	0.83	0.83	0.83	0.82
6	0.83	0.83	0.82	0.82	0.82
7	0.8	0.8	0.81	0.78	0.77
8	0.71	0.71	0.69	0.67	0.66
avg	0.78	0.78	0.78	0.77	0.75

Table 4.2: QWK scores vs. number of hops in our model.

and 5 hops. With 2 hops, set 2, 3, 4, 5, 6 and 8 have the best scores. Set 1 has the best score with 4 hops, and set 7 has the best score with 3 hops. We see that the performance of the model is insensitive to the number of hops.

We then compare our results on 2 hops with the best results from other baselines mentioned above in Table 4.3. Column MN presents the QWK scores from our model. Column EASE (SVR) and column EASE(BLRR) contain the results from EASE with two different regression methods. We also compare our model with other neural models in [TN16] and the best results from [TN16] is listed in column LSTM+CNN of Table 4.3. Note that their best results reported in the paper are obtained by ensembling results from 10 runs of LSTM and 10 runs of CNN. However, in our experiment, the results are recorded from a single run of a single model after optimizing the hyperparameters. This is not fair to compare their best results with ours directly. Therefore we also pick the best performance achieved by a single model from their paper and list in Column LSTM of Table 4.3. In their setup, the number of hidden nodes in LSTM is 300 and pre-trained word embeddings released by [ZSCM13] is used.

As indicated in Table 4.3, our model outperforms in 7 out of 8 sets (except

Set	**MN**	**FNN**	**EASE(SVR)**	**EASE(BLRR)**	**LSTM**	**LSTM+CNN**	**Human**
1	**0.83**	0.75	0.78	0.76	0.78	0.82	0.72
2	**0.72**	0.7	0.62	0.61	0.69	0.69	0.81
3	**0.73**	0.7	0.63	0.62	0.68	0.69	0.77
4	**0.82**	0.8	0.75	0.74	0.8	0.81	0.85
5	**0.83**	0.8	0.78	0.78	0.82	0.81	0.75
6	**0.83**	0.79	0.77	0.78	0.81	0.82	0.78
7	0.8	0.73	0.73	0.73	**0.81**	**0.81**	0.72
8	**0.71**	0.63	0.53	0.62	0.59	0.64	0.63
Avg	**0.78**	0.74	0.7	0.71	0.75	0.76	0.75

Table 4.3: QWK scores on ASAP dataset.

for set 7) and improves the average QWK score by 4.0% compared to the baseline LSTM. Even compared to their best ensembled model (LSTM+CNN), our model still achieved better performance in 7 essay sets (except for essay 7). As expected, our model surpasses EASE in all 8 sets and improves average QWK score by 10%.

The results from the FNN model mentioned above is presented in column FNN of Table 4.3. In our experiments, FNN has 4 hidden layers and each layer has 100 hidden nodes, whose structure is similar to that of our model except that the external memory is removed. When comparing these results to the best results from EASE, we find that this basic model outperforms EASE in 7 out of 8 sets of essays (except for essay set 1) and is even comparable with the complex model (LSTM). This proves that using Glove word embeddings with PE to represent a student response is able to capture important features useful for grading the response. The effectiveness of the external memory is proved by the fact that MN accomplishes better performance on 7 sets (set 4 is equal) than FNN does. The comparison between FNN and other models indicates that representing a student response using GloVe with PE and adding external memory are two key factors which may lead to the good performance on ASAP dataset.

In ASAP dataset, two human graders are assigned to each student response and each grader gives a score separately. The final gold-standard score for each response is calculated based on these two scores. In Column Human of Table 4.3, we calculated the QWK scores between these two graders to measure the agreement between two graders.

4.6 Discussion and Conclusion

In this study, we develop a generic model for automated grading tasks using memory networks and word embeddings. To our best knowledge this is the first study that memory networks are applied for this kind of task. Our model is tested on ASAP dataset and achieves state-of-the-art performance in 7 out of 8 essay sets. Similar to other neural networks models for AES, our model can be trained in an end-to-end fashion and does not require any hand-engineered features. Compared to RNN, CNN, using GloVe word embeddings with PE to represent a student response makes our model simple and cost-effective. Adding external memory improves the performance over FNN model, which means our model is able to take advantage of sampled responses stored in the external memory.

Our model can be generalized to automatically grade assignments from other subjects. As shown above, there are two key factors to the performance: reliable representation and memory component. In order to apply our model to other kinds of assignment, learning a good vector representation for the assignment is the first step. It is analogous to how the regression model is built for supervised NLP tasks: first extract numerical hand-engineered features from text and then apply a regression model on these generated features to predict true labels. In the context of neural networks, a vector is required to represent the student response. Learning the vector can be a part of the predictive model. For example, the word embeddings in [TN16] are learned from their predictive model. These vectors can also come from pre-trained models, like GloVe and word2vec. The next step is to select characterized samples and store these samples to memory. The purpose of this step is to teach the model to understand the grading strategy and eventually associate a vector representation to a score.

However, we only test our model on one dataset. There is a need to explore our model with more datasets that contain various formats of assignments to verify our model. Furthermore, the representation of the assignment and the mechanism for measuring relevance among assignments is still elementary. Future work should therefore focus on these two areas to improve the generalizability of the model. A lot of effort is still needed to better interpret memory networks and explain the key factors behind our performance improvement.

Part III

Counterfactual Inference

Chapter 5

Estimating Individual Treatment Effects with Residual Counterfactual Networks

Proper citation of this chapter is as follows:
Siyuan Zhao and Neil T. Heffernan. Estimating Individual Treatment Effect from Educational Studies with Residual Counterfactual Networks. In *Proceedings of the 10th International Conference on Educational Data Mining, EDM 2017*

Personalized learning considers that the causal effects of a studied learning intervention may differ for the individual student. Making the inference about causal effects of studies interventions is a central problem. In this paper we propose the Residual Counterfactual Networks (RCN) for answering counterfactual inference questions, such as "Would this particular student benefit more from the video hint or the text hint when the student cannot solve a problem?". The model learns a balancing representation of students by minimizing the distance between the distributions of the control and the treated populations, and then uses a residual block to estimate the individual treatment effect based on the representation of the student. We run experiments on semi-simulated datasets and real-world educational online experiment datasets to evaluate the efficacy of our model. The results show that our model matches or outperforms the state-of-the-art.

5.1 Introduction

The goal of personalized learning is to provide pedagogy, curriculum, and learning environments to meet the needs of individual students. For example, an Intelligent Tutor System (ITS) decides which hints would most benefit a specific student. If the ITS could infer what the student performance would be after receiving each hint, then it would simply choose the hint which leads to the best performance for the student. To make this possible, we might run an online educational experiment by randomly assigning students to one of the hints, and collect student performance. Then making predictions about causal effects of possible interventions (e.g. available hints) becomes a central problem in this case. In this paper we focus on the task of answering counterfactual questions [Pea09] such as, ”Would this particular student benefit more from the video hint or the text hint when the student cannot solve a problem?”

There are two ways of collecting data for counterfactual inference: randomized control trials (RCTs) and observational studies. In RCTs, participants (e.g. students) are randomly assigned to interventions (e.g. video hints or text hints), while participants in observational studies are not essentially randomly assigned to interventions. For example, consider the experiment of evaluating the efficacy of video hints and text hints for a certain problem. Under the design of RCT, students who need a hint would be randomly assigned to either the video hints or the text hints. In an observational study, students are assigned to one of the interventions based on their contextual information, such as knowledge level or personal preference. RCTs are expensive in terms of time and money compared to observational studies.

[JSS16] proposed Balancing Neural Networks (BNN) which can be applied to solve the counterfactual inference problem. They used a form of regularizer to enforce the similarity between the distributions of representations learned for populations with different interventions, for example, the representations for students who received text hints versus those who received video hints.This reduces the variance from fitting a model on one distribution and applying it to another. Because of random assignment to the interventions in RCTs, the distributions of the populations within different interventions are highly likely to be identical. However, in the observational study, we may end up with the situation where only male students receive video hints and female students receive text hints. Without enforcing the similarity between the distributions of representations for male and female students, it is not safe to make a prediction of the outcome if male students receive text hints. In machine learning, "domain adaptation" refers to the dissimilarity of the distributions between the training data and the test data.

Neural networks have been shown to successfully learn good representation of high-dimensional data in several tasks [BCV13]. Recent work [LZWJ16] has demonstrated that (deep) neural networks can be used with domain adaptation approaches to produce outstanding results on some domain adaptation benchmark datasets. These successful methods encourage similarity between the deep features representations w.r.t the different domains. This similarity is often enforced by minimizing a certain distance between the networks' domain-specific hidden features.

Motivated by their work, we propose the Residual Counterfactual Networks

(RCN) for the counterfactual inference to estimate the individual treatment effect and evaluate its efficacy in both a simulated dataset and a real-world dataset from an educational online experiment. The RCN extends the BNN by adding a residual block to estimate the individual treatment effect (ITE) based on the learned representation of participants. The idea of the residual block is originated from the state-of-the-art deep residual learning [HZRS16]. We enable the estimation of ITE by plugging several layers into neural networks to explicitly learn the residual function with reference to the learned representation.

The rest of the paper is organized as follows. Section 2 provides an overview of the problem setup of counterfactual inference for estimating the ITE. Section 3 details information of our model. Section 4 gives an overview of related work in this research area. Section 5 describes the datasets and evaluation metrics used to test our model. Section 6 presents the results of our model and compares them with other models. Finally, we discuss the results and conclude the paper.

5.2 Problem Setup

Let $\mathcal{T}$ be the set of proposed interventions we wish to consider, X the set of participants, and Y the set of possible outcomes. For each proposed intervention $t \in \mathcal{T}$, let $Y_t \in Y$ be the potential outcome for x when x is assigned to the intervention t. In randomized control trial (RCT) and observed study, only one outcome is observed for a given participant x; even if the participant is given an intervention and later the other, the participant is not in the same state. In machine learning, "bandit feedback" refers to this kind of partial feedback. The model described above is also known as the Rubin-Neyman causal model [Rub05, Rub74].

We focus on a binary intervention set $\mathcal{T} = \{0, 1\}$, where intervention 1 is often referred as the "treated" and intervention 0 is the "control." In this scenario the ITE for a participant x is represented by the quantity of $Y_1(x) - Y_0(x)$. Knowing the quantity helps assign participant x to the best of the two interventions when making a decision is needed, for example, choosing the best intervention for a specific student when the student has a trouble solving a problem. However, we cannot directly calculate ITE due to the fact that we can only observe the outcome of one of the two interventions.

In this work we follow the common simplifying assumption of no-hidden confounding variables. This means that all the factors determining the outcome of each intervention are observed. This assumption can be formalized as the strong ignorability condition:

$$(Y_1, Y_0) \perp t|x, 0 < p(t = 1|x) < 1, \forall x.$$

Note that we cannot evaluate the validity of strong ignorability from data, and the validity must be determined by domain knowledge.

In the "treated" and the "control" setting, we refer to the observed and unobserved outcomes as the factual outcome $y^F(x)$, and the counterfactual outcome $y^{CF}(x)$ respectively. In other words, when the participant x is assigned to the "control" $(t = 0)$, $y^F(x)$ is equal to $Y_1(x)$, and $y^{CF}(x)$ is equal to $Y_0(x)$. The other way around, $y^F(x)$ is equal to $Y_0(x)$, and $y^{CF}(x)$ is equal to $Y_1(x)$.

Given n samples $\left\{(x_i, t_i, y_i^F)\right\}_{i=1}^n$, where $y_i^F = t_i \cdot Y_1(x_i) + (1 - t_i)Y_0(x_i)$, a common approach for estimating the ITE is to learn a function $f : X \times T \to Y$ such that

$f(x_i, t_i) \approx y_i^F$. The estimated ITE is then:

$$I\hat{T}E(x_i) = \begin{cases} y_i^F - f(x_i, 1-t_i), & t_i = 1. \\ f(x_i, 1-t_i) - y_i^F, & t_i = 0. \end{cases}$$

We assume n samples $\{(x_i, t_i, y_i^F)\}_{i=1}^n$ form an empirical distribution $\hat{p}^F = \{(x_i, t_i)\}_{i=1}^n$. We call this empirical distribution $\hat{p}^F \sim p^F$ the empirical factual distribution. In order to calculate ITE, we need to infer the counterfactual outcome which is dependent on the empirical distribution $\hat{p}^{CF} = \{(x_i, 1-t_i)\}_{i=1}^n$. We call the empirical distribution $\hat{p}^{CF} \sim p^{CF}$. The p^F and p^{CF} may not be equal because the distributions of the control and the treated populations may be different. The inequality of two distributions may cause the counterfactual inference over a different distribution than the one observed from the experiment. In machine learning terms, this scenario is usually referred to as domain adaptation, where the distribution of features in test data are different than the distribution of features in training data.

5.3 Model

We proposed RCN to estimate individual treatment effect using counterfactual inference. The RCN first learns a balancing representation of deep features $\Phi : X \to R^d$, and then learns a residual mapping Δf on the representation to estimate the ITE. The structure of the RCN is shown in Figure 5.1.

To learn a representation of deep features Φ, the RCN uses fully connected layers with ReLu activation function, where $Relu(z) = max(0, z)$. We need to generalize from factual distribution to counterfactual distribution in the feature representation Φ to obtain accurate estimation of counterfactual outcome. The common successful approaches for domain adaptation encourage similarity between the latent feature

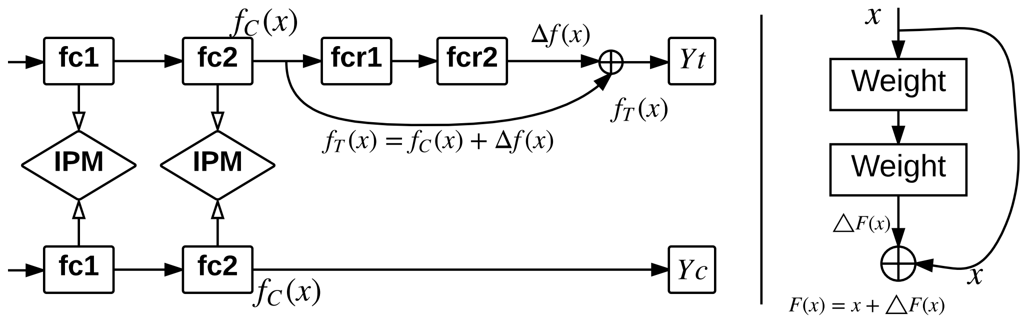

Figure 5.1: Residual Counterfactual Networks for counterfactual inference. IPM is adopted on layers fc1 and fc2 to minimize the discrepancy distance of the deep features of the control and the treated populations. For the treated group, we add a residual block fcr1-fcr2 so that $f_T(x) = f_C(x) + \Delta f(x)$

representations w.r.t the different distributions. This similarity is often enforced by minimizing a certain distance between the domain-specific hidden features. The distance between two distributions is usually referred to as the discrepancy distance, introduced by [MMR09], which is a hypothesis class dependent distance measure tailored for domain adaptation.

In this paper we use an Integral Probability Metric (IPM) measure of distance between two distributions $p_0 = p(x|t = 0)$, and $p_1 = p(x|t = 1)$, also known as the control and treated distributions. The IPM for p_0 and p_1 is defined as

$$\text{IPM}_{\mathcal{F}}(p_0, p_1) := \sup_{f \in \mathcal{F}} \left| \int_S f dp_0 - \int_S f dp_1 \right|,$$

where $\mathcal{F}$ is a class of real-valued bounded measurable functions on S.

The choice of functions is the crucial distinction between IPMs [SFG$^+$09]. Two specific IPMs are used in our experiments: the Maximum Mean Discrepancy (MMD), and the Wasserstein distance. $\text{IPM}_{\mathcal{F}}$ is called MMD, when $\mathcal{F} = \{f : \|f\|_{\mathcal{H}} \leqslant 1\}$, where $\mathcal{H}$ represents a reproducing kernel Hilbert space (RKHS) with k as its reproducing kernel. In other words, the family of norm-1 reproducing kernel Hilbert space (RKHS) functions lead to the MMD. The family of 1-Lipschitz functions $\mathcal{F} = \{f : \|f\|_L \leq 1\}$, where $\|f\|_L$ is the Lipschitz semi-norm of a bounded continuous real-valued function f, make IPM the Wasserstein distance. Both the Wasserstein and MMD metrics have consistent estimators which can be efficiently computed in the finite sample case [SFG$^+$12]. The important property of IPM is that

$$p_0 = p_1 \text{ iff } \mathrm{IPM}_{\mathcal{F}}(p_0, p_1) = 0.$$

The representation with reduction of the discrepancy between the control and the treated populations helps the model to focus on balancing features across two populations when inferring the counterfactual outcomes. For instance, if in an experiment, almost no male student ever received intervention A, inferring how male students would react to intervention A is highly prone to error and a more conservative use of the gender feature might be warranted.

After balancing the feature representations of the control and the treated populations, the next step is to infer the treatment effect for participant x. We adopt the residual block [HZRS16] to estimate the treatment effect.

As shown in Figure 5.2, $F(x)$ is the underlying desired function mapping. Instead of stacking a number of layers to fit the desired $F(x)$, we let stacked fully connected layers learn the residual mapping $\Delta f(x) = F(x) - x$. Then the origin mapping is converted into $\Delta f(x) + x$. The operation $\Delta f(x) + x$ is performed by a shortcut connection and an element-wise addition. Learning residual mapping is favored over fitting the desired mapping directly, because it is easier to find the residual with reference to an identity mapping than to learn the mapping as new.

The goal of the residual block is to approximate a residual function Δf such that

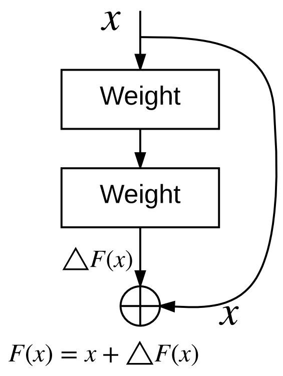

Figure 5.2: Residual block

$f_T(x) = f_C(x) + \Delta f(f_C(x))$, where f_C is the deep representation of participant x before being fed into the output layer, and f_T is the input to the output layer for the treated population. The output layer is a ridge linear regression to generate the final outcome. From the definition of the residual function Δf, we see that $\Delta f(x)$ is the estimated treatment effect for participant x, which is our interest in a control and treated experiment. With the residual block directly connected to fc2, the residual function $\Delta f(x)$ is dependent on the feature representation of participant x.

We plug in the residual block (shown in Figure 5.1) between fc2 layer and final output layer for the treated population in order to estimate the ITE. There is no residual block plugged in between fc2 layer and the final output layer for the control population. The final output layer $\varphi(\cdot)$ is a linear regression to calculate the predicted outcome, such that $Yc = \varphi(f_C(x))$, and $Yt = \varphi(f_T(x))$.

Recall the problem setup described above that there exist n samples $\left\{(x_i, t_i, y_i^F)\right\}_{i=1}^{n}$, where $y_i^F = t_i \cdot Y_1(x_i) + (1 - t_i)Y_0(x_i)$. In the control and the treated setting, we assume that $n_c (n_c > 0)$ samples $\left\{(x_i, 0, y_i^{(0)})\right\}_{i=1}^{n_c} \sim D_c$ are assigned to the control $(t = 0)$, and $n_t (n_t > 0)$ samples $\left\{(x_i, 1, y_i^{(1)})\right\}_{i=1}^{n_t} \sim D_t$ are assigned to the treated $(t = 1)$, such that $n = n_c + n_t$. As described above, RCN is an integration of deep feature learning, feature representation balancing, and treatment effect estimation in an end-to-end fashion with the loss function as such:

$$\begin{aligned}\min_{f_T = f_S + \Delta f(f_S)} \; & \frac{1}{n_c}\sum_{i=1}^{n_c} L(f_c(\mathbf{x}_i), y_i^{(0)}) \\ & + \frac{1}{n_t}\sum_{i=1}^{n_t} L(f_t(\mathbf{x}_i), y_i^{(1)}) \\ & + \lambda \cdot \mathrm{IPM}(D_c, D_t),\end{aligned}$$

where λ is the tradeoff parameter for the IPM penalty, L is the loss function of the model. In the case of binary classification, L is the standard cross entropy. In the case of regression, L is root-mean-square error (RMSE). During the training, the model only has the access to the factual outcome.

5.4 Related Work

From a conceptual point of view, our work is inspired by the work on domain adaptation and deep residual learning. [LZWJ16] proposed the Residual Transfer Network that adopt MMD distance to learn transferable deep features from labeled data in the source domain and unlabeled data in the target domain and adds a residual block to transfer the prediction classifier from the target domain to the source domain. The structure of our model is similar to that of their model. Deep residual learning is introduced by [HZRS16], the winner of the ImageNet ILSVRC 2015 challenge, to ease the training of deep networks. The residual block is designed to learn residual functions $\Delta F(\mathbf{x})$ with reference to the layer input $\mathbf{x}$. Reformulating layers to the residual block makes the training easier than directly learning the original functions $F(\mathbf{x}) = \Delta F(\mathbf{x}) + \mathbf{x}$.

Our model extends the work by [JSS16, SJS16], where the authors build a connection between domain adaptation and counterfactual inference. They use IPMs, such as MMD and wasserstein distance, to learn a representation of the data which balances the control and treated distributions. The treatment assignment is concatenated with the representation to predict the factual outcome as while the reverse treatment assignment is concatenated with the representation to predict the counterfactual outcome. Compared to their work, we add a residual block to estimate the individual treatment effect based on the representation.

[WA17, AI16] proposed causal forests which is built upon the idea of random forests to estimate the heterogeneous treatment effect with semi-parametric asymptotic convergence rate.

ASSISTments is a platform that combines large-scale online education (like Khan Academy & MOOCs) with technology for rigorous scientific research using randomized experiments and data mining. It has been used in over 20 published randomized controlled experiments to investigate different ways to improve student learning. An experiment was conducted by [RH06] to determine if students benefitted more if they were given the scaffolds versus just being given hints that tried to provide them the same information that the scaffolding questions asked them. [OH15] examined adding student preference to the ASSISTments platform. The purpose of this experiment was to see whether providing students a choice in feedback style would alter performance and learning gains.

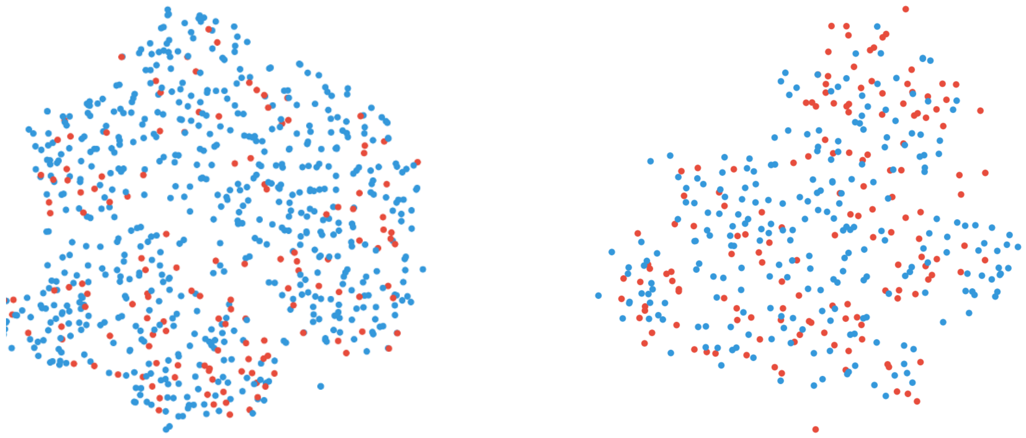

Figure 5.3: Visualization of the IHDP dataset by t-SNE (left). Visualization of the ASSISTments dataset by t-SNE (right). Each dot represents a data point. The blue means the data point from the control while the red means the data point from the treatment.

5.5 Experiments

5.5.1 Evaluation Metrics

To compare among various models, we report the RMSE of estimated individual treatment effect, denoted

$$\epsilon_{ITE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}((Y_1(x_i) - Y_0(x_i)) - I\hat{T}E(x_i))^2},$$

and the absolute error in average treatment effect

$$\epsilon_{ATE} = \left|\frac{1}{n}\sum_{i=1}^{n}(f_t(x_i) - f_s(x_i)) - \frac{1}{n}\sum_{i=1}^{n}(Y_1(x_i) - Y_0(x_i))\right|.$$

Following [Hil11, JSS16], we report the Precision in Estimation of Heterogeneous

Effect (PEHE),

$$PEHE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}((Y_1(x_i) - Y_0(x_i)) - (\hat{y_1}(x_i) - \hat{y_0}(x_i))^2}.$$

Compared to the fact that achieving a small RMSE of estimated ITE needs the accurate estimation of counterfactual responses, a good (small) PEHE requires the accurate estimation of both factual and counterfactual responses.

However, calculating ϵ_{ITE}, ϵ_{ATE}, and PEHE requires the "ground truth" of the ITE for each participant in the experiment. We cannot gather the counterfactual outcomes from RCTs and observational studies, and thus do not have the ITE of each participant. We cannot evaluate ϵ_{ITE} and PEHE on these datasets. In order to evaluate the performance on these datasets across various models, we use a measure, called policy risk, introduced by [SJS16]. Given a model f, the participant x is assigned to the treatment $\pi_f(x) = 1$ if $f(x,1) - f(x,0) > \lambda$ (in the case of RCN, $\Delta f > \lambda$), where λ is the treatment threshold, and to the control $\pi_f(x) = 0$ otherwise. The risk policy is defined as:

$$\begin{aligned} R_{Pol}(\pi_f) = 1 - (&\mathbb{E}[Y_1|\pi_f(x) = 1] \cdot p(\pi_f = 1) \\ &+ \mathbb{E}[Y_0|\pi_f(x) = 0] \cdot p(\pi_f = 0)). \end{aligned}$$

The empirical estimator of the risk policy on a dataset is calculated by:

$$\begin{aligned} \hat{R}_{Pol}(\pi_f) = 1 - (&\mathbb{E}[Y_1|\pi_f(x) = 1, t = 1] \cdot p(\pi_f = 1) \\ &+ \mathbb{E}[Y_0|\pi_f(x) = 0, t = 0] \cdot p(\pi_f = 0)). \end{aligned}$$

To obtain the policy risk, we select a subset of participants in the dataset where

Table 5.1: Hypothetical data for some example students. The predicted outcome is the probability that the student would complete the assignment. Students in bold are those whose randomized treatment assignment is congruent with the recommendation of the counterfactual inference model. Data from these students would be used to calculate the policy risk.

ID	Group	Completion	Predicted outcome if treated	Predicted outcome if not treated	Treatment effect	Treat?
1	Control	1	0.8	0.75	0.05	1
2	**Control**	**0**	**0.3**	**0.45**	**-0.15**	**0**
3	**Treatment**	**0**	**0.50**	**0.38**	**0.12**	**1**
4	**Control**	**1**	**0.78**	**0.9**	**-0.12**	**0**
5	**Treatment**	**1**	**0.9**	**0.6**	**0.3**	**1**
6	Treament	1	0.91	0.99	-0.08	0
7	Control	0	0.83	0.70	0.13	1
8	**Control**	**1**	**0.73**	**0.83**	**-0.1**	**0**

the treatment recommendation inferred by the model is the same as the treatment assignment in the experiment and then calculate the average loss from the subset of the data.

For the datasets without the "ground truth" on ITE, we also calculate the average treatment effect on the treated by ATT $= \frac{1}{n_t}\sum_{i=1}^{n_t} y_i^{(1)} - \frac{1}{n_s}\sum_{i=1}^{n_s} y_i^{(0)}$, and report the error on ATT as $\epsilon_{ATT} = \left|\text{ATT} - \frac{1}{n_t}\sum_{i=1}^{n_t}(f_t(x_i) - f_s(x_i))\right|$.

5.5.2 Baselines

Balancing Neural Networks (BNN) is another neural networks-based model for counterfactual inference. Compared to RCN, it has exactly the same fc1 and fc2 layers with IPM regularizer to learn the representation $\Phi(x)$ of the participant x. However, instead of using residual block to estimate treatment effect, it concatenates the treatment assignment t_i to the output of fc2 layer $\Phi(x)$ and feeds $[\Phi(x_i), t_i]$ to another two fully connected layers to generate the predicted outcome. We refer to this particular structure of BNN as BNN-2-2, following [JSS16].

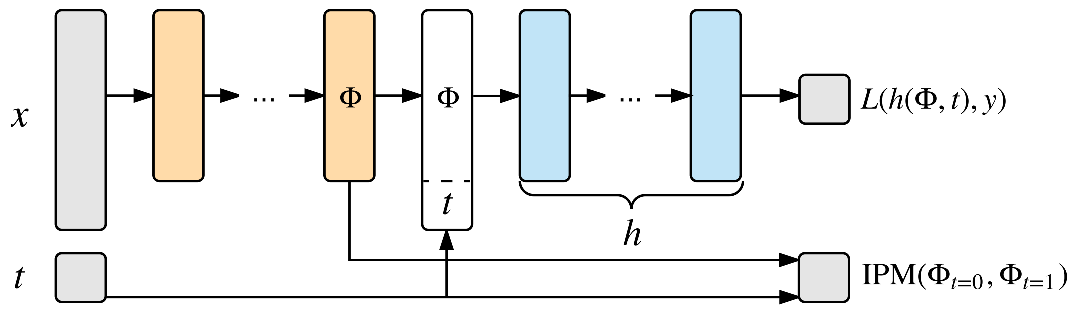

Figure 5.4: CFR for ITE estimation. L is a loss function, IPM is an integral probability metric

The Counterfactual Regression (CFR) [SJS16] is built on the BNN. The important difference between these two models is that the CFR uses a more powerful distribution metric in the form of IPMs to learn a balancing representation. We compare our model with BNN-2-2 and CFR to verify the efficacy of residual block in terms of estimating individual treatment effect.

We introduce a simple neural networks baseline model to evaluate the efficacy of the IPM regularizer and residual mapping. This baseline model is a feed-forward neural networks model with four hidden layers, trained to predict the factual outcome based on X and t, without the IPM regularizer and the residual block. We refer to this as NN-4.

5.5.3 Simulation based on real data - IHDP

The Infant Health and Development Program (IHDP) dataset was a semi-simulated dataset introduced by [Hil11]. The dataset consists of a number of covariates from a real randomized experiment. The goal of the experiment is to study the impact of superior child care and home visits on future cognitive test scores. [Hil11] discarded a biased subset of the treated population in order to introduce imbalance between treated and control subjects and used a simulated counterfactual outcome. Eventually, there are 747 subjects (139 treated, 608 control), each represented by 25 covariates assessing the attributes of the children and their mothers.

5.5.4 ASSISTments dataset

The ASSISTments online learning platform is a free web-based platform utilized by a large user-base of teachers and students. The system, based primarily in math content, allows teachers to assign several types of assignments for classwork and homework, reporting on student performance and learning progress. Students are given immediate feedback on each problem, and are also presented with several forms of instructional aid including hints, that provide a useful message, and scaffolded questions that break down the problem into smaller steps. The platform has been the subject of a recent study within the state of Maine [RFMM16], demonstrating significant learning gains for students using the platform.

The dataset used in this work comes from one of 22 randomized controlled experiments [SPH16] collected within the platform. This experiment was run in assignment types known as "skill builders" in which students are given problems until a threshold of understanding is reached; within ASSISTments, this threshold is traditionally three consecutive correct responses. Reaching this threshold denotes sufficient performance and completion of the assignment. In addition to this experimental data, information of the students prior to condition assignment is also provided in the form of problem-level log data providing a breadth of student information at fine levels of granularity.

In this experiment, there are two kinds of hints (video versus text) available for each problem from the assignment when students answer the problem incorrectly. The assignment to the video hint and the text video was random. Video content was

Table 5.2: IHDP. Results and standard errors for 1000 repeated experiments

Model	ϵ_{ITE}	ϵ_{ATE}	**PEHE**
NN-4	2.0 ± 0.0	0.5 ± 0.0	1.9 ± 0.1
BNN-2-2	1.7 ± 0.0	0.3 ± 0.0	1.6 ± 0.1
CFR	1.4 ± 0.0	0.2 ± 0.0	1.6 ± 0.1
RCN	1.1 ± 0.0	0.05 ± 0.0	1.4 ± 0.1

designed to mirror text hint in an attempt to provide identical assistance. There are 147 students who received the video hint and 237 students who received the text hint. The dataset includes 15 covariates such as student past-performance history, class-past performance history. We solve a binary classification task which is to predict the completion of the assignment for each student.

The visualization of IHDP dataset and ASSISTments dataset by t-Distributed Stochastic Neighbor Embedding (t-SNE) [MH08] is shown in Figure 5.3. The t-SNE is non-parametric visualization technique that can reveal hidden structures in the data by giving each high-dimensional data point a location in a two or three-dimensional map. We see that the control and the treatment populations in both datasets are not completely separated from each other. The ASSISTments dataset is slightly more balanced than the IHDP dataset.

5.6 Results

The results of IHDP is presented in Table 5.2 when the treatment threshold $\lambda = 0$. We see that our proposed RCN performs the best on the dataset in terms of estimating ITE, ATE and PEHE. There is an especially large improvement on estimating ITE. These results indicate that the residual block $\Delta f(x)$ helps accurately predict the value of ITE based on the feature representation $\Phi(x)$ for a given participant x.

The results of ASSISTments dataset are the interest of our work since we hope to apply the RCN to educational experiments in order to support decision making in terms of personalized learning. The results in terms of policy risk and the average treatment effect on the treated are shown in Table 5.3 when the treatment threshold $\lambda = 0$. The model TA means "Treated All" where all students are assigned to the treatment while the model NT means "Not Treated" where all students are assigned to the control. Without considering that the effects of an intervention may differ for individual students, the model with the better performance out of these two models would be adopted when a choice must be made between these two interventions. The RCN, which considers the individual treatment effect, outperforms the TA and the NT. This indicates that taking the individual effect into account helps make a better choice of interventions. The comparison between the CFR and the RCN suggests that the RCN performs better than the CFR does in terms of risk policy and ATT.

To investigate the correlation between policy risk and treatment threshold λ, we plot the value of policy risk as a function of treatment threshold λ in Figure 5.5, and the histogram of the predicted ITE from the RCN and the CFR on the ASSISTments dataset in Figure 5.6 and Figure 5.7 respectively. For the results of the ASSISTments dataset from the CFR, the maximum predicted ITE in the dataset is 0.44. Once the threshold λ is larger than 0.44, the CFR is converted to "Not Treated" where all students are assigned to the control. Since the maximum predicted ITE in the ASSISTments dataset from the CFR is 0.18, the CFR is converted to "Not Treated" once the treatment threshold λ is larger than 0.18.

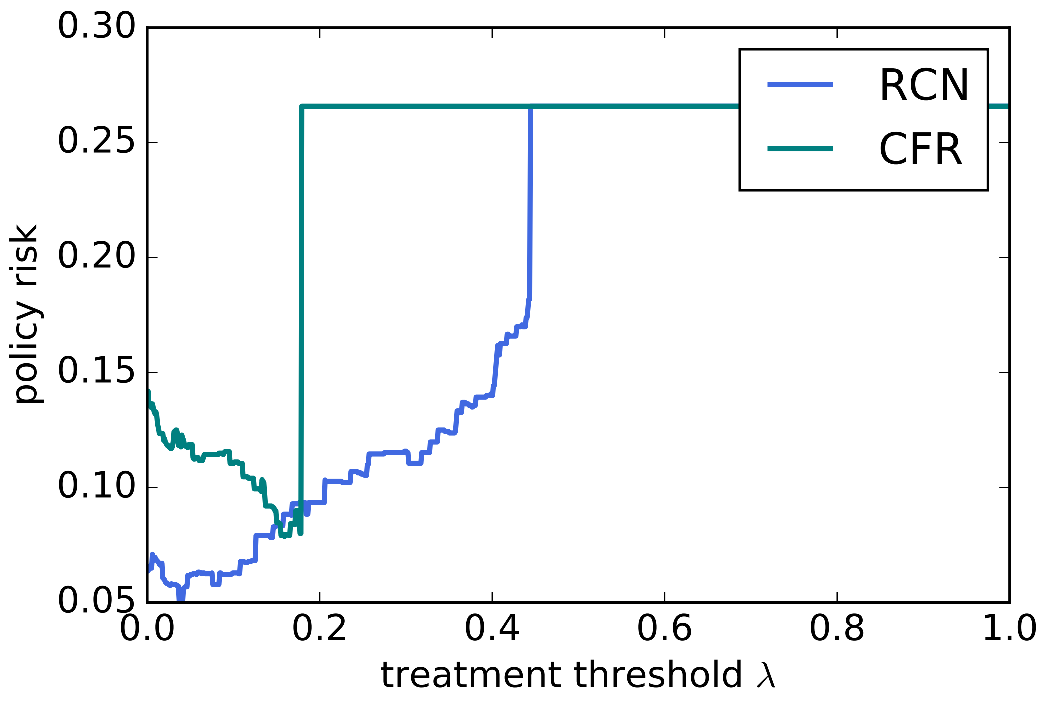

Figure 5.5: Treatment threshold versus policy risk on ASSISTments dataset. The lower policy risk is the better.

Table 5.3: Results of the ASSISTments Dataset

Model	$R_{\mathbf{POL}}$	ϵ_{ATT}
TA	0.14	-
NT	0.27	-
CFR	0.14	0.08
RCN	0.06	0.01

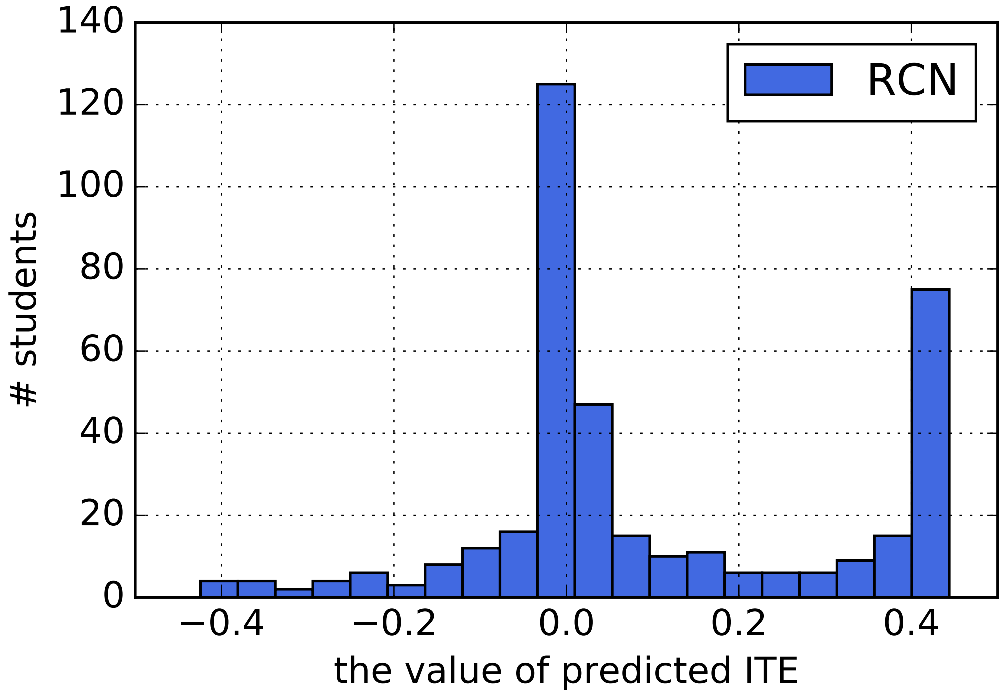

Figure 5.6: Histogram of predicted ITE from the RCN on ASSISTments dataset.

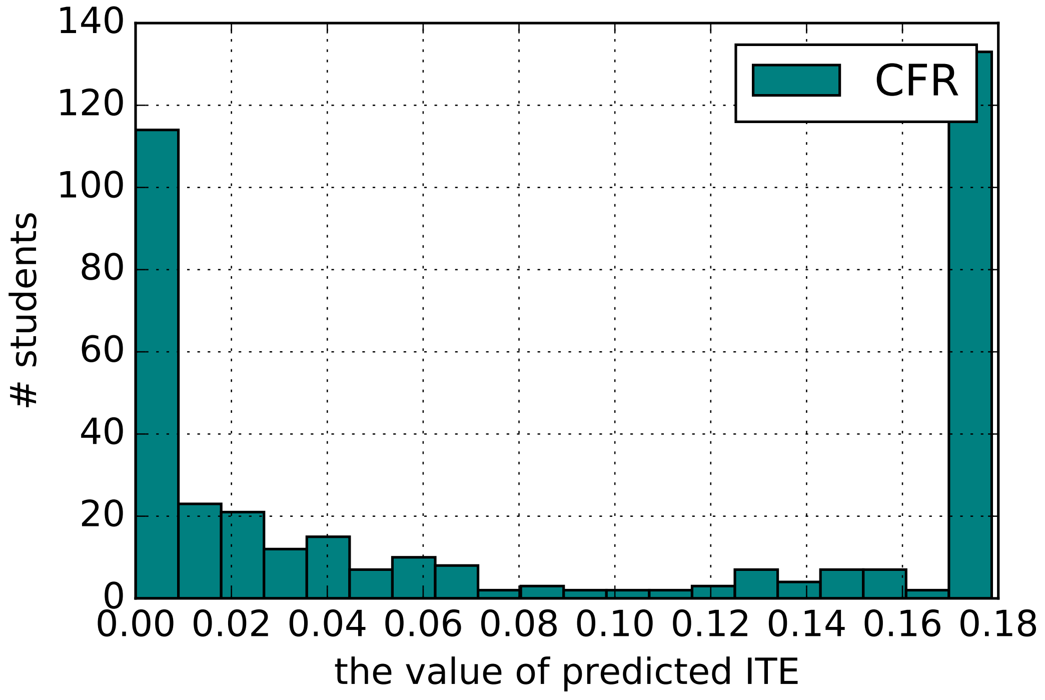

Figure 5.7: Histogram of predicted ITE from the CFR on ASSISTments dataset.

5.7 Conclusion

As online educational experiments become popular and easy to conduct, and machine learning becomes a major tool for researchers, counterfactual inference gains a lot of interest for the purpose of personalized learning. In this paper we propose the Residual Counterfactual Networks (RCN) to estimate the individual treatment effect. Because of the dissimilarity between the distributions of the control and the treated populations, the RCN uses IPMs, such as Wasserstein and MMD distance, to learn balancing deep features from the data. A residual block is adopted on the deep features to learn the individual treatment effect (ITE) so that estimation of the ITE is dependent on the deep features. We apply our model to both synthetic datasets and real-world datasets from online educational experiment, indicating that our model achieves the state-of-the-art.

One open question for the future work is how to generalize our model for the situations where there is more than one treatment in the experiment. Integral Probability Metric (IPM) can only measure the distance between two distributions. We could use pair-wised IPM if there are more than two distributions. But this would be computationally time-consuming if the number of distributions increases. Since running experiments is expensive and collecting enough data for the model to make a reliable prediction is difficult, we need a better optimization algorithm which allows us to train the model efficiently.

Chapter 6

Sequence Learning of Student Representations for Counterfactual Inference

Personalized learning considers that the causal effects of a studied learning intervention may differ for the individual student. Randomized Controlled Trials (RCTs) are golden standard to evaluate a learning intervention by randomly assigning participants to either a control condition or an experiment condition. Counterfactual inference answers "what if" questions such as, "Would this particular student master this skill had she received a different set of hints?". This helps assign a particular student to the best of the two interventions when making a decision is needed. Prior to participating the RCT, students usually have done a number of problems and their actions on each problem are logged. In other words, each student has a sequence of actions (performance sequence). We propose a pipeline to use performance sequence to improve the performance of counterfactual inference. First, student representations are learned by applying the sequence autoencoder to performance sequences.

Then, incorporate these representations into the model for counterfactual inference. Empirical results show that the representations learned from sequence autoencoder improve the performance of counterfactual inference.

6.1 Introduction

Personalized learning provides a learning intervention which satisfies a particular student's needs. Randomized Controlled Trials (RCT) are golden standard to evaluate a learning intervention by randomly assigning participants to either a control condition or an experiment condition. Making the inference about causal effects of studies interventions is a central problem. Counterfactual inference answers "What if" questions, such as "Would this particular student benefit more if the student were given the video hint instead of the text hint when the student cannot solve a problem?" (an illustrative example is displayed in Figure 6.1). Counterfactual prediction provides a way to estimate the individual treatment effects and then allows researchers to determine which learning intervention leads particular students to a better learning.

ASSISTments, an online tutoring system, is a research platform which supports running a number of RCTs. Before students join these experiments, they have done a various number of problems and their interactions with the tutor were logged. Their interactions represent their performance history. We propose to use student performance history prior to joining the experiment to learn a representation of these students. Ideally, the representation indicates student skill proficiency and the goal is to help the counterfactual model to better estimate individualized treatment effects. Currently, student's features are numeric, which measure student performance, such as student percent correctness, number of skills mastered, and are calculated directly from student performance sequence prior to the RCTs.

Autoencoder, an unsupervised neural networks model, has achieved promising results in learning a dense representation from unlabeled data and then incorporating the representation into supervised learning models, such as feed-forward neural networks (FNN) and Recurrent Neural Networks (RNN). Autoencoder is trying to learn the underlying distribution from the unlabeled data and represent a data point in a dense embedding. Note that embedding and representation are used exchangeably in this paper. [KWK$^+$16] applied Variational Autoencoder (VA) to learning an efficient feature embeddings from unlabeled student data and demonstrated that these embeddings improve the performance of two independent classification tasks in educational data mining (EDM). Their work inspires us to learn student embeddings from student performance sequence for counterfactual inference. [SMS15, DL15] introduced the sequence autoencoder (SEA) to learn a representation for sequence data (such as, language and videos). The idea of the SEA is to train an encoder RNN which reads the input sequence into a vector and then a decoder RNN to recover the input sequence from the vector. A SEA can produce a general and task-independent representation for a sequence. Integrating the representation into a classification model reduces the input dimension and makes the training of the classification model quick and stable. Student performance sequence is a sequence of their actions on problems, which represents their skill proficiency. Fitting the performance sequence of a student into SEA presumably generate a dense vector which indicates the skill proficiency of a student.

The counterfactual model used in this work is the Residual Counterfactual Network (RCN) [ZH17]. The model first uses feed-forward neural networks to learn a balancing representation of students by minimizing the distance between the distributions of the control and the treated populations, and then adopt a residual block

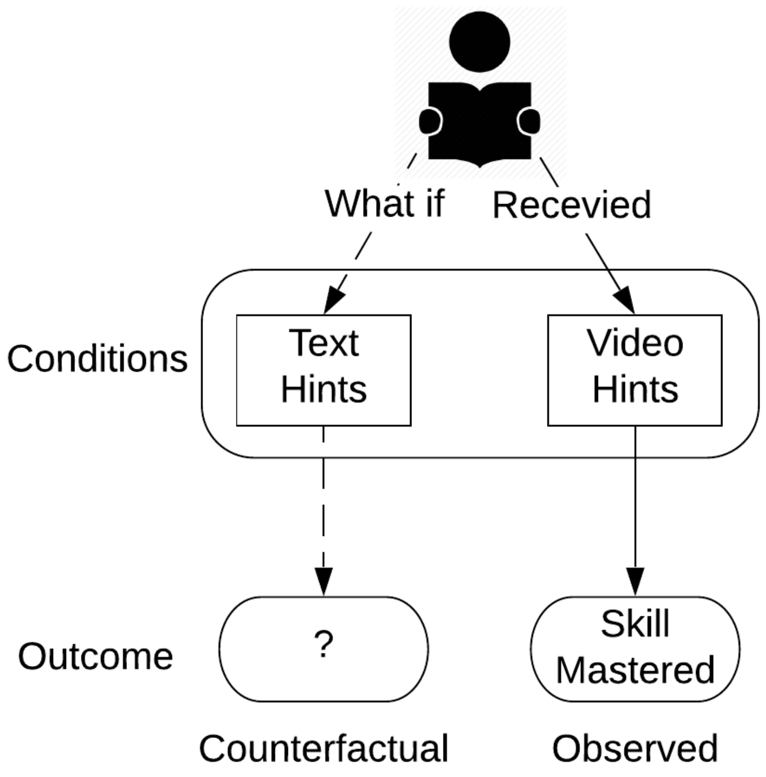

Figure 6.1: An illustrative example to demonstrate the idea behind counterfactual inference. Observed: The student received video hints and mastered the skill. Counterfactual: Would this student have mastered the skill if he had received text hints?

to estimate the individual treatment effect based on the student representation.

The goal of this work is to propose an approach to learn a student embeddings from their behavior on problems that they have worked on so that these embeddings would improve the performance of the RCN. There are an unsupervised way and a supervised way to learn student embeddings. The unsupervised way does not require labeled data and learned embeddings are task-independent. SEA is the most commonly used models for this unsupervised sequence learning. Even though the embeddings are not learned specifically for the counterfactual inference, but these embeddings are general representations of the students which would potentially be helpful for various classification tasks in EDM. The supervised way requires labeled data and the embeddings are learned in the process of solving a classification task. These embeddings are usually task-dependent and less general compared to unsupervised embeddings. We run experiments with these two ways of learning embeddings to verify whether there exists a performance gap between these two types of embeddings.

6.2 Related Work

SEA was first introduced by [SMS15] to learn video representations. Then a similar idea was proposed by [DL15] for semi-supervised sequence learning. SEA is not the only unsupervised neural networks model for sequence learning. [KWK$^+$16] applied Convolutional Neural Networks student autoencoder (CNN-SAE) to learning student embeddings from sequence data. The encoder consists of CNN layers and the decoder consists of RNN layers. The CNN-SAE uses variational autoencoder by combining Bayesian inference with neural networks [KWK$^+$16]. Recent work has shown that variational autoencoders produce better performance compared to normal autoencoders.

From a conceptual point of view, the RCN is inspired by the work on domain adaptation and deep residual learning. [LZWJ16] proposed the Residual Transfer Network that learns transferable deep features from labeled data in the source domain and unlabeled data in the target domain and adds a residual block to transfer the prediction classifier from the target domain to the source domain. The structure of the RCN is similar to that of their model. Deep residual learning is introduced by [HZRS16], the winner of the ImageNet ILSVRC 2015 challenge, to ease the training of deep networks. The residual block is designed to learn residual functions $\Delta F(\mathbf{x})$ with reference to the layer input $\mathbf{x}$. Reformulating layers to the residual block makes the training easier than directly learning the original functions $F(\mathbf{x}) = \Delta F(\mathbf{x}) + \mathbf{x}$.

The RCN extends the work by [JSS16, SJS16], where the authors built a connection between domain adaptation and counterfactual inference. They learned a representation of the data which balances the control and treated distributions. Then the treatment assignment was concatenated with the balancing representation to predict the factual outcome as while the reverse treatment assignment was concatenated with the balancing representation to predict the counterfactual outcome. Compared to their work, the RCN does not concatenate the treatment assignment, and adds a residual block to estimate the individual treatment effect based on the representation.

[WA17, AI16] proposed causal forests which is built upon the idea of random forests to estimate the heterogeneous treatment effect with semi-parametric asymptotic convergence rate.

Another research direction in this pipeline is the counterfactual model for individualized treatment effects estimation. Recent work [HLLBT17, SS17, AvdS17] have proposed various models for counterfactual inference. Especially, the Gaussian process is used in these two papers [SS17, AvdS17]. The Gaussian process GP(μ_0, k) is a non-parametric model that is fully characterized by its prior mean function $\mu_0 : \mathcal{X} \rightarrow \mathbb{R}$ and its positive-definite kernel, or covariance function, $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$.

6.3 Problem setup

The problem setup for counterfactual inference in this work is similar to the setup in [ZH17]. Let $\mathcal{T}$ be the set of proposed interventions we wish to consider, X the set of participants, and Y the set of possible outcomes. For each proposed intervention $t \in \mathcal{T}$, let $Y_t \in Y$ be the potential outcome for x when x is assigned to the intervention t. In RCT and observed study, only one outcome is observed for a given participant x; even if the participant is given an intervention and later the other, the participant is not in the same state. The model described above is also known as the Rubin-Neyman causal model [Rub05, Rub74].

We focus on a binary intervention set $\mathcal{T} = \{0, 1\}$, where intervention 1 is often referred as the "treated" and intervention 0 is the "control." In this scenario the individual treatment effects (ITE) for a participant x is represented by the quantity of $Y_1(x) - Y_0(x)$. Knowing the quantity helps assign participant x to the best of the two interventions when making a decision is needed, for example, choosing the best intervention for a specific student when the student has a trouble solving a problem. However, we cannot directly calculate ITE due to the fact that we can only observe the outcome of one of the two interventions.

When applying the counterfactual inference, we follow the common simplifying assumption of no-hidden confounding variables. This means that all the factors determining the outcome of each intervention are observed. This assumption can be formalized as the strong ignorability condition:

$$(Y_1, Y_0) \perp t|x, 0 < p(t = 1|x) < 1, \forall x.$$

Note that we cannot evaluate the validity of strong ignorability from data, and the validity must be determined by domain knowledge.

In the "treated" and the "control" setting, we refer to the observed and unobserved outcomes as the factual outcome $y^F(x)$, and the counterfactual outcome $y^{CF}(x)$ respectively. In other words, when the participant x is assigned to the "control" $(t = 0)$, $y^F(x)$ is equal to $Y_1(x)$, and $y^{CF}(x)$ is equal to $Y_0(x)$. The other way around, $y^F(x)$ is equal to $Y_0(x)$, and $y^{CF}(x)$ is equal to $Y_1(x)$.

Given n samples $\left\{(x_i, t_i, y_i^F)\right\}_{i=1}^n$, where $y_i^F = t_i \cdot Y_1(x_i) + (1-t_i)Y_0(x_i)$, a common approach for estimating the ITE is to learn a function $f : X \times T \to Y$ such that $f(x_i, t_i) \approx y_i^F$. The estimated ITE is then:

$$I\hat{T}E(x_i) = \begin{cases} y_i^F - f(x_i, 1 - t_i), & t_i = 1. \\ f(x_i, 1 - t_i) - y_i^F, & t_i = 0. \end{cases}$$

We assume n samples $\left\{(x_i, t_i, y_i^F)\right\}_{i=1}^n$ form an empirical distribution $\hat{p}^F = \{(x_i, t_i)\}_{i=1}^n$. We call this empirical distribution $\hat{p}^F \sim p^F$ the empirical factual distribution. In order to calculate ITE, we need to infer the counterfactual outcome which is dependent on the empirical distribution $\hat{p}^{CF} = \{(x_i, 1 - t_i)\}_{i=1}^n$. We call the empirical distribution $\hat{p}^{CF} \sim p^{CF}$. The p^F and p^{CF} may not be equal because the distributions of the control and the treated populations may be different. The inequality of two distributions may cause the counterfactual inference over a different distribution than the one observed from the experiment. In machine learning terms, this scenario is usually referred to as domain adaptation, where the distribution of features in test data are different than the distribution of features in training data.

6.4 Models for Sequence Learning

In this section, we propose to use the SEA to learn the representation of a student from their performance sequence. The SEA combines the idea of the RNN and the autoencoder. We first start with the autoencoder, then introduce the RNN, and finally dive into the details of the SEA.

6.4.1 Autoencoder

Autoencoder [B$^+$09] is an unsupervised neural network that is trained to read the input into a lower-dimensional vector and then reconstruct the input from the vector. Unlike supervised learning algorithms, the unsupervised learning algorithms do not require labels for the data. An autoencoder consists of two parts: the encoder and the decoder. The encoder learns the mapping from the input x to the embedding z. The decoder reconstructs the input to $\tilde{x}$ from the vector. Figure 6.2 shows the structure of an autoencoder. It has a hidden layer that maps the input x to a vector z. Since the encoder and the decoder have its own parameters respectively, an autoencoder cannot perfectly learn to reconstruct the input. The model is forced to prioritize which aspects of the input should be copied, and it often learns useful properties of the data [GBC16]. However, the autoencoder cannot be directly applied to sequence data.

6.4.2 Recurrent Neural Networks

RNNs are popular models for sequence learning task. It has a memory in the hidden layer which keeps the sequential information that the model has calculated.

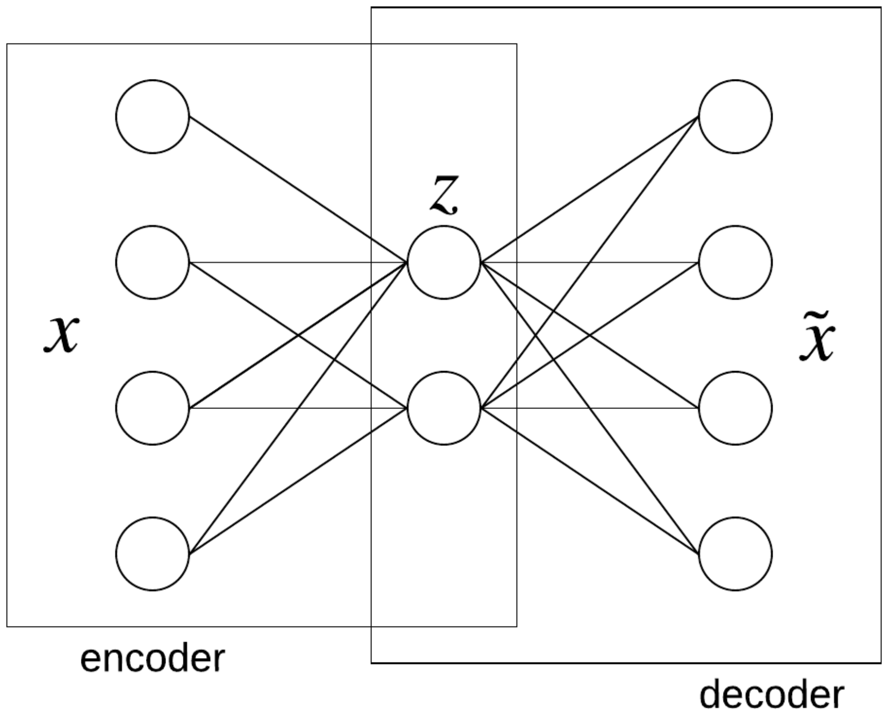

Figure 6.2: The structure of an autoencoder. An autoencoder consists of the encoder and the decoder. In this figure, the encoder maps input data from 4 dimensions to 2 dimensions with one hidden layers. The decoder reconstructs the input data from the 2-dimension code.

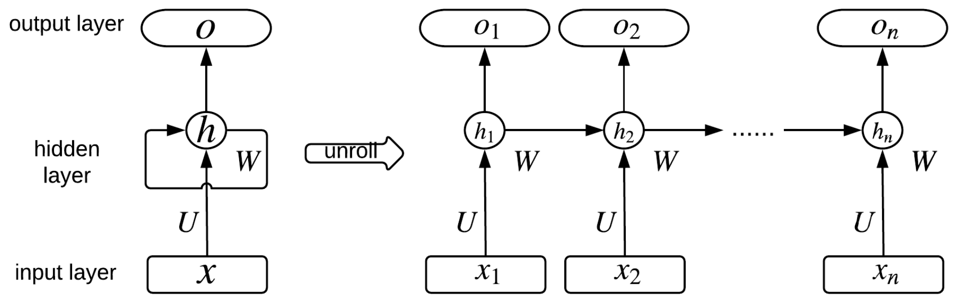

Figure 6.3: A Recurrent Neural Network model.

As shown in Figure 6.3, there is a loop in the hidden state h, which passes the information from the previous time step to the next time step. In practice, RNN suffers from the long-term dependencies. In other words, if the input sequence is long, the model cannot remember all dependencies from the beginning to the end. To better understand the RNN, equations for calculating the hidden state h of RNN are listed below.

$$h_t = \tanh(Ux_t + Wh_{t-1}) \tag{6.1}$$

Long-short Term Memory (LSTM) is a variant of RNN which is introduced to solve the long-term dependencies issue. Each LSTM hidden cell has a memory unit. The unit is shown in Figure 6.4. The memory unit learns to capture new information and forget irrelevant old information at each time step. Equations for calculating the hidden state h of LSTM are listed below.

$$i = \text{sigmoid}(U^i x_t + W^i h_{t-1}) \tag{6.2}$$

$$f = \text{sigmoid}(U^f x_t + W^f h_{t-1}) \tag{6.3}$$

$$o = \text{sigmoid}(U^o x_t + W^o h_{t-1}) \tag{6.4}$$

$$g = \tanh(U^g x_t + W^g h_{t-1}) \tag{6.5}$$

$$c_t = c_{t-1} \cdot f + g \cdot i \tag{6.6}$$

$$h_t = \tanh(c_t) \cdot o \tag{6.7}$$

Compared to a RNN, a LSTM has three extra gates: an input gate i, a forget gate f, and an output gate o. Since the sigmoid function outputs the value between 0 and 1, these gates determine how much of information passes through. g is the candidate hidden state, which is the same equation for the hidden state in vanilla RNN. c_t is the internal memory of the LSTM unit. When calculating c_t, the forget gate f controls how much of the previous internal memory passes through while the input gate i defines how much of candidate hidden state passes through. Finally, the hidden state h_t is computed by multiplying the internal memory c_t with the output gate o.

Gated Recurrent Unit (GRU) is a variant of LSTM, which combines the forget and the input gates into a single "update gate". Equations for calculating the hidden state h of GRU are listed below.

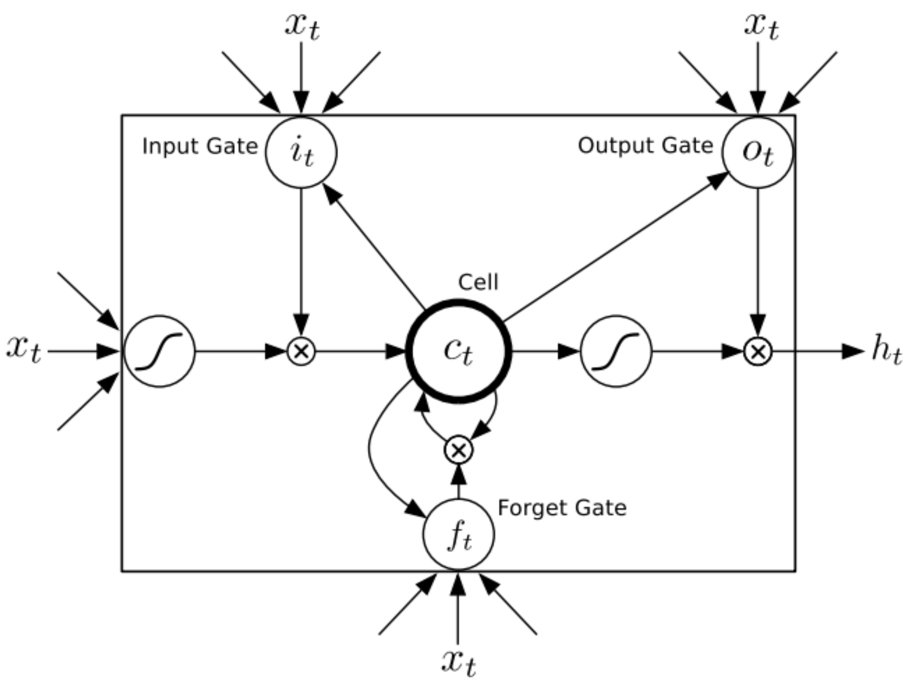

Figure 6.4: A LSTM cell. Reprint from [Gra13].

$$z = \text{sigmoid}(U^z x_t + W^z h_{t-1}) \tag{6.8}$$

$$r = \text{sigmoid}(U^r x_t + W^r h_{t-1}) \tag{6.9}$$

$$\tilde{h} = \tanh(U^h x_t + W^h(h_{t-1} \cdot r)) \tag{6.10}$$

$$h_t = z \cdot h_{t-1} + (1 - z) \cdot \tilde{h} \tag{6.11}$$

A GRU has two gates: a reset gate r and an update gate z. The reset gate determines how much of the past hidden state is kept when the current input is combined with the past hidden state. The update gate defines how much of the past hidden state is retained.

6.4.3 Sequence Autoencoder

As mentioned ahead, [SMS15, DL15] applied the SEA to sequence learning. The details of training the SEA in these two papers are slightly different, but the ideas behind these two papers are the same: encode the input sequence into a vector and then recover the exact input sequence from the vector. Figure 6.5 shows the approach described in [SMS15]. The structure of the SEA consists of two RNNs, the encoder RNN and the decoder RNN. The encoder reads in the input sequence and after the last input is read, the cell state and the output state are copied over to the decoder. The learned representation of the input sequence is the cell state at the last input. The decoder tries to reconstruct the input sequence from the learned representation. To make the decoder easier to be trained, the decoder can predict the input sequence in reverse order and take the output generated at previous step as input at current step, i.e. the dotted box indicated in Figure 6.5.

To apply the SEA to student performance sequence, the input on each time step

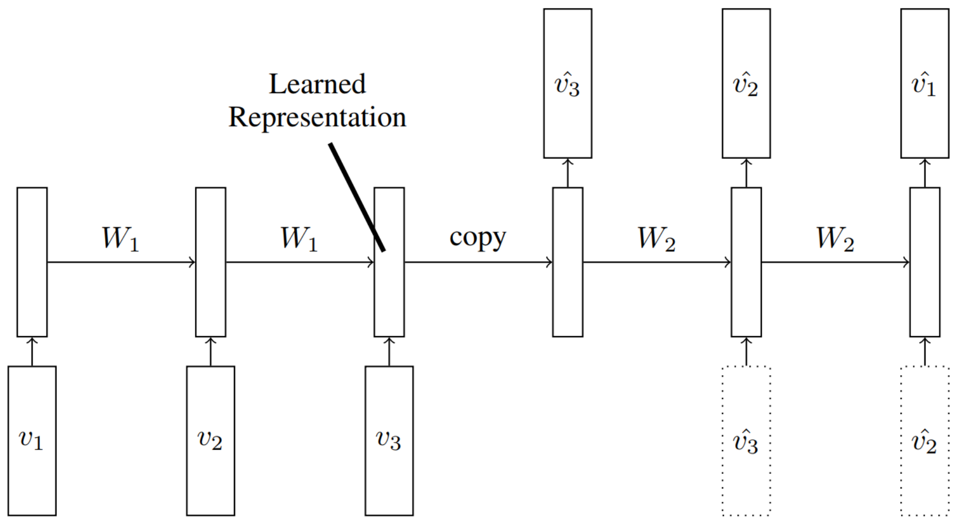

Figure 6.5: The structure of SEA. The model consists of two RNNs, an encoder RNN and a decoder RNN. The input is a sequence of vectors and the encoder reads in the input sequence. After the last input is read, the cell state and output state are copied over to the decoder. The decoder predicts the target sequence, which is the same as the input sequence. Reprint from [SMS15].

to the encoder RNN is student's logged actions on one problem (such as, correctness, number of hints requested, the time taken to solve this problem, etc.). After the encoder RNN reads all logged actions of a student, the hidden state of the current model is the student representation. The model reconstructs the input sequence from the student representation. The encoder RNN and the decoder RNN can be either the same RNN or two separate RNNs.

6.4.4 Counterfactual Model

We used the RCN model for the counterfactual inference. The RCN first learns a balancing representation of deep features $\Phi : X \to R^d$, and then learns a residual mapping Δf on the representation to estimate the ITE. The structure of the RCN is shown in Figure 6.6.

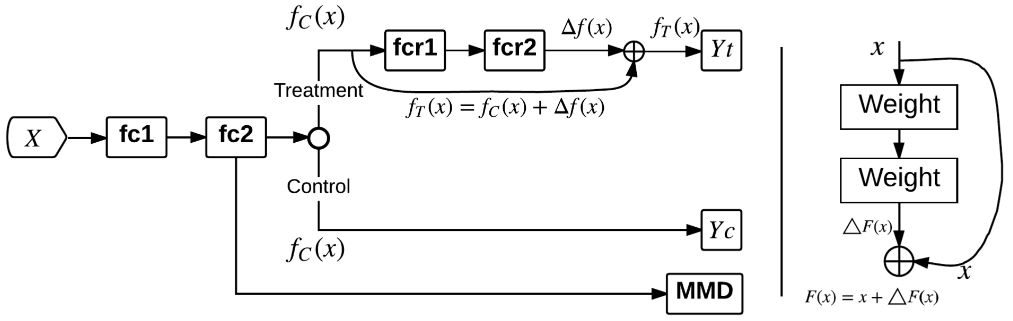

Figure 6.6: Residual Counterfactual Networks for counterfactual inference. Maximum Mean Discrepancy (MMD) is adopted on layers fc1 and fc2 to minimize the discrepancy distance of the deep features of the control and the treated populations. MMD measures the distance between control and treated in hidden layers. For the treated group, we add a residual block fcr1-fcr2 to learn treatment effects $\Delta f(x)$ so that $f_T(x) = f_C(x) + \Delta f(x)$ where $f_T(x)$ is predicted treatment outcome and $f_C(x)$ is predicted control outcome.

The RCN uses feed-forward neural networks (*fc1* and *fc2* in Figure 6.6) with ReLu activation function to learn a balancing representation of deep features Φ, where $Relu(z) = max(0, z)$. We need to generalize from factual distribution to counterfactual distribution in the feature representation Φ to obtain an accurate estimation of the counterfactual outcome. The common successful approaches for domain adaptation encourage similarity between the latent feature representations w.r.t the different distributions. This similarity is often enforced by minimizing a certain distance between the domain-specific hidden features. The distance between two distributions is usually referred to as the discrepancy distance, introduced by [MMR09], which is a hypothesis class dependent distance measure tailored for domain adaptation.

Let $\mathcal{H}$ be the reproducing kernel Hilbert space (RKHS) and $k : \Omega \times \Omega \to \mathbb{R}$ be a characteristic kernel associated with it. The RKHS $\mathcal{H}$ satisfies the reproducing property $\langle f, k(x, \cdot)\rangle_{\mathcal{H}} = f(x) \forall f \in \mathcal{H}, \forall x \in \Omega$. The kernel function $k(x, \cdot)$ implies a feature mapping $\phi(x) : \Omega \to \mathcal{H}$, such that $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$. Alternatively, $k(x, x')$ can be considered as a metric of similarity between two points $x, x' \in \Omega$. Maximum Mean Discrepancy (MMD) is a way to measure the distance between two distributions p and q in the RKHS $\mathcal{H}$. MMD between p and q is interpreted as the distance between the mean embeddings of p and q in the RKHS. The RCN uses MMD to measure the distance between the distribution of student features in the control and the distribution of student features in the treatment. The empirical estimator of MMD can be calculated with the student features from each condition using the equation below:

$$\begin{aligned}\mathrm{MMD}^2(D_c, D_t) = &\sum_{i=1}^{n_c}\sum_{j=1}^{n_c}\frac{k(x_i^c, x_j^c)}{n_c^2}\\ &+\sum_{i=1}^{n_t}\sum_{j=1}^{n_t}\frac{k(x_i^t, x_j^t)}{n_t^2}\\ &-2\sum_{i=1}^{n_c}\sum_{j=1}^{n_t}\frac{k(x_i^c, x_j^t)}{n_c n_t}\end{aligned}$$

where $k(\cdot, \cdot)$ is the kernel function in RKHS, D_c is the distribution of student features in the control, x^c is sample data from D_c, D_t is the distribution of student features in the treatment, x^t is sample data from D_t, n_c is the total number of sample data from D_c, and n_t is the total number of sample data from D_t.

With this empirical estimator, we do not need to inference the real distribution D_c and D_t in order to estimate the MMD. It can be calculated from given independent i.i.d data from both distributions. The counterfactual model minimizes the MMD as a regularizer to learn a balanced representation between the control and the treated.

A residual block [HZRS16] (*fcr1* and *fcr2* in Figure 6.6) is used to estimate the treatment effects $\Delta f(x)$, where x is balancing representations after $fc2$. $f_T(x)$ is

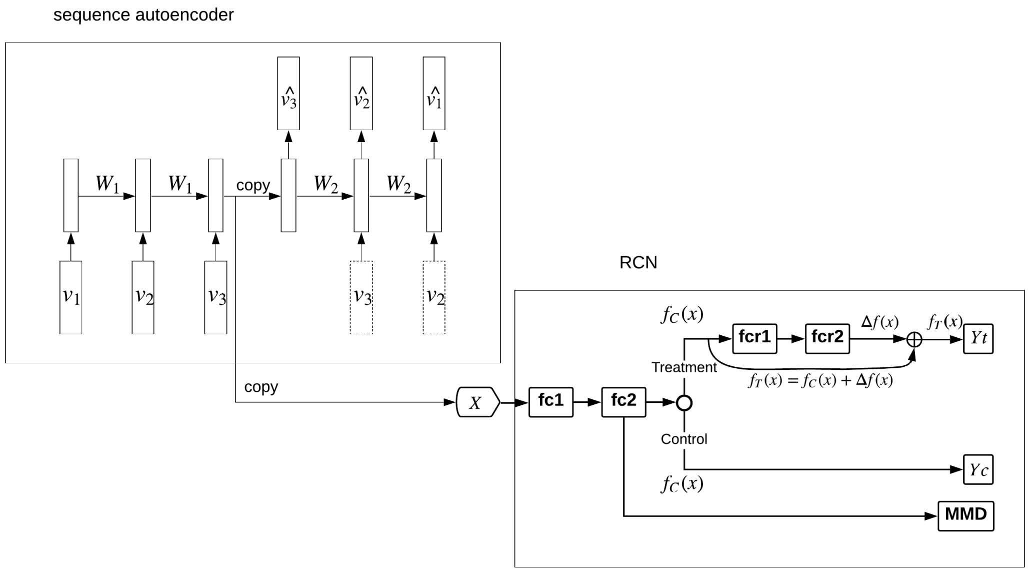

Figure 6.7: The structure of the pipeline. First, train the sequence autoencoder with problem logs. After the sequence autoencoder finishes training, the student representations are calculated. Then take the student representations as the input to the RCN.

predicted treatment outcome and $f_C(x)$ is predicted control outcome. So predicted treatment outcome can be expressed as $f_T(x) = f_C(x) + \Delta f(x)$.

6.4.5 Pipeline

In summary, there are two components in this pipeline: the SEA model and the RCN model. The structure of the pipeline is shown in Table 6.7. The data available for each student are their performance history, e.g., their actions on every problem that they have worked on prior to joining RCTs. We first fit student performance history into the SEA to learn a representation. These representations presumably indicate student skill proficiency and distinguish these students. Then we take these representations of the students in the RCT as input to the RCN and train the RCN to estimate the potential treatment outcome and control outcome. With potential

Exp 1		**Exp 2**	
# in control	198	# in control	141
# in treatment	184	# in treatment	166
CR in control	0.64	CR in control	0.88
CR in treatment	0.62	CR in treatment	0.85
p-value	0.66	p-value	0.34

Table 6.1: Statistics on two experiments. CR = completion rate.

outcomes, the ITE for each student is the difference between the potential treatment outcome and the potential control outcome. If the ITE is larger than 0, it means that the student benefits more from the treatment condition. Otherwise, it means that the student benefits more from the control condition. An accurate estimate of ITE helps the decision makers (e.g., Intelligent Tutoring System) provide a learning intervention which leads to a better learning outcome.

6.5 Datasets

6.5.1 Randomized Controlled Trials

We investigated two RCTs in our experiments. In both RCTs, students were randomly assigned to text hints or video hints. For students in text/video hints condition, they would receive text/video hints when they asked for hints. The content of text hints and the content of video hints are basically the same. Some of statistics on these two experiments are shown in Table 6.1. The results of t-test (p-value) indicate that neither of condition in two RCTs is significantly better. This makes the task more difficult for the pipeline. Because if one of the conditions is significantly better, the only thing that the pipeline needs to learn is to assign all student to that condition regardless of the contextual information of a student. For some of RCTs, even though two conditions are not significantly different, there still might exist the interaction between the condition and the contextual information of a student. The goal of the pipeline is to find the interaction (e.g., the slight difference between these two conditions) and assign students to the proper condition. The outcome measured in these two experiments is the completion rate. The average completion rate for each condition is listed in Table 6.1.

6.5.2 Problem Logs

RCTs were run in assignment types known as ”skill builders” in which students are given problems until a threshold of understanding is reached; within ASSISTments, this threshold is traditionally three consecutive correct responses. Reaching this threshold denotes sufficient performance and completion of the assignment. In addition to this experimental data, information of the students prior to condition assignment is also provided in the form of problem-level log data providing a breadth of student information at fine levels of granularity.

For each student in RCTs, we have their completeness on the problem set. Beyond that, we also have their action information on all problems that they have worked on inside the ASSIStments prior to the experiment. Action information on one problem includes the correctness of this problem, the number of hints requested by the student, the number of attempts and the time taken to make the first action on this problem. Once the student finishes a problem, one row will be inserted into the logged data. Note that students in ASSISTments cannot proceed to the next problem until they type the correct answer to the current problem. Students usually work on a number of problem sets or skill builders across a school year. So one student ends up with various length of the sequence of their action information.

The features of problem logs in all RCTs include correctness, hint count, attempt count, first response time, bottom hint and first action. The meaning of these features is listed below.

- Correctness is binary correctness as measured by the student's first action or attempt at solving the problem, where 1 means correct on first attempt and 0 means incorrect on first attempt, or asked for help.
- Hint count is the number of hints that a student asked for prior to solving the problem. Attempt count is the number of attempts that a student made prior to solving the problem.
- First response time is the time between when the problem was started and when the student made his/her first action.
- The bottom out hint is the last hint for a problem that usually gives out the correct answer to students so that they can move on to the next problem in the problem set. Bottom hint is a binary variable where 1 means the student asked for the bottom out hint and 0 means the student did not ask for the bottom out hint.
- First action is a numerical value representing the student's first action taken after the problem started where 0 means attempt and 1 means requested a hint.

A sample data of problem logs for one student is shown in Table 6.2.

6.5.3 Preprocessing

From Table 6.2, we can see that problem set id is a categorical feature. There are usually a large number of unique problem sets in the dataset for a RCT since students have worked on various problem sets in the past. One-hot encoding is a widely used approach to convert a categorical feature into a finite-length vector

Student Id	**Problem Set Id**	**Problem Id**	**Correctness**	**Attempt count**	**Hint count**	**First response time**
1	1	273719	0	2	0	20s
1	1	426035	1	1	0	10s
1	1	426037	0	3	2	16s
1	1	426038	0	1	3	30s
1	1	285171	1	1	0	21s
1	2	21054	1	1	0	8s
1	2	32154	0	2	2	12s
1	2	62104	1	1	0	14s

Table 6.2: An example of problem logs for an student. The student has worked on two problem sets (1 and 2), and solved 5 problems for problem set 1 and 3 problems for problem set 2.

Problem Set ID	**One-Hot Encoding**
1	[1, 0, 0, 0]
2	[0, 1, 0, 0]
3	[0, 0, 1, 0]
4	[0, 0, 0, 1]

Table 6.3: An illustrative example of using one-hot encoding to represent a categorical feature. In this example, there are 4 unique problem sets (1, 2, 3, 4), so the number of elements in one-hot vector is 4. Each element corresponds to one problem set.

where only one of elements is one and the rest are zeros. The length of one-hot vector is equal to the number of unique values in the categorical feature. However, if the number of unique values is too large (e.g., 5000), it is not practical to use a 5000-dimensional vector to represent this feature. We can truncate unique values to a small set by picking top n most frequent values and combining the rest of values into a single value as ’others’. In the end, this categorical feature can be represented as a $(n + 1)$-length vector. An illustrative example of using one-hot encoding to represent problem set id is shown in Table 6.3.

The values of attempt count, hint count and first response time vary according to the difficulty of the problem and the total number of hints for the problem. To measure these three features on the same scale, we normalized these three features by calculating the percentile within the problem set and then dividing the results by 100.

Afterward, for each row of problem logs, we concatenated the one-hot vector for problem set id with three normalized features (attempt count, hint count, and first response time) and two binary features (bottom hint and first action) to form the input vector.

6.6 Experiments

6.6.1 Evaluation

The RCT data are randomly split to two parts. One part is the training data, which is used to train the model. The other one is the testing data. The hyper-parameters were tuned only on training data. The model cannot access the test data during the training and its performance was tested on the test data after it was trained. This corresponds to the case where a new student requests hints and the goal is to select the best set of hints.

The benefit of the counterfactual inference is that it can assign an individual student to the best condition since it calculates the ITE. The effectiveness of treatment in RCTs has traditionally been measured by the ATE. Since ATE only estimates the average effects of treatment across the control population and the treatment population, it does not verify whether any particular student would benefit by the treatment when a decision needs to be made about whether giving the treatment to the student. For instance, in the case of text hints as the control and video hints as the treatment, ATE is close to zero. In fact, boys benefit more from text hints and girls benefit more from video hints. With ATE, we cannot find this kind of

ID	**Actual condition**	**Completion**	**Predicted outcome if treated**	**Predicted outcome if not treated**	**Predicted treatment effect**	**Recommended condition**
1	Control	0	0.8	0.75	0.05	Treatment
2	**Control**	**1**	**0.3**	**0.45**	**-0.15**	**Control**
3	**Control**	**0**	**0.5**	**0.7**	**-0.20**	**Control**
4	**Treatment**	**1**	**0.5**	**0.38**	**0.12**	**Treatment**
5	Treatment	0	0.91	0.99	-0.08	Control
6	**Treatment**	**1**	**0.87**	**0.37**	**0.50**	**Treatment**

Table 6.4: Hypothetical data for some example students. The predicted outcome is the probability that the student would complete the assignment. Students in bold are those whose randomized treatment assignment is congruent with the recommendation of the counterfactual inference model. Data from these students would be used to calculate the policy risk. In this example, the completion rate for this model is 0.75 (3/4).

preference. Counterfactual inference potentially could identify the preference with accurate estimation of ITE.

However, the "ground truth" of the ITE for each student in RCTs is unknown. We cannot gather the counterfactual outcomes from RCTs and thus do not have the ITE of each student. To accurately evaluate the counterfactual model, [VKD07] proposed a method, which first identifies all students where the treatment recommendation from the model is the same as the actual randomized assignment. Once we have students with congruent treatment recommendations, we can check whether these students are assigned to the better condition by looking at their performance (e.g., completion, test score). An illustrative example of how this evaluation method works is shown in Table 6.4.

6.6.2 Baselines

To properly evaluate the proposed pipeline, we developed some baseline models. The effectiveness of treatment in RCTs has traditionally been measured by the average treatment effects (ATE), and ATE does not verify any particular student would benefit by the treatment. In other words, traditional ATE methods try to find better condition between the control and the treatment, and then assign all students to the better one. To mimic these methods, we adopted two simplest baselines, "assign all to the control" and "assign all to the treatment". "Assign all to the control" provides the control condition to all students regardless of the contextual information, on the other hand, "assign all to the treatment" provides the treatment condition to all students.

When we previously analyzed these two RCTs, we engineered the numeric features for each student which were aggregated from problem logs on two dimensions: the student and the class that the student enrolled in. There are 16 features in total including 'prior percent correct', 'prior percent completion', 'prior class percent completion', etc. The meaning of these three features is listed below to give a brief idea of how the numeric features look like.

- Prior percent correct - the percent of past problems that the student answered correctly.
- Prior percent completion - the percent of previously completed assignments.
- Prior class percent completion - the percent of previously completed assignments of the class that the student enrolled in.

We used the RCN with these numeric features as another baseline to verify that whether the representation learned from the SEA can help the RCN assign a particular student to a more suitable condition. We referred to this baseline as "RCN with numeric features" in Table 6.5.

The effectiveness of unsupervised representation learning is another point that we need to verify in our experiments. To this end, we developed another baseline model by first feeding student performance sequence into a RNN and then taking the hidden state at the last input from RNN as the input to the RCN. Compared to the SEA, there is no decoder RNN in this baseline, so the RNN is trained differently. The parameters of the RNN and the parameters of the RCN were trained together. We refer to this baseline as 'RCN with RNN' in Table 6.5.

6.6.3 Data Collection

We collected problem logs for students in the RCT prior to participating it. This leads to various length of sequences. The average length of the sequences is around 800-900. There does exist crazy-long sequence, i.e., 8000. Due to the nature of the input of the RNN, all sequences have to be either truncated or padded to the same length. This length was determined by the value of 70th percentile of length vector of all sequences, which is referred to as max sequence length. If the length of a sequence is shorter than the max sequence length, the sequence is padded with zeros to reach the max sequence length. If the length is longer than the max sequence length, the sequence is truncated by removing time steps from the beginning.

6.6.4 Configurations of Sequence Autoencoder

Before we started to run experiments, there are several decisions to make about the configuration of the SEA. First is the type of RNN. As mentioned before, there are three types of RNN: vanilla RNN, LSTM and GRU. Second is whether the decoder of the SEA predicts the input sequence in reverse order or not. Third is whether the encoder RNN and the decoder RNN are the same RNN or not. A sanity check was conducted on the performance of these configurations by predicting whether a student who had started an assignment would finish the assignment. We randomly chose an assignment and collected problem logs for students who started the assignment. The problem logs were fed into the sequence autoencoder to learning the student representations. Then a two-layer feed-forward neural network was built to predict the completion of the assignment and it takes the learned representations as the input. We conducted the sanity check for each combination of three configurations and chose the one which has the best performance. In our experiments, the type of RNN in the sequence autoencoder was GRU, and the encoder and the decoder are exactly the same GRU. Also, the decoder predicts the input sequence in the reverse order.

Exp 1 (263052)	
Assign all to treatment	0.61
Assign all to control	0.65
RCN with numeric features	0.65
RCN with RNN	0.66
RCN with SEA	**0.71**

Exp 2 (263115)	
Assign all to treatment	0.88
Assign all to control	0.88
RCN with numeric features	0.90
RCN with RNN	0.92
RCN with SEA	**0.93**

Table 6.5: Completion rate of 4 baselines and RCN with SEA on the testing data. The higher the better.

6.7 Results

As mentioned before, all RCT data were split into two parts: the training data and the testing data. All baselines and the RCN with sequence autoencoder were trained on the training data. Once models were trained, the final results were calculated on the testing data. The results of 4 baselines and RCN with SEA on two RCTs are listed in Table 6.5. Since the outcome which is used to measure the goodness of conditions is the completion rate for both RCTs, we calculated the completion rate for all matched students found by the evaluation method mentioned in Section 6.6.1.

RCN with SEA achieved the best completion rate compared to baselines in both RCTs. The interpretation of this performance achievement is three-fold:

- Achieving better results compared to "assign all to treatment" and "assign all to control" indicates that the proposed pipeline is able to detect the interaction between conditions and the contextual information of the student.

- Achieving better results compared to RCN with numeric features indicates that the representations learned from the SEA indeed help the RCN more accurately assign individual students to the correct condition.

- RCN with RNN reaches the comparable result with RCN with SEA in Exp. 2, but worse result in Exp. 1. This indicates the performance of the RCN with RNN is not as stable as the RCN with SEA across RCTs.

Another interesting finding is that the RCN with numeric features and the RCN with RNN do not outperform the simple baseline "assign all to treatment" in Exp. 1. The goodness of student representations has an impact on whether the RCN can find the interactions. When the student is under-represented, the RCN cannot assign particular students to proper condition. Intuitively, it is not surprising that both the RCN with RNN and the RCN with SEA outperform the RCN with numeric features in two RCTs since raw problem logs contain rich information, such as the problem sets that the student has worked on and the action changes over problems, compared to numeric features aggregated from problem logs.

Incorporating the RNN into the RCN makes the model more complicated and increases the difficulty of training the model. This is one of possible reasons why the RCN with RNN is not as stable as the RCN with SEA across RCTs. Learning representations using sequence autoencoder is independent of predicting tasks and keeps the complexity of the RCN as it is.

6.7.1 Power Analysis

To verify the reliability of the results from RCN with SEA, we first conducted a series of the power analysis. After running the model on the test data, students in the test data are splitting into two groups: a group of students whose recommended condition is the same as the actual assigned condition, called the "matched group", and a group of students whose recommended condition does not match the actual assigned condition, called the "unmatched group". The purpose of comparing the matched group and the unmatched group is to verify that whether the model can reliably assign individual student to the better condition. We reported the p-value and effect size between the completion rate of the matched group and that of the unmatched group in the first part of Table 6.6 and Table 6.7 for Exp 1 and Exp 2, respectively. In Exp 1, the completion rate on the matched group (N = 103, M = 0.72, SD = 0.45) was significantly higher than that on the unmatched group (N = 114, M = 0.56, SD = 0.5), p = 0.03, effect size = 0.29. In Exp 2, the completion rate on the matched group (N=75, M = 0.93, SD = 0.25) was higher than that on the unmatched group (N=74, M = 0.84, SD = 0.37) with p = 0.06, effect size = 0.30.

We also compared the matched group with the group of students who were assigned to the better condition (either the treatment group or the control group). The treatment group had a higher completion rate in both Exp 1 and Exp 2. The p-value and effect size between the matched group and the treatment group were reported in the second part of Table 6.6 and Table 6.7, respectively. In Exp 1, the completion rate on the matched group (N=75, M = 0.93, SD=0.25) was higher than that on the treatment group (N=79, M = 0.88, SD=0.32) with p = 0.31 and effect size = 0.16. In Exp 2, the completion rate on the matched group (N=75, M = 0.93, SD=0.25) was higher than that on the treatment group (N=79, M = 0.88, SD=0.32) with p = 0.31 and effect size = 0.16.

Exp 1

	n	**mean**	**std**	**p-value**	**effect size**
Matched	101	0.71	0.45	0.01	0.36
Unmatched	96	0.54	0.5		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	101 (38/63)	0.71	0.45	0.37	0.13
Control	92	0.65	0.48		

Table 6.6: Power analysis of Exp 1 for the RCN with SEA. The first part of the table indicates that the completion rate on the matched group (N=103, M = 0.72, SD=0.45) was significantly higher than that on the unmatched group (N=114, M = 0.56, SD=0.5) with p = 0.03 and effect size = 0.29. The second part of the table indicates that the completion rate on the matched group (N=103, M = 0.72, SD=0.45) was higher than that on the treatment group (N=105, M = 0.66, SD=0.48) with p = 0.34 and effect size = 0.13. In the matched group, 38 students is in the control and 63 students in the treatment.

Exp 2

	n	**mean**	**std**	**p-value**	**effect size**
Matched	75	0.93	0.25	0.06	0.30
Unmatched	74	0.84	0.37		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	75 (42/33)	0.93	0.25	0.31	0.16
Treatment	79	0.88	0.32		

Table 6.7: Power analysis of Exp 2 for the RCN with SEA. The first part of the table indicates that the completion rate on the matched group (N=75, M = 0.93, SD = 0.25) was higher than that on the unmatched group (N=74, M = 0.84, SD = 0.37) with p = 0.06, effect size = 0.30. The second part of the table indicates that the completion rate on the matched group (N=75, M = 0.93, SD=0.25) was higher than that on the treatment group (N=79, M = 0.88, SD=0.32) with p = 0.31 and effect size = 0.16. In the matched group, 42 students is in the control and 33 students in the treatment.

6.8 Discussion

Using counterfactual inference for analyzing RCTs allows researchers to calculate ITE so particular students can be assigned to a learning condition that leads to better learning. The RCN model is designed for this purpose and the empirical results reveal that the RCN requires effective student representations to better detect the interaction between conditions and the student's contextual information. Sometimes, the numeric features aggregated from problem logs are not sufficient for the RCN to learn something useful. Aggregating problem logs loses information about the problem sets that a student has worked on and action changes over problems. To alleviate this downside, we proposed to use sequence autoencoder to learn student representations from problem logs. Empirical results indicate that the SEA can produce effective student representations which help the RCN reach better performance.

The SEA is an unsupervised learning algorithm, thus it does not require labelled information from the data. Besides problem logs, logged data on assignment level and on action level can also be used for student representation learning. The representations learned from the SEA are task-independent and can be applied to various predicting tasks in EDM. In our experiment setup, we only used problem logs from students who participated the RCT. To learn a more general representation, we can sample some of students who were not in the RCT and mix these students with students in the RCT.

Integrating a RNN into RCN (RCN with RNN) is more intuitive and direct approach compared to the unsupervised SEA. However, this approach increases the difficulty of training a RCN and a RNN together. The performance of RCN with RNN is not as stable as that of RCN with SEA.

Not all RCTs have the interactions between the conditions and student's contextual information that the RCN can detect. It is obvious that this type of interaction does not exist when one of conditions is significantly better than the other one. Empirically speaking, this type of interaction exists when both conditions are slightly different and certain types of students have some preferences on one of conditions. Both RCTs in our experiments do not have a small p-value, so p-value is an indicator of the interaction to some extent. Surveys of student's preferences on some conditions might serve as another indicator.

6.9 Conclusions

To make use of problem logs, we proposed to learn student representations with the SEA. The empirical results illustrate that the representations learned from the SEA improve the performance of the RCN so particular students can be assigned to appropriate condition. The comparison between the RCN with RNN and the RCN with SEA indicates that the performance of the unsupervised way of learning representations is more stable. Representations learned from the SEA are task-independent and potentially can be applied to other predicting tasks in EDM.

Appendix A

More Experiments for RCN with SEA

A.1 Results

I included results on more RCTs in Table A.1 - Table A.10.

A.2 Results Analysis

In total, we run the RCN with SEA on 12 RCTs. The model achieved better performance than the better of the control and the treatment on 5 out of 12 RCTs (PS263052, PS263115, PS246647, PS246482, PS241622). The model prescribed the better of the control and the treatment on 2 out of 12 RCTs (PS237447, PS259379). The model performed worse than the better of two conditions on 2 out of 12 RCTs (PS226210, PS246627). The model achieved similar results with the better of two conditions on 3 out of 12 RCTs (PS303899, PS243393, PS255116).

We also reported effect size from using RCN across all 12 RCTs in Table A.11.

PS: 246627	
completion rate on test data	
Assign all to treatment	0.68
Assign all to control	0.69
Model with numeric features	0.61
Model with autoencoder	0.68

	n	**mean**	**std**	**p-value**	**effect size**
Treatment	120	0.68	0.47	0.87	-0.02
Control	140	0.69	0.46		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	120 (0/120)	0.68	0.47	0.87	-0.02
Unmatched	140	0.69	0.46		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	120	0.68	0.47	0.87	-0.02
Control	140	0.69	0.46		

Table A.1: Results on the problem set 246627. In the matched group, none of students is in the control and 120 students in the treatment.

PS: 237447	
completion rate on test data	
Assign all to treatment	0.95
Assign all to control	0.97
Model with numeric features	0.97
Model with autoencoder	0.97

	n	**mean**	**std**	**p-value**	**effect size**
Treatment	194	0.95	0.21	0.30	-0.11
Control	158	0.97	0.16		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	158 (158/0)	0.97	0.16	0.30	0.11
Unmatched	194	0.95	0.21		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	158	0.97	0.16	1	0
Control	158	0.97	0.16		

Table A.2: Results on the problem set 237447. In the matched group, all students is in the control and none in the treatment.

PS: 255116	
completion rate on test data	
Assign all to treatment	0.79
Assign all to control	0.80
Model with numeric features	0.79
Model with autoencoder	0.80

	n	**mean**	**std**	**p-value**	**effect size**
Treatment	131	0.79	0.41	0.81	-0.03
Control	119	0.80	0.40		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	136 (40/96)	0.80	0.40	0.69	0.05
Unmatched	114	0.78	0.41		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	136	0.80	0.40	0.95	0.01
Control	119	0.80	0.40		

Table A.3: Results on the problem set 255116. In the matched group, 40 students is in the control and 96 students in the treatment.

PS: 246647	
completion rate on test data	
Assign all to treatment	0.83
Assign all to control	0.83
Model with numeric features	0.87
Model with autoencoder	0.85

	n	**mean**	**std**	**p-value**	**effect size**
Treatment	138	0.83	0.38	0.99	0
Control	132	0.82	0.38		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	138 (20/118)	0.85	0.36	0.33	0.18
Unmatched	132	0.80	0.40		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	138	0.85	0.36	0.63	0.06
Treatment	138	0.83	0.38		

Table A.4: Results on the problem set 246647. Even though Model with numeric features achieved best performance, the reliable test was conducted on Model with autoencoder. In the matched group, 20 students is in the control and 118 students in the treatment.

PS: 246482	
completion rate on test data	
Assign all to treatment	0.73
Assign all to control	0.71
Model with numeric features	0.73
Model with autoencoder	0.76

	n	**mean**	**std**	**p-value**	**effect size**
Treatment	124	0.73	0.44	0.66	0.06
Control	106	0.71	0.46		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	114 (43/71)	0.76	0.43	0.17	0.18
Unmatched	116	0.68	0.47		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	114	0.76	0.43	0.35	0.12
Treatment	124	0.73	0.44		

Table A.5: Results on the problem set 246482. In the matched group, 43 students is in the control and 71 students in the treatment.

PS: 243393	
completion rate on test data	
Assign all to treatment	0.69
Assign all to control	0.72
Model with numeric features	0.69
Model with autoencoder	0.72

	n	**mean**	**std**	**p-value**	**effect size**
Treatment	454	0.69	0.46	0.30	-0.07
Control	479	0.72	0.45		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	479 (380/99)	0.72	0.45	0.46	0.05
Unmatched	454	0.69	0.46		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	479	0.72	0.45	0.89	-0.01
Control	479	0.72	0.45		

Table A.6: Results on the problem set 243393. In the matched group, 380 students is in the control and 99 students in the treatment.

PS: 241622	
completion rate on test data	
Assign all to treatment	0.82
Assign all to control	0.86
RCN with numeric features	0.89
RCN with SEA	0.84

	n	**mean**	**std**	**p-value**	**effect size**
Treatment	152	0.82	0.38	0.30	-0.12
Control	163	0.86	0.34		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	151 (41/110)	0.89	0.32	0.11	0.18
Unmatched	152	0.82	0.38		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	151	0.89	0.32	0.55	0.07
Control	163	0.87	0.34		

Table A.7: Results on the problem set 241622. Since RCN with numeric features achieved the best performance and RCN with SEA was worse than Assign all to control, the reliable test was conducted on results from RCN with numeric features. In the matched group, 41 students is in the control and 110 students in the treatment.

PS: 303899	
completion rate on test data	
Assign all to treatment	0.9
Assign all to control	0.86
RCN with numeric features	0.87
RCN with SEA	0.9

	n	**mean**	**std**	**p-value**	**effect size**
Treatment	210	0.92	0.27	0.64	0.04
Control	235	0.91	0.29		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	213 (195/18)	0.92	0.28	0.82	0.02
Unmatched	232	0.91	0.29		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	213	0.92	0.28	0.74	0.03
Treatment	210	0.92	0.27		

Table A.8: Results on the problem set 303899. In the matched group, 195 students is in the control and 18 students in the treatment.

PS: 226210	
completion rate on test data	
Assign all to treatment	0.56
Assign all to control	0.64
RCN with numeric features	0.6
RCN with SEA	0.61

	n	**mean**	**std**	**p-value**	**effect size**
Treatment	177	0.56	0.50	0.09	-0.17
Control	210	0.64	0.48		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	207 (158/49)	0.61	0.49	0.7	0.04
Unmatched	180	0.59	0.49		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	207	0.61	0.49	0.54	-0.06
Control	210	0.64	0.48		

Table A.9: Results on the problem set 226210. In the matched group, 158 students is in the control and 49 students in the treatment.

PS: 259379	
completion rate on test data	
Assign all to treatment	0.52
Assign all to control	0.4
RCN with numeric features	0.39
RCN with SEA	0.52

	n	**mean**	**std**	**p-value**	**effect size**
Treatment	75	0.52	0.50	0.14	0.24
Control	75	0.4	0.49		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	75 (0/75)	0.52	0.50	0.14	0.24
Unmatched	75	0.4	0.49		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	75	0.52	0.50	1	0
Treatment	75	0.52	0.50		

Table A.10: Results on the problem set 259379. In the matched group, none of students is in the control and 75 students in the treatment.

All 12 RCTs	
completion rate on test data	
Assign all to treatment	0.75
Assign all to control	0.76
RCN with SEA	0.77

	n	**mean**	**std**	**p-value**	**effect size**
Treatment	1959	0.75	0.43	0.39	-0.03
Control	1979	0.76	0.42		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	1954 (874/1080)	0.77	0.42	0.08	0.06
Unmatched	1984	0.75	0.43		

	n	**mean**	**std**	**p-value**	**effect size**
Matched	1954	0.77	0.42	0.64	0.01
Control	1979	0.76	0.42		

Table A.11: Results across all 12 RCTs. In the matched group, 874 students were in the control and 1080 in the treatment.

Bibliography

[AI16] Susan Athey and Guido Imbens. Recursive partitioning for heterogeneous causal effects. *Proceedings of the National Academy of Sciences*, 113(27):7353–7360, 5 July 2016.

[AvdS17] Ahmed M Alaa and Mihaela van der Schaar. Bayesian inference of individualized treatment effects using multi-task gaussian processes. *arXiv preprint arXiv:1704.02801*, 2017.

[B$^+$09] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[BBJV14] Michael Brooks, Sumit Basu, Charles Jacobs, and Lucy Vanderwende. Divide and correct: using clusters to grade short answers at scale. In *L@S*, 2014.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. 1 September 2014.

[BCV13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: a review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, August 2013.

[C$^+$15] Open Science Collaboration et al. Estimating the reproducibility of psychological science. *Science*, 349(6251):aac4716, 2015.

[CA94] Albert T Corbett and John R Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, 4(4):253–278, 1994.

[CBS$^+$16a] Edward Choi, Mohammad Taha Bahadori, Andy Schuetz, Walter F Stewart, and Jimeng Sun. Doctor ai: Predicting clinical events via recurrent neural networks. In *Machine Learning for Healthcare Conference*, pages 301–318, 2016.

[CBS$^+$16b] Edward Choi, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter Stewart. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. In *Advances in Neural Information Processing Systems*, pages 3504–3512, 2016.

[CCKP16] Billy Chiu, Gamal K. O. Crichton, Anna Korhonen, and Sampo Pyysalo. How to train good word embeddings for biomedical nlp. In *BioNLP@ACL*, 2016.

[CCW$^+$16] Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-over-attention neural networks for reading comprehension. *CoRR*, abs/1607.04423, 2016.

[CH13] Hongbo Chen and Ben He. Automated essay scoring by maximizing Human-Machine agreement. In *EMNLP*, 2013.

[CKH$^+$16] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 7–10. ACM, 2016.

[CKJ06] Hao Cen, Kenneth Koedinger, and Brian Junker. Learning factors analysis-a general method for cognitive model evaluation and improvement. In *Intelligent tutoring systems*, volume 4053, pages 164–175. Springer, 2006.

[CvMG$^+$14] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN Encoder-Decoder for statistical machine translation. 3 June 2014.

[DL15] Andrew M. Dai and Quoc V. Le. Semi-supervised sequence learning. In *NIPS*, 2015.

[DLCS16] Bhuwan Dhingra, Hanxiao Liu, William W. Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. *CoRR*, abs/1606.01549, 2016.

[dSG14] Cícero Nogueira dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*, pages 69–78, 2014.

[DZWH13] Hien Duong, Linglong Zhu, Yutao Wang, and Neil Heffernan. A prediction model that uses the sequence of attempts and hints to better predict knowledge:" better to attempt the problem first, rather than ask for a hint". In *Educational Data Mining 2013*, 2013.

[FW65] G E Forsythe and N Wirth. Automatic grading programs. commun. *Commun. ACM 8*, 5:275–278, 1965.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[GBH10] Yue Gong, Joseph E Beck, and Neil T Heffernan. Comparing knowledge tracing and performance factor analysis by using multiple model fitting procedures. In *International conference on intelligent tutoring systems*, pages 35–44. Springer, 2010.

[Gra13] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[GWD14a] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[GWD14b] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. 20 October 2014.

[GZF16] Chase Geigle, Chengxiang Zhai, and Duncan C Ferguson. An exploration of automated grading of complex assignments. In *L@S*, 2016.

[HBM00] Andrew Heathcote, Scott Brown, and DJK Mewhort. The power law repealed: The case for an exponential law of practice. *Psychonomic bulletin & review*, 7(2):185–207, 2000.

[Hel07] Michael T Helmick. Interface-based programming assignments and automatic grading of java programs. In *ITiCSE*, 2007.

[Hil11] Jennifer L Hill. Bayesian nonparametric modeling for causal inference. *J. Comput. Graph. Stat.*, 20(1):217–240, 2011.

[HKG$^+$15] Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *NIPS*, 2015.

[HLLBT17] Jason Hartford, Greg Lewis, Kevin Leyton-Brown, and Matt Taddy. Deep IV: A flexible approach for counterfactual prediction. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1414–1423, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[HS97] S Hochreiter and J Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 15 November 1997.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[JPS+16] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3, 2016.

[JSS16] Fredrik Johansson, Uri Shalit, and David Sontag. Learning representations for counterfactual inference. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 3020–3029, 2016.

[K+95] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Stanford, CA, 1995.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[KIO+16] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *ICML*, 2016.

[KLM16] Mohammad Khajah, Robert V. Lindsey, and Michael C. Mozer. How deep is knowledge tracing? In *EDM*, 2016.

[KWK+16] Severin Klingler, Rafael Wampfler, Tanja Käser, Barbara Solenthaler, and Markus Gross. Efficient feature embeddings for student classification with variational auto-encoders. In *Educational Data Mining*, 2016.

[Lar98] Leah S Larkey. Automatic essay grading using text categorization techniques. In *SIGIR*, 1998.

[LKEW15a] Zachary Chase Lipton, David C. Kale, Charles Elkan, and Randall C. Wetzel. Learning to diagnose with lstm recurrent neural networks. *CoRR*, abs/1511.03677, 2015.

[LKEW15b] Zachary Chase Lipton, David C. Kale, Charles Elkan, and Randall C. Wetzel. Learning to diagnose with LSTM recurrent neural networks. *CoRR*, abs/1511.03677, 2015.

[LZWJ16] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Unsupervised domain adaptation with residual transfer networks. In *Advances in Neural Information Processing Systems 29*, pages 136–144. Curran Associates, Inc., 2016.

[MAP$^+$15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale machine learning on heterogeneous systems, 2015.

[MFD$^+$16] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-Value memory networks for directly reading documents. *CoRR*, abs/1606.03126, 2016.

[MH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[MM09] Michael Mohler and Rada Mihalcea. Text-to-Text semantic similarity for automatic short answer grading. In *EACL*, 2009.

[MMR09] Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. Domain adaptation: Learning bounds and algorithms. In *COLT*, 2009.

[MPRo13] P Mitros, V Paruchuri, J Rogosic, and others. An integrated framework for the grading of freeform responses. *The Sixth Conference of*, 2013.

[MSC$^+$13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C J C Burges, L Bottou, M Welling, Z Ghahramani, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[NPHG14] Andy Nguyen, Chris Piech, Jonathan Huang, and Leonidas J Guibas. Codewebs: scalable homework search for massive open online programming courses. In *WWW*, 2014.

[OH15] Korinn S Ostrow and Neil T Heffernan. The role of student choice within adaptive tutoring. In *Artificial Intelligence in Education*, pages 752–755. Springer, Cham, 21 June 2015.

[PBH$^+$15] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. Deep knowledge tracing. In *Advances in Neural Information Processing Systems*, pages 505–513, 2015.

[PCN15] Peter Phandi, Kian Ming Adam Chai, and Hwee Tou Ng. Flexible domain adaptation for automated essay scoring using correlated linear regression. In *EMNLP*, 2015.

[Pea09] Judea Pearl. Causal inference in statistics: An overview. *Stat. Surv.*, 3(0):96–146, 2009.

[PH11] Zachary Pardos and Neil Heffernan. Kt-idem: introducing item difficulty to the knowledge tracing model. *User Modeling, Adaption and Personalization*, pages 243–254, 2011.

[PHN$^+$15] Chris Piech, Jonathan Huang, Andy Nguyen, Mike Phulsuksombati, Mehran Sahami, and Leonidas J Guibas. Learning program embeddings to propagate feedback on student code. *CoRR*, abs/1505.05969, 2015.

[PJCK09] Philip I Pavlik Jr, Hao Cen, and Kenneth R Koedinger. Performance factors analysis–a new alternative to knowledge tracing. *Online Submission*, 2009.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[RFMM16] Jeremy Roschelle, Mingyu Feng, Robert F Murphy, and Craig A Mason. Online mathematics homework increases student achievement. *AERA Open*, 1 October 2016.

[RH06] Leena Razzaq and Neil T Heffernan. Scaffolding vs. hints in the assistment system. In *Intelligent Tutoring Systems*, pages 635–644. Springer, Berlin, Heidelberg, 26 June 2006.

[RL02] Lawrence M Rudner and Tahung Liang. Automated essay scoring using bayes' theorem. *The Journal of Technology, Learning and Assessment*, 1(2), 1 June 2002.

[Rub74] Donald B Rubin. Estimating causal effects of treatments in randomized and nonrandomized studies. *J. Educ. Psychol.*, 66(5):688, October 1974.

[Rub05] D B Rubin. Causal inference using potential outcomes: Design, modeling, decisions. *J. Am. Stat. Assoc.*, 2005.

[SBB16] Alessandro Sordoni, Phillip Bachman, and Yoshua Bengio. Iterative alternating neural attention for machine reading. *CoRR*, abs/1606.02245, 2016.

[SFG$^+$09] Bharath K Sriperumbudur, Kenji Fukumizu, Arthur Gretton, Bernhard Schölkopf, and Gert RG Lanckriet. On integral probability metrics,\phi-divergences and binary classification. *arXiv preprint arXiv:0901.2698*, 2009.

[SFG$^+$12] Bharath K Sriperumbudur, Kenji Fukumizu, Arthur Gretton, Bernhard Schölkopf, Gert RG Lanckriet, et al. On the empirical estimation of integral probability metrics. *Electronic Journal of Statistics*, 6:1550–1599, 2012.

[SHK$^+$14] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[SJS16] Uri Shalit, Fredrik Johansson, and David Sontag. Estimating individual treatment effect: generalization bounds and algorithms. *arXiv preprint arXiv:1606.03976*, 2016.

[SMS15] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. In *ICML*, 2015.

[SPH16] Douglas Selent, Thanaporn Patikorn, and Neil Heffernan. ASSISTments dataset from multiple randomized controlled experiments. In *Proceedings of the Third (2016) ACM Conference on Learning @ Scale*, L@S '16, pages 181–184, New York, NY, USA, 2016. ACM.

[SS17] Peter Schulam and Suchi Saria. What-if reasoning with counterfactual gaussian processes. *CoRR*, abs/1703.10651, 2017.

[SSWF15] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-To-End memory networks. In C Cortes, N D Lawrence, D D Lee, M Sugiyama, and R Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2440–2448. Curran Associates, Inc., 2015.

[TN16] Kaveh Taghipour and Hwee Tou Ng. A neural approach to automated essay scoring. In *EMNLP*, 2016.

[Tre11] Lyndal Trevena. Wikiproject medicine, 2011.

[TZS$^+$16] Makarand Tapaswi, Yukun Zhu, Rainer Stiefelhagen, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Movieqa: Understanding stories in movies through question-answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4631–4640, 2016.

[VIAWH15] Eric G Van Inwegen, Seth A Adjei, Yan Wang, and Neil T Heffernan. Using partial credit and response history to model user knowledge. In *Educational Data Mining*, 2015.

[VKD07] Andrew J Vickers, Michael W Kattan, and Sargent Daniel. Method for evaluating prediction models that apply the results of randomized trials to individual patients. *Trials*, 8(1):14, 5 June 2007.

[WA17] Stefan Wager and Susan Athey. Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 2017.

[WBC+15] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

[WCB14] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014.

[WKHE16] Kevin H. Wilson, Yan Karklin, Bojian Han, and Chaitanya Ekanadham. Back to the basics: Bayesian extensions of irt outperform neural networks for proficiency estimation. In *EDM*, 2016.

[XZIB16] Xiaolu Xiong, Siyuan Zhao, Eric Van Inwegen, and Joseph Beck. Going deeper with deep knowledge tracing. In *Proceedings of the 9th International Conference on Educational Data Mining, EDM 2016, Raleigh, North Carolina, USA, June 29 - July 2, 2016*, pages 545–550, 2016.

[YBM11] Helen Yannakoudakis, Ted Briscoe, and Ben Medlock. A new dataset and method for automatically grading ESOL texts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 180–189, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[YLH+10] Hsiang-Fu Yu, Hung-Yi Lo, Hsun-Ping Hsieh, Jing-Kai Lou, Todd G McKenzie, Jung-Wei Chou, Po-Han Chung, Chia-Hua Ho, Chun-Fu Chang, Yin-Hsuan Wei, et al. Feature engineering and classifier ensemble for kdd cup 2010. In *KDD Cup*, 2010.

[ZH17] Siyuan Zhao and Neil Heffernan. Estimating individual treatment effect from educational studies with residual counterfactual networks. In *Proceedings of the 10th International Conference on Educational Data Mining, EDM 2017, Wuhan, China, June 25 - June 28, 2017*, 2017.

[ZSCM13] Will Y Zou, Richard Socher, Daniel M Cer, and Christopher D Manning. Bilingual word embeddings for Phrase-Based machine translation. In *EMNLP*, 2013.