Fall 12-19-2020

Deep Neural Network Based Student Response Modeling With Uncertainty, Multimodality and Attention

Xinyi Ding
Southern Methodist University, xding@smu.edu

DEEP NEURAL NETWORK BASED STUDENT RESPONSE MODELING WITH UNCERTAINTY, MULTIMODALITY AND ATTENTION

Approved by:

Dr. Eric C. Larson
Associate Professor of Computer Science

Dr. Liguo Huang
Associate Professor of Computer Science

Dr. King Ip Lin
Associate Professor of Computer Science

Dr. Corey Clark
Assistant Professor of Computer Science

Dr. Eli V. Olinick
Associate Professor of Engineering

Management, Information, and Systems

DEEP NEURAL NETWORK BASED STUDENT RESPONSE MODELING WITH UNCERTAINTY, MULTIMODALITY AND ATTENTION

A Dissertation Presented to the Graduate Faculty of the

Lyle School of Engineering

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Doctor of Philosophy

with a

Major in Computer Science

by

Xinyi Ding

B.E., Computer Science, ZheJiang GongShang University, China
M.S., Computer Science, University of Texas at Dallas, USA

Dec 19, 2020

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor professor Eric C. Larson for his dedicated support and guidance. Eric has provided countless insightful discussions about research. He gave me great freedom and encouraged me to explore different research fields. Besides my advisor, I want to thank professor Eli V. Olinick, who helped me in different phases of my research career. I also want to thank my other committee members for their assistance and support: professor Liguo Huang, professor Corey Clark and professor King Ip Lin. During my study in SMU, Dr. Chatchai Wangwiwattana also gave me great help not just in research. He set up a great model of being a disciplined person. Last but not the least, I want to thank my parents. Although they are thousands of miles away when I am writing these words, they never stop loving me and are always being supportive.

Ding, Xinyi B.E., Computer Science, ZheJiang GongShang University, China
M.S., Computer Science, University of Texas at Dallas, USA

Deep Neural Network Based Student Response Modeling
With Uncertainty, Multimodality and Attention

Advisor: Dr. Eric C. Larson
Doctor of Philosophy degree conferred Dec 19, 2020
Dissertation completed Sep 4, 2020

In this thesis, I investigate deep neural network based student response modeling, more specifically Knowledge Tracing (KT). Knowledge Tracing allows Intelligent Tutoring Systems to infer which topics or skills a student has mastered, thus adjusting curriculum accordingly. Deep neural network based knowledge tracing models like Deep Knowledge Tracing (DKT) and Dynamic Key-Value Memory Network (DKVMN) have achieved significant improvements compared with conventional probabilistic models. There are mainly two goals in this thesis: 1) To have a better understanding of existing deep neural network based models and their predictions through visualization and through incorporating uncertainties. 2) To improve the performance of student response modeling with multimodality and attention mechanisms. In this thesis, I will first introduce the background and show why deep neural network based knowledge tracing models might have less depth than anticipated through visualization. Next, I propose a more practical way of alleviating the concerns of these deep models by incorporating uncertainty for each prediction. Then, I will discuss how adding more modalities and attention mechanisms might help improve model performance.

TABLE OF CONTENTS

LIST OF FIGURES ix

LIST OF TABLES xii

CHAPTER

1. INTRODUCTION 1

1.1. Knowledge Tracing 4

1.1.1. Bayesian Knowledge Tracing 6

1.1.2. Bayesian Networks for Knowledge Tracing 8

1.2. Learning Factors Analysis and Performance Factors Analysis 9

1.3. Item Response Theory 11

1.3.1. 1-PL, 2-PL and 3-PL Models 12

1.3.2. Parameters Estimation of IRT Models 13

1.3.3. IRT Extensions for Knowledge Tracing 15

1.4. Datasets 17

1.5. Contributions 19

1.6. Declaration of Previous Works 22

1.7. Structure of Dissertation 22

2. DEEP LEARNING AND STUDENT RESPONSE MODELING 23

2.1. Deep Neural Networks 24

2.1.1. Recurrent Neural Networks 26

2.1.2. Memory Augmented Networks 29

2.2. Neural Architecture Search 32

2.2.1. NAS using Reinforcement Learning 33

2.2.2. Sequential Model Based Optimization 35

2.3. Evaluation of Deep Neural Networks 36
2.3.1. Cross Validation 37
2.3.2. Statistical Significance Test 38
2.4. Deep Neural Network Based Models for Knowledge Tracing 40
2.4.1. Deep Knowledge Tracing 40
2.4.2. Dynamic Key-Value Memory Network for Knowledge Tracing 41
3. TOWARDS INTERPRETABLE DEEP NEURAL NETWORK MODELS FOR STUDENT RESPONSE MODELING 44
3.1. Interpretable Deep Learning 45
3.1.1. Transparency 45
3.1.2. Post-hoc Interpretability 46
3.2. Open the Black Box of DKT Model 47
3.2.1. Experiment Setup 48
3.2.2. Prediction Vector Changes and Relation Among Skills 49
3.2.3. Using Synthetic Data 51
3.2.4. Temporal Impact 53
3.2.5. Is the RNN Representation Meaningful? 56
3.2.6. Conclusion 57
4. UNCERTAINTIES IN STUDENT RESPONSE MODELING 58
4.1. Uncertainties in Deep Learning 59
4.1.1. The Uncalibrated Fact of Deep Neural Networks 59
4.1.2. Epistemic Uncertainty 60
4.1.3. Aleatoric Uncertainty 61
4.2. Uncertainties for Knowledge Tracing 62
4.2.1. Aleatoric Uncertainty 63
4.2.2. A Theoretical Analysis of Kendall's method 64

4.2.3. Epistemic Uncertainty 67
4.2.4. Model Training 68
4.3. Datasets 68
4.4. Results 69
4.4.1. Monte Carlo Sampling for Variance 69
4.4.2. Aleatoric Uncertainty 70
4.4.3. Epistemic Uncertainty 72
4.4.4. Synthetic Data 73
4.5. Discussion 75
5. INCORPORATING MORE MODALITIES FOR STUDENT RESPONSE MODELING 77
5.1. The EduAware Learning Module 78
5.1.1. Summary of Tablet Learning Modules 78
5.1.2. Tablet-based Features 81
5.1.3. Study Design 82
5.1.4. Predicting Student Responses 85
5.1.5. Which Feature Subsets Are Most Important? 88
5.1.6. What Specific Features Are Most Important? 89
5.1.7. How Many Self-assessment Exercises Are Required? 91
5.1.8. Conclusion 92
5.2. Multimodal Fusion 93
5.2.1. Early Fusion 93
5.2.2. Late Fusion 94
5.2.3. Intermediate Fusion 95
5.3. Multimodality for Knowledge Tracing 96
5.3.1. Related Work 96

5.3.2. Recurrent Cell Search in The Case of One Modality 98
5.3.3. Fusion Structure Search for Multimodality 101
5.3.4. Extending sub-graph Sampling to Multimodality 103
5.3.5. Extend the LSTM Cell for Multimodality 105
5.3.6. A New Metric for The Measurement of Performance With Time 106
5.3.7. Results 107
6. ATTENTION MECHANISMS AND KNOWLEDGE TRACING 111
6.1. Attention Mechanisms 112
6.1.1. Attention for Recurrent Neural Networks 113
6.1.2. Self Attention 115
6.1.3. Soft/Hard Attention 115
6.1.4. The Transformer 117
6.2. Attention Mechanisms and Knowledge Tracing 118
7. CONCLUSION 124
7.1. Conclusion 124
7.2. Contributions to Other Research Fields 126
7.3. Future Research 127
APPENDIX
BIBLIOGRAPHY 128

LIST OF FIGURES

Figure Page

1.1 The error rate changes with the number of opportunities to apply each rule. This is also called the power law [25]. 5

1.2 Mean learning curve for a complex skill [25]. 9

1.3 Mean learning curves for decomposed two skills [25]. 10

1.4 Item response theory 3-PL model with different discriminative parameter a and difficult level b values. 14

2.1 MuCulloch-Pitts neuron 25

2.2 A simple artificial neural network with one input layer, two hidden layers and one output layer. 26

2.3 A neural network model learns hierarchical representation [47]. 27

2.4 An unrolled recurrent neural network [6]. 28

2.5 Long Short Term Memory (LSTM) network [6]. 29

2.6 End to End Memory Network [123]. 30

2.7 Neural Turing Machine [50]. 31

2.8 Neural architecture search using reinforcement learning [152]. 34

2.9 Parameters sharing among all sub models. Higher level nodes are fully connected with lower level nodes. 35

2.10 10 fold cross validation, reported performance are averaged across 10 folds. Image source from [4]. 38

2.11 Deep Knowledge Tracing Model [107]. 41

2.12 Dynamic Key-Value Memory Network for knowledge tracing [149]. 42

3.1 Activation of filter maps for first and 5th CONV layer of AlexNet when looking at a cat. With each square corresponds to one filter map [1]. 46

3.2 First two components of T-SNE of the activation vector for first time step inputs. Numbers are skill identifiers, blue for correct input, orange for incorrect input. 49

3.3 The prediction changes for one student, 23 steps, correct input is marked blue, incorrect input is marked orange. 50

3.4 **Left:** Activation vector changes for 100 continuous correctness of randomly picked 3 skills **Right:** Activation vector difference of randomly picked 3 skills through time. 52

3.5 Prediction vector after 20 steps for skill #7, #8, #24. 53

3.6 DKT predictions from two different students. The blue line is the prediction of correctness from DKT. The red line is the actual response correctness(1 or 0). 54

4.1 Confidence histograms (top) and reliability diagrams (bottom) for a 5-layer LeNet and a 110-layer ResNet on the CIFAR-100 Dataset [51]. 60

4.2 Architecture of Deep Knowledge Tracing model with uncertainties. 63

4.3 Demonstration of sampled logit and its relation to output variance. 65

4.4 Monte Carlo Sampling for different T, distorted loss is calculated using the noisy logit ($\hat{y}_i$) and the target. Undistorted loss is calculated using the raw logit. ($\mathbf{f}(x_i)^{\mathbf{W}}$) and target. 71

4.5 Average uncertainties change with 50 exercises attempts for our proposed method and Kendall's method for high performing students and mid-range performing students. Notice that our proposed method becomes more certain as a students responds to more questions. Moreover, for high performing students, this uncertainty can be significantly reduced. ... 76

5.1 A participant using the EduAware learning tool. 79

5.2 A flowchart of the learning tool with callouts of various modules. 80

5.3 A Wright Map summarizing the item difficulty and person-ability as estimated by our IRT model. The dark band corresponds to questions selected for prediction analysis. 84

5.4 Model performance in predicting each subset question, paired t-test t = 4.9873, $p < 0.001$ for 3PL model, t = 5.3033, $p < 0.001$ for majority classifier. 87

5.5 Sorted accuracies of different feature subsets and combinations of feature subsets. Error bars signify the 95% interval. 88

5.6 Feature categories selected by RFE for each comprehensive test question. 90

5.7 Performance of EduAware versus the number of questions used to train the system for each participant. 91

5.8 Different levels of fusion. (a) Early level fusion, (b) Late level fusion, (c) intermediate level fusion. 94

5.9 DKT Model with more features [150]. 97

5.10 DKT Model with more features and decision tree classifier [143]. 98

5.11 Discovered recurrent cell for knowledge tracing. 99

5.12 NAS for multimodality fusion with each skill has its own separate encoder. 102

5.13 Extend sub-graph sampling to multimodality. 104

5.14 Extend LSTM cell for Multimodality. 105

6.1 Sequence to sequence modeling with encoder and decoder. Here <EOS> marks the end of a sentence. 113

6.2 The encoder-decoder model with additive attention mechanism [133]. 114

6.3 Color red represents the current word, Color blue indicates the degree of memory activation. [21] 116

6.4 NAS Extend Model with Attention Mechanism. The recent hidden states are cached in the pool. 119

6.5 NAS Extend Model with Attention Mechanism. The recent hidden states for skill #9 are cached in the pool. 120

LIST OF TABLES

Table		Page
3.1	Area under the ROC curve	56
3.2	Square of linear correlation (r^2) results	56
4.1	Dataset Summary	69
4.2	Area under the ROC curve (AUC), Grouped by Different Aleatoric Uncertainty	72
4.3	Square of linear correlation (r2) for Different Aleatoric Uncertainty	73
4.4	Area under the ROC curve (AUC) for Different Epistemic Uncertainty	74
4.5	Square of linear correlation (r2) for Different Epistemic Uncertainty	75
5.1	Participant Demographics	83
5.2	Model accuracy and F1 score for various methods of predicting student responses.	86
5.3	Dataset Statistics	100
5.4	Results Comparison	100
5.5	Dataset Statistics for Multimodality	107
5.6	Comparision of different models for knowledge tracing. SM stands for Single Modality. MM stands for multimodality. SC stands for simple concatenation. FS stands for fusion search.	108
6.1	Comparision of different models for knowledge tracing. SM stands for Single Modality. MM stands for multimodality. SC stands for simple concatenation. FS stands for fusion search. A stands for attention.	122

To my parents

Chapter 1

INTRODUCTION

Our accumulated knowledge in the past few decades is more than that of any other time in history. Human knowledge has gone through the transition from linear growth to exponential growth. To catch up with this speed and stay competitive, we are required to learn more efficiently, which implies more efficient ways of instruction. However, the way we conduct education is not very different from that of thousands of years ago. Even though the appearance of techniques like computers and Internet greatly changed how content is delivered. For instance, massive open online courses (MOOC) [89] allow the access to high quality resources from rural areas. Discussions with other students from all over the world was also made possible. However, the teaching paradigm is almost the same: students (from all around the world) gather together in a classroom (although virtual) to learn one topic with fixed curriculum from one teacher. Besides, the effectiveness of these online courses is also questionable [89]. A lot of research has been conducted and we learnt that the most effective way of learning is through one on one tutoring, thus customized curriculum could be designed for each student [139]. However, this methodology does not scale, considering the fact that mentors or teachers are always limited resources.

To achieve large scale individualized education, we seek Artificial Intelligence (AI) for help. Systematic research about AI could be traced back to the 1940s [114]. Pioneering researchers at that time include John McCarthy, Alan M. Turing, etc. The research field of AI in education was established in the 1970s [15, 16, 23, 116, 117]. There were golden ages for AI research and its applications when researchers failed to appreciate the difficulty. There were also winter times when a lot of projects could not meet the high expectations of investors and funding was cut off. Recently, we have seen the renaissance of AI research, which is largely due to the success of deep learning [47]. Deep learning is the study of neural networks

with many layers (could be hundreds of layers). However, it is not because we have achieved significant breakthroughs from the algorithmic perspective. The computational model of deep neural networks was not too much different from the ones proposed in the late 1940s and the algorithms used to train deep neural networks, which is called backpropagation [113], was proposed in the 1980s. Why then we waited for nearly three decades? Generally speaking, there are two reasons. Firstly, these deep neural networks with millions of parameters usually require a lot of training data. We did not have such amount of data available until recently. Secondly, the training process is very computational demanding. Even in nowdays, with current computers, the training process for a mediocre network could take days if not weeks.

Deep Learning has achieved tremendous success in fields like Natural Language Processing (NLP), Computer Vision and Healthcare [31, 37, 38, 77, 132]. Yet, we still haven't seen very exciting breakthroughs in the field of education. Most existing intelligent tutoring systems are still very naive [139]. One student usually has to work through a fixed series of exercises. When the system detects an incorrect response, fixed hints will be given. Thus, one student is always in a recognized path. It is difficult to add more exercises or change the curriculum dynamically. The passing of information is usually one way, from the system to the student. To infer the mastery level of different skills for one student, most intelligent tutoring systems only utilize the *correctness* of one problem (skill), despite the fact that other user activities like *time spent*, *number of hints* will also be collected. However, such a situation is understandable, since student response modeling is a much more challenging task than object recognition, which is pretty much solved. Different students may have different abilities when solving different problems. Different problems usually involve several different skills with different difficult levels. These different skills could be correlated to each other. One student who has already mastered some skill could also make mistakes (slip). One who has not mastered the corresponding skill might have a correct guess. Other factors including emotion, fatigue could also play important roles. Building a successful intelligent tutoring system requires expertise from domains like education, computer science, psychology, etc.

Intelligent Tutoring Systems (ITS) [139] is one of the applications of AI in education. A successful intelligent tutoring system should at least include the following four modules [139]: 1) Student Knowledge Module. The system must have a good understanding of the student currently involved. For example, which skills the student has mastered, which ones need more practice. What's more, an intelligent system should also take factors like emotion, fatigue into consideration, as they could greatly impact the learning efficiency. 2) Teaching Knowledge Module. What is the best teaching strategy for this student at the current moment? When to intervene or motivate? 3) Communication Knowledge Module. What is the best way to communicate with the student? Use natural language, texts or videos? Different students may have different preferences. 4) Evaluation Knowledge Module. The system must have an efficient way to assess the performance of a student, thus curriculum could be adjusted to either give more remedial exercises or to move to more advanced topics. However, we are still very far away from building such an ITS. Most current works only target one or two modules [139]. In this thesis, I will only focus on student response modeling, more specifically on Knowledge Tracing (KT) [25]. KT is a technique of tracking the mastery level of each skill for one student, thus customized curriculum could be designed. It is considered as a core component of the student knowledge module. Student response modeling is a broader concept than knowledge tracing, but in this thesis I will use these two interchangeably.

Knowledge Tracing (KT) uses all the information available to build a 'knowledge' model of a student. Since this model is updated whenever new information is available, it could 'trace' the knowledge changes of one student. This is different from other student response modeling techniques that have a separate evaluation phase. Using this model we could predict the next response for a student on a specific topic. Knowledge tracing allows the inference of the mastery level of a skill for a student, thus is the key for building intelligent tutoring systems. I will use skills, knowledge components (KCs), production rules, or topics interchangeably in this thesis. Given a learning domain, the skills are usually discovered manually by domain experts (although, there are some models that could automatically discover these skills [107]). For instance, one skill could be knowing how to calculate the

area of a circle in geometry. These skills are considered the basic learning units. For the following discussions, I assume these skills are already given (For each problem a student tries, we know the skill or skills related to the problem. One problem or exercise may have several steps, with each step corresponds to one or more skills. This is domain dependent).

In section 1.1, I will briefly describe the history of knowledge tracing, introducing the most recognized works in this field and summarizing the advantages and limitations of these models. Learning Factors Analysis (LFA) [17] and Performance Factors Analysis (PFA) [102] will be covered in section 1.2. Then, I will talk about Item Response Theory (IRT) [40] in section 1.3. IRT models are widely used in the assessment community. A lot of knowledge tracing models are built upon IRT models. A briefly description of the public datasets that will be used throughout this thesis is presented in section 1.4. The contributions of this thesis will be summarized in section 1.5. A declaration of previous works and the structure of this thesis will be given in section 1.6 and 1.7.

1.1. Knowledge Tracing

Knowledge Tracing (KT) refers to the process of tracking the changes of mastery level of skills for one student. As mentioned above, given a domain, skills are usually discovered by domain experts manually through brainstorm. Take Geometry as an example, calculating the area of a circle could be considered as one skill. There are underlying two assumptions here. First, each skill is considered to be a basic learning unit and not dividable. Second, skills in a domain are independent to each other. However, these two assumptions are usually not hold in practice. Since these skills are discovered manually, it is not guaranteed that each skill represents the basic learning unit. It is possible for one skill to be decomposed into sub skills or some skills to be combined into one skill to achieve better prediction performance. Also, some skills could be correlated to each other. A lot of research works for student response modeling actually were proposed to tackle these problems. I will describe them in the following sections.

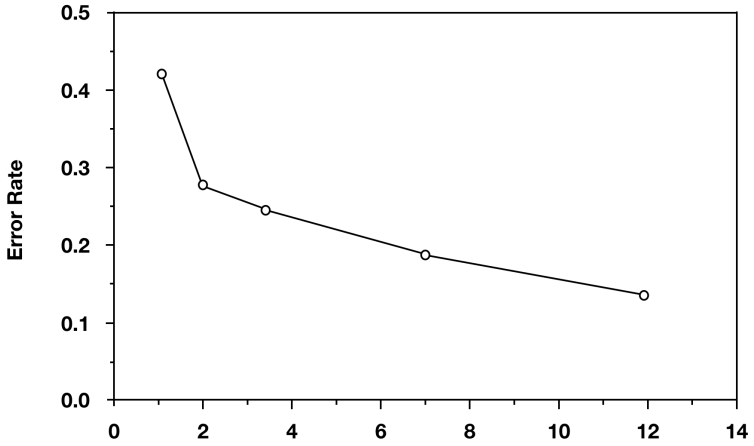

Figure 1.1. The error rate changes with the number of opportunities to apply each rule. This is also called the power law [25].

Our goal in learning is to master all corresponding skills in a domain. Mastery learning [25] could be achieved if: 1) A domain knowledge is decomposed into hierarchical skill components. 2) Prerequisite skills are learnt first before higher level skills in this hierarchy. A curriculum is designed to make sure prerequisite skills are practiced first. Based on this assumption, each skill component is the smallest learning unit. Figure 1.1 gives the mean learning curve [25]. The x-axis is the number of opportunities to apply each rule and the y-axis is the error rate. This is also called the power law, as we can see the error rate drops monotonically with time. However, if we use a more superficial unit like exercise, no systematic learning curve could be found (an exercise could contain more than one skill). A set of skills from one domain defines a cognitive model. A high quality cognitive model is a prerequisite of building successful intelligent tutoring systems. If one skill generated through brainstorm could not actually reflect the underlying truth (could be decomposed into sub skills), the prediction performance suffers. Prior to knowledge tracing, an intelligent tutoring system usually uses what is called model tracing. In model tracing, an ITS has an ideal student model and the curriculum is usually fixed. The system solves a fixed sequence of exercises together with the student, each exercise may contain several steps, with each step corresponds to one or several skills. When one student makes a mistake, the intelligent

system will give some pre-programmed hints, until this student successfully moves to the next step. Thus, this student is always in a recognized position. Apparently, this methodology is not very efficient and flexible. All students solve the same fixed number of exercises and no individual difference is taken for consideration. Ideally, we want to give more remedial exercises to those students who struggle, but fewer to those who already mastered these skills. Knowledge tracing could be used to dynamically generate a series of practices based on the mastery level of different skills for a specific student, thus is considered to be a better alternative to model tracing.

1.1.1. Bayesian Knowledge Tracing

Bayesian Knowledge Tracing (BKT) [25] is a type of Hidden Markov Model (HMM), with latent variables modeling the skills and observed variables modeling the responses. BKT assumes a binary state of one skill for a student as either mastered or not, as well as the response of a question from a student as either correct or incorrect. It builds a model for each student skill pair and updates the model using Bayes's rule after observing the responses. There are four parameters in the BKT model:

- $p(L_0)$, our prior knowledge about the probability of being in a learned state before seeing any evidence.
- $p(T)$, the probability of transition from an unlearned state to the learned state after an opportunity to apply it.
- $p(G)$, the probability that a student will have a correct guess despite in an unlearned state.
- $p(S)$, the probability that a student will make a mistake (slip) even if in the learned state.

In the standard implementation of BKT, these four parameters are item specific. In other words, these four parameters are different from item to item, but shared by all students. No student specific parameters means that all students learn at the same rate. There is also no

forgetting parameters in the BKT model. Each student's state is updated using the following equation:

$$p(L_n) = p(L_{n-1}|evidence) + (1 - p(L_{n-1}|evidence)) * p(T) \tag{1.1}$$

Equation 1.1 means a student will be in a learnt state if this student was already in the learnt state in the previous step or not in the learnt state but will be transferred into this state after one attempt. The conditional probability $p(L_{n-1}|evidence)$ could be easily calculated after seeing the responses from the student. Then we could use these probabilities to predict the next response for that student using the following equation.

$$p(C_{is}) = p(L_{rs}) * (1 - p(S_r)) + (1 - p(L_{rs})) * p(G_r) \tag{1.2}$$

The probability of a student s will have a correct response to item i associated with skill r is the sum of 1): the probability that this student has already mastered skill r and will not make a mistake. 2) the probability that the student has not mastered the skill r, but has a correct guess. Please note here, we have such a model for each student item pair.

The biggest advantage of BKT is its simplicity and interpretability. However, BKT assumes skills are independent, which is usually not true in the real world (The hard part is to find independent skills, because this process is usually conducted manually by domain experts. It is unknown if there are truly independent skills. If there are, the real problem is how to find them). Another limitation of the BKT model is that, in the standard implementation of BKT, the parameters are skill specific, which means all students practice the same skill will share these four parameters. This is also not true in the real world. Different students may have different learning rate and their prior knowledge will also be different. In the original BKT paper [25], authors discussed using different four parameters for each student and four parameters for each skill, and then do a combination through some function. However, it did not improve the performance.

Yudelson *et al.* [146] introduced student specific parameters and proposed the Individ-

ualized Bayesian Knowledge Tracing model. Experiment results show that, incorporating student specific parameters improves the model performance. Also, modeling the learning rate parameter is more effective than modeling the prior knowledge parameter. The parameters of BKT are usually learnt using algorithms like Expectation Maximization (EM) [28] or conjugate gradients [56].

1.1.2. Bayesian Networks for Knowledge Tracing

We can think of BKT as a simple dynamic bayesian network, in which each skill is modeled as a hidden variable. However, skills are independent to each other in BKT (there is one model for each student skill pair). There are some works that based on more complicated bayesian networks can capture the relations among skills [66].

Käser *et al.* [66] proposed using dynamic bayesian networks (DBNs) for knowledge tracing. Firstly, a hierarchical skills topology is built by domain experts. Each skill is associated with a series of parameters. For instance, for each skill S there is a guess parameter $1 - p_G = p(observe = 0|S = 0)$ and slip parameter $p_S = p(observe = 0|S = 1)$, which are similar to those from the BKT model. There are also parameters that model the learning and forgetting for each skill p_L, p_F. The current state of some skill, for instance S_3 in time step t does not only depend on the state of S_3 in its previous state, but also the states of its parents S_1 and S_2. Skills are modeled as hidden variables h, while student responses are observed data y. The learning process is to find a parameters setting θ that maximize the likelihood for student m.

$$L(\theta) = \sum_m ln \left(\sum_{h_m} p(y_m, h_m|\theta) \right)$$

They tested on 5 different datasets and showed that explicitly modeling skills topology could achive better results compared with models like BKT. However, the authors also argued that the improvement is small in domains that do not require hierarchical representation, for instance, spelling learning. One limitation of this work is the requirement of building

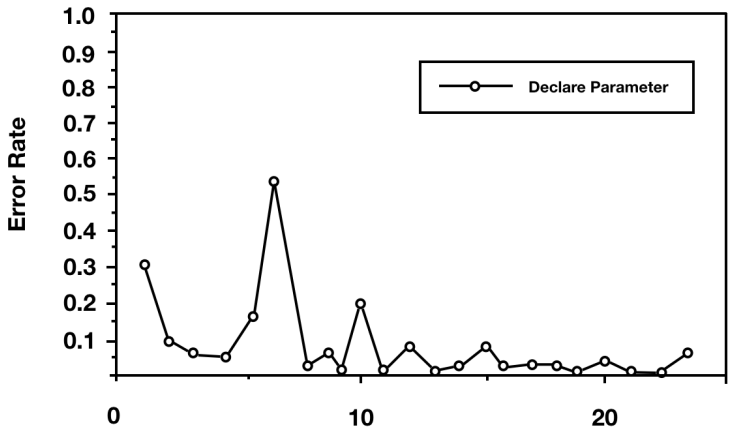

Figure 1.2. Mean learning curve for a complex skill [25].

a skills topology manually by domain experts. This process is tedious when the domain is complicated and might bring in human bias. Deep Learning based models like Deep Knowledge Tracing (DKT) [107] and Dynamic Key-Value Memory Networks (DKVMN) try to model skills relation implicitly. I leave the discussion of these models in later chapters.

1.2. Learning Factors Analysis and Performance Factors Analysis

The performance of student response modeling largely depends on the accuracy of the underlying cognitive model. If the cognitive model discovered by domain experts through brainstorm could not reflect the ground truth (or at least close to), the performance of student response modeling may suffer. For instance, some skill might be further decomposed into sub skills.

To alleviate this problem, Cen *et al.* [17] proposed a semi-automatic approach called Learning Factors Analysis (LFA) to improve the cognitive model. They proposed a probabilistic model by extending the power law model. The standard form of the probabilistic model from LFA is given by the following equations:

$$m(i, j \in KCs, s, f) = \alpha_i + \sum_{j \in KCs} (\beta_j + \gamma_j n_{i,j}) \tag{1.3}$$

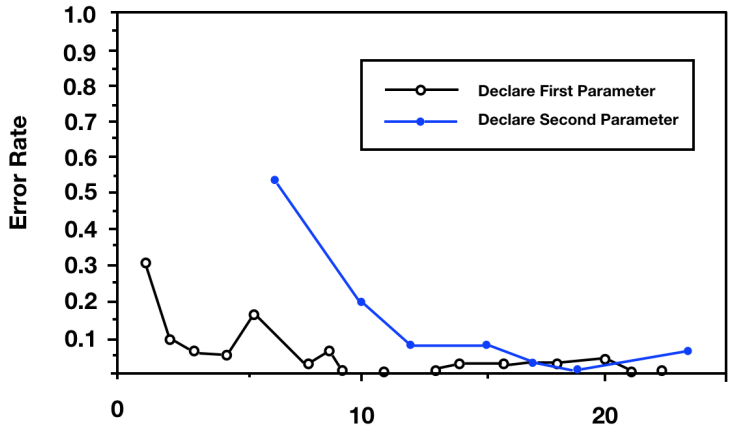

Figure 1.3. Mean learning curves for decomposed two skills [25].

$$p(m) = \frac{1}{1 + \exp^{-m}} \tag{1.4}$$

where m is the accumulated learning, α_i captures the ability for student i, β_j is the difficult level of item j. $n_{i,j}$ represents the number of opportunities to apply skill j for student i. And γ_j is a discriminative parameter. A logistic model is used to get the probability p of getting an item correct. Using this probabilistic model, for a given cognitive model, we could evaluate its performance by observing how well this cognitive model fits our data. The next part of LFA is a semi automatic way of improving the cognitive model. First, domain experts identify which skills could be further decomposed into sub skills and which skills could be combined into one. For example, a CIRCLE-AREA skill which is to calculate the area of circle could be further decomposed into CIRCLE-AREA-ALONE and CIRCLE-AREA-EMBEDDED. That is because students find it harder to calculate the area when it is embedded in another shape, for instance a square. Figure 1.2 gives the mean learning curve for a complex skill. As we can see, it is not very smooth and no systematic trend is observed. After decomposing this complex skill into two sub skills, the learning curve is more smooth, as shown in Figure 1.3. Using search algorithms like A*, through splitting and combining nodes, a better cognitive model could be found. Two scoring functions were used. Akaike Information Criterion (AIC) and Bayesian Information Ceriterion (BIC). The

choice between these two is a trade off between the complexity of the model (for instance, number of parameters) and the log likelihood.

Performance Factors Analysis (PFA) [102] also uses logistic regression to model the accumulated knowledge. It is based on Learning Factors Analysis (LFA) [17] and overcomes the limitation that LFA only counts the opportunities in its model but ignores the responses from the student. The model of PFA is given by:

$$m(i, j \in KCs, s, f) = \sum_{j \in KCs} (\beta_j + \gamma_j s_{i,j} + \rho_j f_{i,j}) \tag{1.5}$$

where, β_j is the easiness for item j. $s_{i,j}$ is the success counts of skill j for the student i. $f_{i,j}$ is the failure counts of skill j for the student i. γ_j and ρ_j are discriminative parameters. The accumulated knowledge m is sent to a logistic function to give a probability of getting item j correct. Compared with LFA, as we can see the student specific parameter has been removed, since it is usually not feasible to estimate the student ability ahead of time for a tracing model. LFA model is more used in a data mining context and it is not a knowledge tracing model. However, after the changes, we could use PFA for knowledge tracing. PFA calculates the accumulated learning as a function of the number of success attempts and failure attempts for different skills. Compared with BTK, PFA punishes the model less for each failure attempt for the student, which is believed to be one of the reasons that PFA performs better than BTK on some datasets.

1.3. Item Response Theory

Item Response Theory (IRT) [40] is a family of probabilistic models that are widely used in the assessment community. Strictly speaking, IRT models are not knowledge tracing models, because they assume the student ability or the latent trait does not change with time. For this reason, they are considered static models and are mainly used for exercises development. However, there are a lot of knowledge tracing models that are built upon IRT. In this section, I will first introduce the basic ideas of IRT models, then talk about their

extensions for knowledge tracing.

IRT models assume that one student's ability could be summarized by one single latent variable (latent trait). With larger value of this latent variable, the probability of a correct response is also higher. IRT models try to capture the link between this latent variable and its manifestations (the responses). There are mainly four parameters involved in IRT models. θ_s is the latent trait, modeling student s' ability. b_i models the difficult level of each item i. a_i is a discrimination parameter of item i, which is often useful when students have similar abilities. c is the guess parameter, which is used to model the situation that a student could have a correct guess even without mastering the corresponding skill. Based on the number of parameters involved. IRT models could be categorized as 1-PL model, 2-PL model and 3-PL model.

1.3.1. 1-PL, 2-PL and 3-PL Models

The most simple model is the 1-PL model and is given by Equation 1.6.

$$p(correct|\theta_s, b_i) = \frac{\exp(a(\theta_s - b_i))}{1 + \exp(a(\theta_s - b_i))} \tag{1.6}$$

It involves the latent variable for the student θ_s and one parameter b_i which is the difficult level for item i. a is a constant in this model. We can see the 1-PL model is just a logistic function that squashes the difference between the student ability θ_s and the item difficult level b_i to a number between 0 and 1. We can think of this number as the probability of getting one item correct. The larger the difference between the student ability θ_s and the difficult level of item b_i, the higher the probability of having a correct response. Thus, it is straightforward to interpret this model. There is a special case when $a = 1$.

$$p(correct|\theta_s, b_i) = \frac{\exp(\theta_s - b_i)}{1 + \exp(\theta_s - b_i)} \tag{1.7}$$

We call it Rasch Model. In practice, the Rasch model usually performs better because it constrains itself to measure the variables of interest. Especially when the size of the dataset is

limited, more variables provide more challenges for estimations. However, sometimes, when students have very close abilities, adding a discrimination parameter a_i is still beneficial, such as:

$$p(correct|\theta_s, a_i, b_i) = \frac{\exp(a_i(\theta_s - b_i))}{1 + \exp(a_i(\theta_s - b_i))} \tag{1.8}$$

Equation 1.8 indicates the 2-PL model. The discrimination parameter a_i is different for each item i. Sometimes, we also want to model the situation in which a student could have a correct guess even without mastering the corresponding skills. We could incorporate a guess parameter c here. The parameter c is usually selected to be 0.25 if a multiple choice question has 4 options (0.2 for 5 options).

$$p(correct|\theta_s, a_i, b_i) = c + (1 - c)\frac{\exp(a_i(\theta_s - b_i))}{1 + \exp(a_i(\theta_s - b_i))} \tag{1.9}$$

Figure 1.4 shows a 3-PL model with different values for a and b. As we can see, when having the same value for a but different values for b, the curve moves horizontally. In the case of having the same value for b but different values for a, larger a is more likely to discriminate students with small ability differences.

IRT models assume the responses for different items are independent given the latent trait variable. Thus, parameters could be simply estimated using Maximum Likelihood or Expectation Maximization Algorithms. Once we get these parameters, they usually will not change with time. Thus, 1-PL, 2-PL and 3-PL models are considered static models. There are no "tracing" in these models. But these models do give us a good starting point. We will see in later sections how can we extend IRT models for knowledge tracing.

1.3.2. Parameters Estimation of IRT Models

Most algorithms used for the estimation of parameters of IRT models assume: 1) Given student latent variable θ, the responses for different items are independent. 2) All students are independent from each other. We usually will have two settings of parameters in IRT

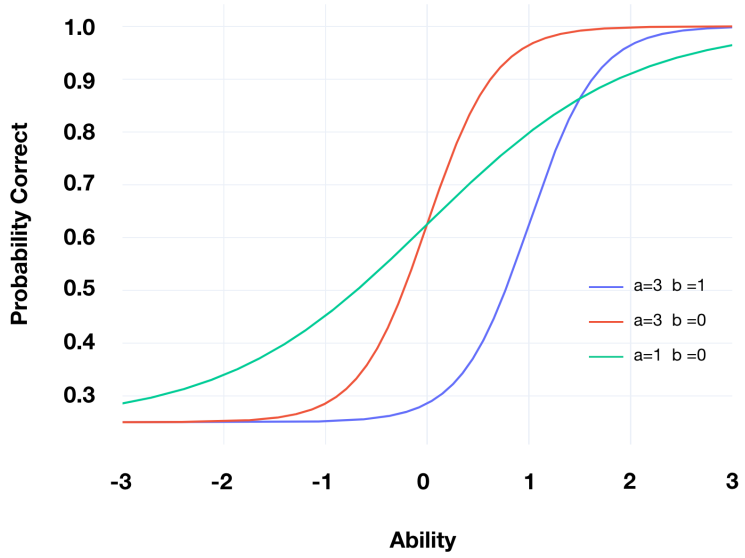

Figure 1.4. Item response theory 3-PL model with different discriminative parameter a and difficult level b values.

models: student related parameters θ and item related parameters ϕ. If we know either of them, to estimate the other parameters is easy based on our assumptions. However, sometimes, it is required to estimate these two types of parameters simultaneously. I will describe two popular algorithms that are all based on likelihood maximization. A more comprehensive description of these algorithms could be found in [64].

Joint Maximum Likelihood (JML) assumes both the student latent variable θ and item related parameters ϕ (includes discrimination parameter a, difficult parameter b, etc) are unknown, but fixed. The likelihood $L(\theta, \phi, X)$ could be decomposed into $\prod_s L_s(\theta_s|x_s, \phi)$, which could be maximized with respect to θ_s and ϕ.

Marginal Maximum Likelihood (MML) takes a different approach from JML. MML assumes the student latent trait is sampled from some distribution $F(\theta)$. This distribution does not need to be continuous. By integrating out the student latent variable, we have a marginal distribution

$$Pr(x_s|\phi) = \int_\theta L_s(\theta|x_s, \phi) dF(\theta) \tag{1.10}$$

Thus, we have the marginal likelihood for the parameters ϕ:

$$L(\phi|X) = \prod_{s} Pr(x_s|\phi) \tag{1.11}$$

There are other algorithms available for the parameters estimation like Conditional Maximum Likelihood and Bayesian Estimation with Markov Chain Monte Carlo, interested readers could refer to this survey [64].

1.3.3. IRT Extensions for Knowledge Tracing

As discussed above, IRT models are static models, which means the student ability (latent variable) does not change with time. Because IRT models were first developed for psychometrics. In the context of education, these models are often used for assessment and items development. Parameters are estimated using algorithms like Marginal Maximum Likelihood [64]. Using a wright plot we could easily see the distributions of the student abilities and item difficulties. However, there are a lot of knowledge tracing models that are built upon IRT models. In this section, I will discuss some of the works that are most recognized by the learning science community.

One possible way of using IRT for knowledge tracing without much modification could be conducted like this. A portion of the dataset is reserved for parameters estimation (for example, 20%). Once we have all the values for these parameters, the values of non-human related parameters will be fixed. When making a prediction of the response for time index t, we will estimate the human related parameters using all the data till time $t-1$. Please note here, we assume the data used for item parameters estimation include all skills. The human related parameters will be reevaluated after the response at each time step. But, the item related parameters will be fixed.

Now, let's take a look at another approach of extending IRT models for KT. The probability of getting an item i correct could be expressed using KT:

$$P(y_{i,t}|y_{i,1}, y_{i,2}, ...y_{i,t-1}) = \sum_{l \in (0,1)} P(y_{i,t}|k_t = l)P(k_t = l|y_{i,1}, y_{i,2}, ...y_{i,t-1}) \tag{1.12}$$

where l is the knowledge state, 1 represents learnt and 0 represents unlearnt. To combine IRT with KT, we could replace the emission probability $P(y_{i,t}|k_t = l)$ with an IRT model, which is basically a logistic function. The combined model becomes:

$$P(y_{i,t}|y_{i,1}, y_{i,2}, ...y_{i,t-1}) = \sum_{l \in (0,1)} logistic(d_i, \theta_s, c_l)P(k_t = l|y_{i,1}, y_{i,2}, ...y_{i,t-1}) \tag{1.13}$$

The logistic model is parameterized with the difficult level of an item d_i, the student ability θ_s and a bias c_l indicating if the current student is in a learnt state or not. In [70], the authors compared two works that combined IRT and KT: FAST [46] and LFKT [69]. These two models only differ in the training process and the way these parameters are learnt. A detailed description could be found in [70].

Another recent work [144] combines deep neural networks with IRT, exploiting the representational power of deep neural networks and the interpretability of IRT model. In this work, the authors use deep neural networks to learn the student ability θ_i and item difficult level b_j, then these two variables will be used by an IRT model to make a final prediction. More knowledge tracing models based on deep neural networks will be discussed in the next chapter.

Models like BKT, LFA and PFA share one common advantage: interpretability. In other words, all the parameters used are meaningful and the features are easy to explain. For example, the guess parameter in BKT model represents the probability of having a correct guess even in an unlearnt state. However, deep learning based models which we will talk in the next chapter do not have this advantage. How the final decisions are made for those models are unclear and the features learnt by the deep neural network models are hard to explain. The advantage of deep learning models is the ability to learn hierarchical representation, which will be described in chapter 2.

1.4. Datasets

The public datasets that are used throughout this thesis are described below. Novel dataset will also be used when evaluating multimodality and will be discussed in detail in the corresponding chapter. In the following datasets, I have removed students with less than three attempts and those having empty values for some features when considering multimodality. The statistics of each dataset used for different purposes will be given in corresponding sections. In this section, I give an overall description of these datasets.

Assistment skill builder 09-10 The ASSISTment system is an online tutoring system originally built on 8^{th} grade MCAS test items (mathematics) [42]. The grade 8 mathematics test includes the following five domains: The number system, Expressions and equations, Functions, Geometry, Statistics and probability. A sample test item looks like the following: *Between which pair of numbers on a number line does* $\sqrt{6}$ *lie?*

A. 2.3 and 2.5

B. 2.5 and 2.7

C. 2.7 and 2.9

D. 2.9 and 3.1

This dataset is gathered in school year 2009-2010 and is one of the most used datasets for comparing knowledge tracing models. A student is considered to have mastered one skill when meeting some criterion (for instance, correctly answered three problems related to that skill in a row). After mastery, different problems related to other skills are given. Xiong *et al.* [140] reported a duplication issue in the 09-10 dataset and created a variant of the dataset with duplicates removed. Authors of this dataset have been noticed and the duplication issue has been fixed. I use an updated version of skill builder 09-10 dataset with removed duplicate entries, directly downloaded from this link[1]. Since there are different

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

versions of this 09-10 dataset used in the wild by researchers, the results reported in this study might be marginally different from those reported in other works.

KDD 2010: The 2010 KDD cup challenge [122] is to predict student performance. Data were collected from the Carnegie Learning's Cognitive Tutoring System (Algebra system and the Bridge to Algebra system) for K12 education from school year 2008 to 2009. The system covers the following mathematics strands: Number and operations, Geometry, Measurement, Probability and statistics, and Algebra. A sample question looks like *Find the unique factor pairs of 72.* In this dataset, students are required to solve problems, with each problem consisting of several steps. Each step is associated with one or more knowledge components (skills).

OLI Engineering statics 2012 Fall: The Carnegie Mellon's Open Learning Initiative (OLI) is an online platform that provides customized learning. Compared with the previous two learning systems, OLI aims for higher education. The OLI Engineering Statics course [74] is the study of methods for quantifying the forces between bodies. The data were collected for the term 2012 Fall. A sample question looks like the following:
Can the combination of forces A & B be represented as a couple?

A. definitely not

B. maybe

C. definitely yes

When one problem or step involves more than one skill, different dataset handles this situation differently. For the assistment 09-10 dataset. It has two versions available. For one version, if one problem has two skills associated, there will be two transactions in the dataset with the same response. For the second collapsed version, these two skills will be combined into one new skill. Such that there is only one transaction in the dataset. I used the previous version. For the other two datasets, a new skill is formed if there are more than one skill associated with the step. The first two datasets are from the domain of mathematics and the last is from engineering. This is justified by the fact that these domain is well defined,

in which it is much easier to define skills like AREA-CALCULATION. It is much harder to define suitable skills in other domains like literature. I leave investigations in other domains for future work. Also, for the following discussions, I assume all the skills related to these problems (steps) are known, I do not discuss techniques to automatically find these skills.

1.5. Contributions

In this section, I summarize all the contributions of this thesis. I investigate deep neural network based student response modeling. There are mainly two goals in this thesis: 1) To have a better understanding of existing deep neural network based models and their predictions through visualization and through incorporating uncertainties. This is what I call interpretable deep models for knowledge tracing. 2) To improve the performance of student response modeling with multimodality and attention mechanisms. Even though all the methodologies are evaluated in the context of student response modeling, many of the contributions also influence the machine learning community. I mark those contributions with bold font. I divide my contributions into five general areas, listed as follows:

- Explanation about why deep knowledge tracing has less depth than anticipated. These contributions have been disseminated in the work [33].
  - **I design and use simulated data and activation vectors to analyze question-answering behaviors in high dimensional space**.
  - I modify and explore the deep knowledge tracing model, finding that some irrelevant information is reinforced in the recurrent architecture.
  - I find that an untrained deep knowledge tracing model (with gradient descent applied only to layers outside the recurrent architecture) can be trained to achieve similar performance as a fully trained deep knowledge tracing architecture.
- Proposed a practical way of incorporating uncertainties in student response modeling [35].
  - **I empirically show that Monte Carlo alone is not sufficient to learning**

the data variance.

  - **I propose a regularizer that could guide the training process to result in sensible uncertainties.**
  - **I propose using simulated data to quantitatively understand the generated uncertainties.**

- The EduAware system, that uses tablet based features to improve student performance prediction [36].
  - An evaluation of EduAware in a user study with 21 college students is presented, showing that EduAware can identify if a user will respond correctly or incorrectly to specific questions with 87.7% accuracy (without use of any personalized calibration data)
  - An analysis of the most important types of features for predicting responses by comparing different model performances as complementary feature sets are included or excluded. Recursive feature elimination is employed to understand the most important aspects. Both methods reveal that tablet interactions significantly improve prediction accuracy.
  - Different levels of generalization by altering training sets is investigated. The conclusion is that performance suffers significantly without content-specific training data.

- I propose using neural architecture search for optimal recurrent cell design for knowledge tracing [34].
  - I propose a new recurrent cell which performs better than the LSTM cell using reinforcement learning. McNemar's test is conducted, showing the predictions from the proposed cell and LSTM cell are significantly different.
  - **I propose the concept of sub-model sampling as a method of regularization for NAS methods, finding that sub-model sampling is an effective strategy to mitigate overfitting.**

- I propose better performance knowledge tracing models by combining neural architecture search, multimodality and attention mechanisms.
  - I extend neural architecture search to multimodality for the looking of the best fusion architecture. I show the found fusion architecture performs better than simple concatenation.
  - **I propose to combine network architecture search and multimodal fusion search. To my best knowledge, it is the first model that combines these two ideas in a unified methodology.**
  - I investigate attention mechanisms in the context of knowledge tracing.
  - I propose using a new metric, weighted AUC (wAUC) for the assessment of knowledge tracing models. Compared with AUC, wAUC could be used to measure how one model performs with time.

Other research works completed during my doctoral study, but not included in this dissertation:

- *Swapped Face Detection using Transfer Learning and Human Subjects assessment* [38]. This is a collaborative work with Zohreh Raziei, Eric C Larson, Eli V Olinick, Paul Krueger and Michael Hahsler. In this study, we created a large swapped face dataset and built a deep convolutional neural network using transfer learning for the detection of swapped faces.
- *Measuring Oxygen Saturation with Smartphone Cameras using Convolutional Neural Networks* [37]. This is a collaborative work with Damoun Nassehi and Eric C. Larson. In this work, we collected video data using smartphone cameras, then built a convolutional neural network to estimate the oxygen saturation in arterial blood.
- *PupilNet, Measuring Task Evoked Pupillary Response using Commodity RGB Tablet Cameras: Comparison to Mobile, Infrared Gaze Trackers for Inferring Cognitive Load* [132]. This is a collaborative work with Chatchai Wangwiwattana and Eric C Larson. In this work, we collected video data using tablet cameras and built a convolutional neural network to estimate the pupil changes intrigued by different cognitive load.

1.6. Declaration of Previous Works

- Part of Chapter 3 was published in the educational data mining conference in 2019 [33].
- Part of Chapter 4 was published in the Neurocomputing journal, 2020 [35].
- Part of Chapter 5 was published in the Interactive Learning Environments journal, 2019 [36].

However, these initial drafts have been repurposed and greatly extended. New analysis are also added. While I am writing this thesis, some new analysis of chapter 5 are being published in the learning @scale conference [34].

1.7. Structure of Dissertation

We have seen conventional probabilistic models for knowledge tracing. The biggest advantage of these models is interpretability. It's easy to interpret the features and each parameter. Deep neural network based models do not share this advantage. In Chapter 2, I will introduce the basic concepts of deep learning and its application for knowledge tracing. I propose interpretable deep neural networks for knowledge tracing. This is tackled from two perspectives. In chapter 3, I use visualization to analyze the behaviors of Deep Knowledge Tracing (DKT) in high dimensional space. This allows a better understanding of the inner workings of deep neural network based knowledge tracing models. Modeling uncertainties for knowledge tracing will be covered in Chapter 4. Extending the deep neural network based knowledge tracing models with multimodality and attention mechanisms will be discussed in Chapter 5 and Chapter 6. A summary of this dissertation and future research will be presented in Chapter 7.

Chapter 2

DEEP LEARNING AND STUDENT RESPONSE MODELING

We have seen significant improvements deep neural network models have achieved in domains like image processing, natural language processing and healthcare [31,37,77]. What's more, a lot of these models have been deployed in the real world with the help of tools like Tensorflow and Pytorch [7,101]. Use face recognition technique built on convolutional neural networks [77] to unlock screen or make a payment has been a common practice. In May 2017, AlphaGo Master [119] beat KeJie, the world ranked No.1 professional Go player in a three game match. The empirical success of deep neural networks has encouraged researchers in the learning science community to take similar approaches. Several deep neural network based knowledge tracing models have been proposed in recent years [8,98,107,144,149]. All these deep neural network based models have achieved better performance compared with conventional probabilistic models like BKT and PFA. However, deep neural network based models have limitations too. It is not clear, given a task, how one model adapts its weights during training. Thus, most researchers use these models as black boxes. Further research are needed to fully understand these models.

In this chapter, I introduce deep neural networks and their applications in knowledge tracing. In Section 2.1, I give a brief description of deep neural networks. I will mainly describe two types of neural networks: recurrent neural network and memory augmented network. In Section 2.2, I will introduce Neural Architecture Search (NAS), which is a very promising technique for the automatic design of neural architectures. Evaluation of deep neural network based models will be discussed in Section 2.3. Then I will discuss the KT models based on these deep neural networks, their advantages and limitations in Section 2.4. In the next chapter, I present the work that tries to open the black box of deep knowledge tracing through visualization. Specifically, using activation vector and synthetic data, I

explain why deep knowledge tracing model might has less depth than anticipated.

2.1. Deep Neural Networks

Deep Neural Networks or Deep Learning [78] refers to the study of using deep artificial neural networks to automatically learn hierarchical representations. Despite the fact that the concept Deep Learning was coined a few years ago, artificial neural networks have been in existence for decades. The idea of artificial neural network was loosely inspired by how biological neural network works (To keep it short, I will use neural network instead of using its whole name artificial neural network in the following texts). The first computational neural network model was proposed in 1943 by Warren MuCulloch and Walter Pitts [90]. Figure 2.1 shows the MuCulloch-Pitts Neuron. For each input x_i there is a corresponding weight parameter w_i. Then an activation function is applied on the summation $\sum w_i x_i$. This process is to simulate how the actual biological neuron works in our brain. The dendrites take electrical signals as input, when the accumulated electrical signal is above some threshold, it fires and pass this signal through axon to other neurons (Note this is just a simplified process of how the biological neuron actually works). We could put together multiple such neurons to form a layer. Use multiple layers we could form a neural network as shown in Figure 2.2.

For a regression task, take the output from the neural network $h(x^i)$ and the true label y^i, we could construct the following loss function:

$$J(\theta) = \frac{1}{2N} \sum_{i=1}^{N} (h(x^i) - y^i)^2 \tag{2.1}$$

where N is the number of total samples in the dataset, θ refers to all the parameters. We could use cross entropy loss for a classification task. For complex models like deep neural networks, the loss function is usually very complicated and non-convex. Thus, it is often impossible to find a global optimal. Instead, we usually randomly initialize the weights of one neural network, then iteratively tweak the weights to minimize the loss function.

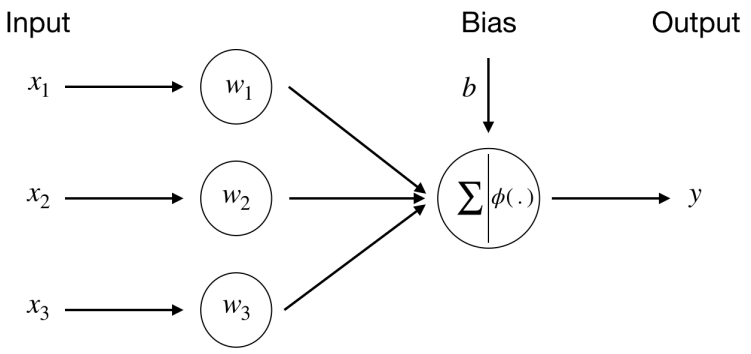

Figure 2.1. MuCulloch-Pitts neuron

Backpropagaton [113] is an application of chain rule to calculate the partial derivative of the loss function $J(\theta)$ with respect to each parameter W_{ji}^k. Here, W_{ji}^k represents the weights for the ith input of the jth neuron in layer k. Once we have the partial derivatives, we could update our weights using gradient descent algorithms [112].

$$W_{ji}^k = W_{ji}^k - \alpha \frac{\partial}{\partial W_{ji}^k} J(W_{ji}^k) \tag{2.2}$$

where α is the learning rate. For complex models like deep neural networks, we could only hope to find a good local optimal. Thus, the initialization of the weights [45,54] and the choice of learning rate usually play important roles on what local optimal we are ended up with. Popular gradient descent based optimizers include Adam [72], RMSprop [5], Adadelta [148], etc. Neural networks differ in the number of layers, the width of each layer, activation function used, etc.

We call neural networks like the one shown in Figure 2.2 shallow networks, because it only has one input layer, two hidden layers and one output layer. Deep neural networks usually contain tens (even hundreds) of layers. Even though, it has been proved that feedforward neural network with only one hidden layer, as long as it is wide enough, is able to approximate any functions [27]. However, deep but narrow networks are often preferred for at least two reasons: 1) It saves parameters. 2) Deep neural networks allow the learning of hierarchical representation, which is especially useful for imaging processing. For example, in an object

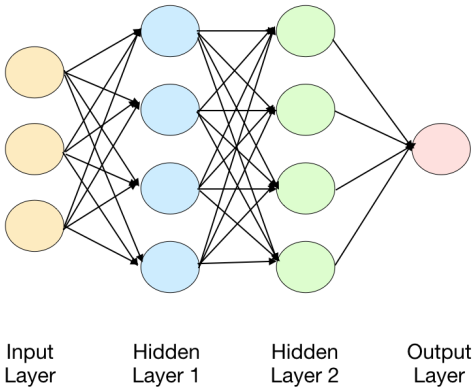

Figure 2.2. A simple artificial neural network with one input layer, two hidden layers and one output layer.

recognition task as shown in Figure 2.3. The first layer is our input image, which we call input layer. The first hidden layer could learn simple edges. The second hidden layer takes the output from the first hidden layer and learns stuff like corners and contours (from edges). Similarly, higher layers could learn more complex objects. The last layer will output our classification. Such deep neural networks need a lot of data for training and the training process usually takes several days or even weeks on a modern computer.

There are different kinds of neural networks. The most recognized ones are convolutional neural network (CNN) [80] and recurrent neural network (RNN) [113]. CNNs are widely used in tasks that involve images or videos processing [77]. RNNs are often used for sequence to sequence modeling like natural language processing or time series modeling. Since I will mainly use RNNs in this study, I will give a brief description of the most common RNNs. For readers who are interested in CNNs or other deep learning models could refer to the deep learning book from Goodfellow *et al.* [47].

2.1.1. Recurrent Neural Networks

We call neural networks like the one shown in Figure 2.2 feedforward networks. Because information goes one way from the input layer to the output layer. Recurrent neural network

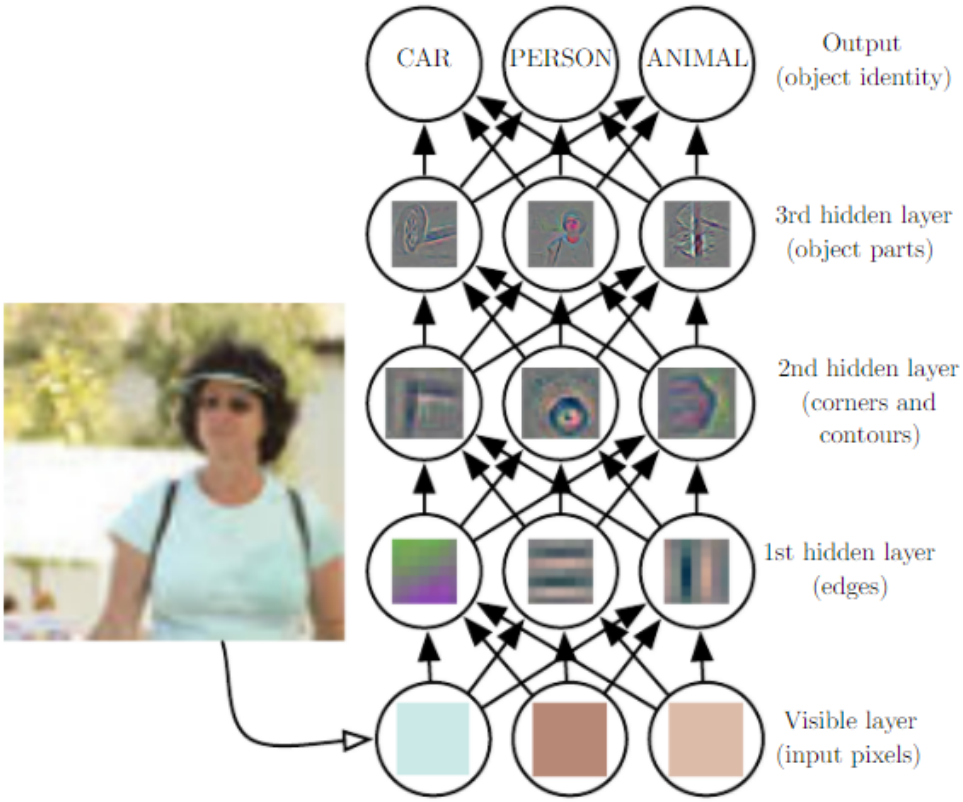

Figure 2.3. A neural network model learns hierarchical representation [47].

(RNN), on the other hand, has a loop which allows information to be passed from one step to the next. This is useful, because it allows access to previous information for the current computation, which is usually required for tasks like machine translation. Figure 2.4 shows an unrolled recurrent neural network. x_t is the input for time step t. h_t is the hidden state. The hidden state is calculated using the following equation:

$$h_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.3}$$

where σ is the activation function, W_i and b_i are learnable parameters. Usually the hidden state h_t will be sent to another output layer to get the final predictions.

$$y_t = \sigma(W_o \cdot h_t + b_o) \tag{2.4}$$

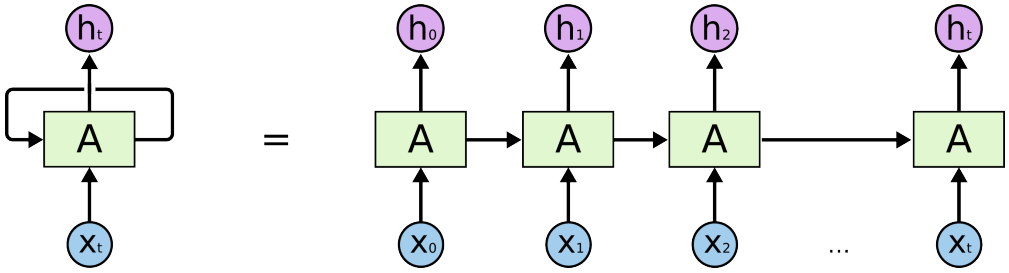

Figure 2.4. An unrolled recurrent neural network [6].

As we can see, the recurrent neural network takes both the current time step input x_t as well as the previous hidden state h_{t-1}. It is believed this hidden state h_t could accumulate knowledge across a long distance. However, the vanilla RNN usually does not work very well in practice, especially when the input consists of data that across long distance. In practice, we usually use variants of the vanilla recurrent neural network, including Long Short Term Memory (LSTM) [58], Gated Recurrent Unit (GRU) [22], etc. The architecture of LSTM is shown in Figure 2.5. An LSTM unit consists of the following parts:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.5}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.6}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.7}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.8}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.9}$$

$$h_t = o_t * \tanh(C_t) \tag{2.10}$$

where σ refers to a logistic (sigmoid) function, $\cdot$ refers to dot products, $*$ refers to element-wise vector multiplication, and $[,]$ refers to vector concatenation. Different from the vanilla recurrent cell, the LSTM has a memory cell c_t as well as the hidden state h_t. The assumption is all useful information will be stored in this memory cell c_t. And there are three gates called

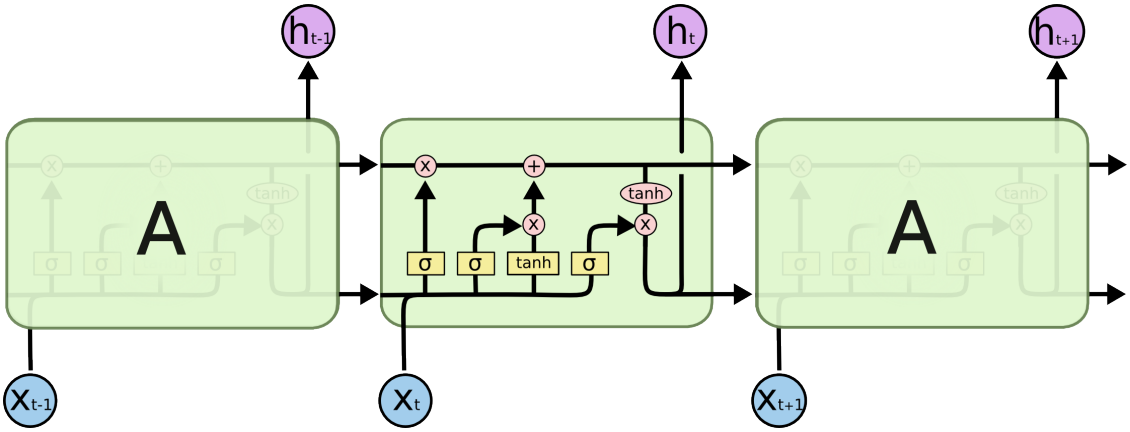

Figure 2.5. Long Short Term Memory (LSTM) network [6].

input gate i_t, forget gate f_t and output gate o_t that decide which information should be stored into this cell and which information to erase. The new hidden state h_t is generated using equation 2.10. The above equations will be used for visualization and help us understand the inner workings of Deep Knowledge Tracing [107] in the next chapter.

Recurrent neural networks are usually trained using a variant of Backpropogation called Backpropogation through time (BPTT) [134]. RNNs face unique challenges compared with feedforward neural networks. Gradients could easily explode or vanish when the input sequence is long. Truncated BPTT or gradients clipping are usually used for RNNs to mitigate these issues [99].

2.1.2. Memory Augmented Networks

Recurrent neural networks are known to have problems learning dependency of knowledge across long time steps. Therefore, much research has been conducted trying to solve this issue. Memory augmented neural networks are one such attempt. The main idea of memory augmented neural network is to use an external storage (memory) to keep information across long time steps. However, we will see that the functionality of this external memory is very similar to the cell c in LSTM, especially when recurrent structures are used. Figure 2.6 shows the End-to-End Memory Network [123] that analyzes a question and short answer task. The

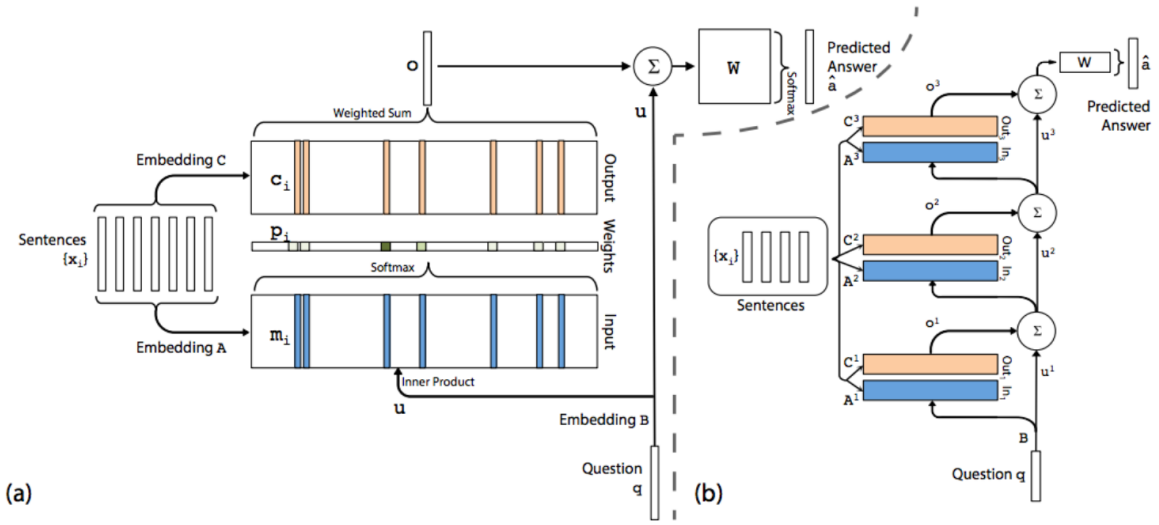

Figure 2.6. End to End Memory Network [123].

short answer sentences are embedded using matrices **A** and **C** to the input and output space separately. The asked question, q, is also embedded using an embedding matrix **B**. An inner product of the embedded question and the input space is used to produce weights, as shown in eq (2.11).

$$w(i) = \frac{f(d(q, M(i)))}{\sum_i^N f(d(q, M(i)))} \tag{2.11}$$

$$r = \sum_i^N w(i)M(i) \tag{2.12}$$

A weighted sum from the output space is used to predict an answer (equation 2.12), which may be further processed using a fully connected layer. The above process consists of one hop. Multiple hops can be used to process multiple questions in sequence, thus the architecture is recurrent. This End to End Memory Network is differentiable, therefore can be trained using gradient descent algorithms. This type of memory network is essentially learning the embedding matrices only and there is no mechanism to 'write' into memory (as described below).

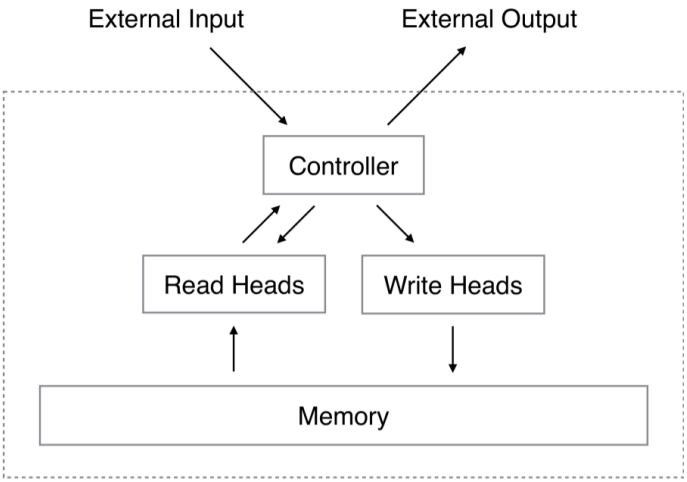

Figure 2.7. Neural Turing Machine [50].

Another type of memory network is Neural Turing Machine (NTM) [50]. Figure 2.7 gives the overall architecture of NTM. There are mainly two parts in this architecture: the controller and the memory. The controller is typically a recurrent neural network or feedforward network. The controller emits outputs that adjust 'read' and 'write' heads that map outputs to different parameters such as a weighting vector. The heads are typically fully connected layers. NTM supports explicitly writing to the memory:

$$M_t = (1 - R_t)M_{t-1} + A_t \tag{2.13}$$

where R_t is the erase matrix and A_t is the addition matrix, both are produced by the heads. NTM is also end to end differentiable.

However, both of these memory augmented networks have one common issue—they all use soft attention. In other words, each read and write is applied to the whole memory, which is not a scalable solution. For example, increasing the memory size will increase the sparsity of information, making it difficult to train and reducing the benefits of large memory. Moreover, writing to the whole memory could also overwrite previously important

information. To solve this problem, several methods have been proposed. Sparse addressing memory (SAM) and reinforcement learning neural turing machine [110,147] are two popular proposed solutions. SAM could run 1000x faster and 3000x less physical memory than non sparse models. Besides, it can scale to tasks requiring 100,000s of time steps and memories. There are at least two significant benefits of the introduction of sparse addressing. First, it decouples memory size from computation. Second, information can be selectively written to specific locations in memory, avoiding being repeatedly overwritten, which could potentially improve performance.

2.2. Neural Architecture Search

A lot of successful deep neural network architectures are developed by experienced experts through trial and errors. Apparently, this process is tedious and not very efficient. Neural Architecture Search (NAS) [84, 105, 152] aims to find good architectures for some specific task automatically and has gained more and more attention in the past few years. It has been shown that some architectures found through NAS have achieved better performance than those developed manually by human experts [41]. We can think of NAS as a discrete optimization problem, thus any discrete optimization algorithms could be used for NAS. For instance, Evolutionary algorithms [9]. Evolutionary algorithm is one of those earliest used for NAS. The idea is to represent the neural network architecture with a fixed size vector. For example, the first element in this vector could be the type of the first layer (conv layer, max pooling layer, etc). Off springs are generated using mutations of the parents vectors. The generated architectures are evaluated based on some metrics (accuracy on a validation set). This process keeps going until we have found some good architectures or we hit the maximum number of iterations. If we look more closely, we could further decompose the model finding problem into architecture search and parameters optimization. In such cases, Evolutionary algorithms are usually used together with gradient based methods, in which Evolutionary algorithms are used for finding the architecture, and gradient based methods are used to optimize the weights.

Other techniques used for NAS include Reinforcement Learning (RL) [100, 105, 152], gradient based methods [39, 115, 131], sequential model based optimization (SMBO) [84], etc. For a comprehensive survey of NAS, readers could refer to this survey [41]. In this thesis, I mainly focus on NAS using reinforcement learning and sequential model based optimization.

2.2.1. NAS using Reinforcement Learning

Reinforcement learning (RL) [124] deals with the problem of an agent interacting with its environment. The agent can take a series of actions. For each action, there might or might not be a reward. The agent also has access to the current state of the environment. The goal of RL is to come up with a strategy of taking actions to maximize the accumulated rewards. There are some unique challenges in RL. For example, the reward might be delayed. Besides, in a series of actions that result in a reward, some actions might actually be harmful. A good introduction of RL is presented in [124]. We can cast neural architecture search as a reinforcement learning problem as shown in Figure 2.8 [152]. There is a controller (recurrent neural network in this case) which acts as our agent. The outputs of this controller are actions which we could further decode into network architectures. In the case of recurrent neural network controller, there will be a sequence of outputs. If our goal is to generate a convolutional neural network, we could interpret the first output as the size of filter, the second output as the number of filters, etc. Then we can build and train this architecture and use the accuracy of this model on a validation set as the reward. This reward will guide the controller to output better architectures. A lot of RL algorithms could be used here like REINFORCE [137], Policy gradient [125], Deep Q network [92], etc.

One tradeoff we have to face is that NAS is usually very computational demanding. This is understandable, since the training of one mediocre size model could take several hours if not several days and NAS has to train and evaluate each generated model. To constrain the NAS problem at a maintainable level, we could either restrict the search space or reduce the training time of a single model. To prevent the search space from going extremely large,

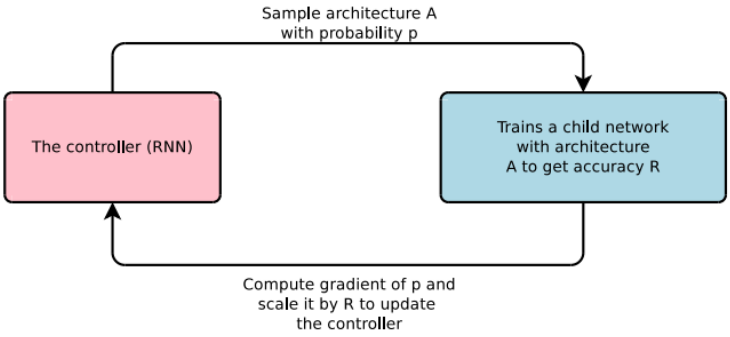

Figure 2.8. Neural architecture search using reinforcement learning [152].

researchers usually search a single unit cell, which could be stacked to form deep models. For instance, we could search a convolve block and then stack several of these blocks to form a deep CNN for image processing. Or we could search a recurrent cell that will be used in a recurrent neural network. Parameters sharing is an effective way of reducing training time [105]. The idea of parameters sharing is that weights are shared among all sub-models. Figure 2.9 shows an example of recurrent cell search. Each node (layer) is fully connected to its previous nodes. The outputs of earlier nodes will be used as inputs to later nodes and a later node could choose the output of any its previous node. We can regard model search as sampling a sub graph from the global graph. A sampled sub graph (model) is indicated as red arrow in this figure. Node 1 is the input node, node 4 and node 5 are leaf nodes and their outputs will be combined to generate the final predictions. A fully connected layer involves the operation of a matrix multiplication and activation function. For the input node, the following computations are performed:

$$c_1^t = \phi(x^t \cdot W_0^{(x,c)} + h_0^{t-1} \cdot W_0^c)$$
$$h_1^t = c_1^t * f_1(x^t \cdot W_0^{(x,h)} + h_0^{t-1} \cdot W_0^h) + (1 - c_1^t) * h_0^{t-1}$$

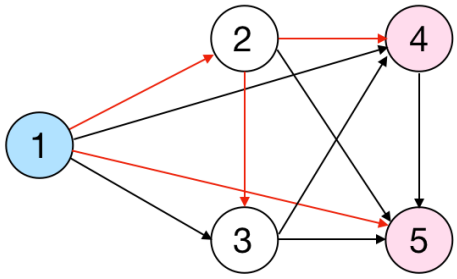

Figure 2.9. Parameters sharing among all sub models. Higher level nodes are fully connected with lower level nodes.

where ϕ and f_1 are both activation functions. For sub sequence layers, the following computations are performed:

$$c_l^t = \phi(h_{j,l}^t \cdot W_{j,l}^c)$$

$$h_l^t = c_l^t * f_l(h_{j,l}^t \cdot W_{j,l}^h) + (1 - c_l^t) * h_{j,l}^t$$

where ϕ and f_l are activation functions, $W_{j,l}$ is the weights from node j to node l. This is similar to the highway recurrent network [151]. At the beginning, all the weights are randomly initialized (other initialization techniques could also be use, like the Kaiming initialization [54] or Xavier initialization [45]). After we trained a sampled model, the corresponding weights changed. When we train subsequence models, these weights will be used as the starting point instead of reinitialization. This weights sharing technique greatly reduce the training time. I also find this sub-model sampling process might be an effective way of mitigating overfitting. I will go into more details about this in Chapter 5 when I present the work for automatic knowledge tracing cell design.

2.2.2. Sequential Model Based Optimization

Sequential model based optimization (SMBO) [84] assumes that the search space could be gradually unfolded from simple to complex. For example, when constructing a recurrent cell, the model becomes more and more complex when more layers are added. Lying in the heart of SMBO is a function that could predict the accuracy of a generated model. This function is

usually called surrogate function. The search starts from the simplest case (for example only one layer), after all possible models are trained. The architectures and their accuracies will be used to train the surrogate model. Then more complex models are unfolded (by adding more layers). When the search space becomes too large to train all possible models, a sampling process is taken. In this case, the surrogate function will be used to predict the accuracies of these models. Then k models are sampled (with higher accuracy, higher the probability to be sampled. This allows exploration of all possible architectures). Then the sampled k models are trained. Their accuracies and architectures are then used again to update the surrogate function. Thus, it is an alternative process. Compared with reinforcement learning, SMBO is simpler and more stable. The NAS problem shown in Figure 2.9 could also be solved using SMBO instead of reinforcement learning. Because the sampled architectures could be encoded as a sequence of list like $[[p_1, a_1], [p_2, a_2], ...[p_i, a_i]]$, where p_i stands for previous node and a_i stands for activation function, we could use a recurrent neural network as our surrogate function. SMBO has been proved to produce as well as good architectures while at the same time reducing the exploration space [104]. In chapter 5, I will dive into the details about how to use SMBO for multimodal fusion and architecture search.

2.3. Evaluation of Deep Neural Networks

After building our model, the next step is to evaluate its performance. For machine learning models, a key insight is that we are not interested in the performance of our model on the dataset that we used for training (training set), but the performance on unseen data. Using a complex model with enough parameters, we could get zero error in the training data. But this model's performance on unseen data will be bad. This is what we call the problem of overfitting. We build machine learning models for prediction. In other words, we care about the model's generalization ability. Thus, a common practice is to split the dataset in hand into two parts: training set and testing set. We train our model using the training set and report the performance on the testing set. This method is also called holdout method. However, one problem about the holdout method is that it is very sensitive to how we split

the dataset. The distributions of samples with different classes might be very different in the training set and testing set. A better way is to use cross validation.

2.3.1. Cross Validation

The idea of K fold cross validation is to split the dataset into K folds. We use K-1 folds for training and the remaining one fold for testing. This process is repeated K times to make sure all folds have been used for testing. Figure 2.10 shows an example of 10 fold cross validation. The final score is averaged across all folds. There is an improved version of cross validation called stratified cross validation, in which the class proportions are the same for all folds. Stratified cross validation usually yield better estimates. If we set K=N, where N is the number of samples in the dataset, this special case is called leave one out cross validation (LOOCV). LOOCV is often used when the dataset is small. There is a similar concept called leave one subject out cross validation (LOSOCV), in which N-1 subjects' data are used for training, one subject's data is used for testing. In LOSOCV, there are usually more than one sample from one subject. LOSOCV could be used to prevent one model from learning subject specific information, thus is often used in the context of medical diagnosis.

Now, let's consider a more complicated situation, **hyperparameters selection**. Models like deep neural networks usually involve a lot of hyperparameters like batch size, learning rate, number of neurons for each layer, etc. Unlike the weights associated with each neuron that could be learnt, the values of these hyperparameters need to be decided beforehand. The selection of these hyperparameters might have a big impact on the performance of the model. If we use the testing set to select our hyperparameters, that is we try different combinations of hyperparameters and choose the best one. We are overfitting our testing set. Because deep neural networks might have millions of parameters, it is very easy for these models to adjust parameters to remember the noise in the dataset. If we use the model tuned on the testing set for future predictions. The performance might be disappointing. Thus, one solution is to use a separate dataset for validation. We use the validation set for hyperparameters search. In the case of holdout method, we could split the original dataset

Figure 2.10. 10 fold cross validation, reported performance are averaged across 10 folds. Image source from [4].

into three parts: training set, validation set and testing set. In the case of cross validation, we could use the technique called nested cross validation. That is, for the K-1 folds for training, we apply another cross validation, further partition this training set into a smaller training set and validation set. Thus, we are using the inner loop for model selection and the outer loop for model evaluation. However, in practice, it is not uncommon to only report the cross validation score.

2.3.2. Statistical Significance Test

Statistical significance test is used to help quantify whether a result is due to chance. To compare the performance of two models, z-score is the most straightforward option. If we consider each prediction as a Bernoulli trial, then the number of correct predictions X follows a Binomial distribution $X \sim Bin(m|N, u)$, where N is the total tries and m is the correct prediction, u is the mean. When N is large enough, we could use Normal approximation to get the z-score:

$$z = \frac{ACC_1 - ACC_2}{\sqrt{\sigma_1^2 + \sigma_2^2}}$$

where ACC_1 and ACC_2 are the accuracies of two different models on two independent testing set. σ_1 and σ_2 are accuracy variances and could be calculated using the following equation:

$$\sigma^2 = \frac{ACC(1 - ACC)}{n}$$

Thus, if our p value is below some threshold (For example, 0.05), this means our null hypothesis claim is very unlikely. Then we accept our alternative hypothesis that the result is not due to chance. However, the z score method usually does not work very well in practice [32].

A more preferred method is to use McNemar's test [91], which focuses on the distributions of predictions. The test is applied to a 2x2 table as shown below:

	Model 2 positive	Model 2 negative	Row total
Model 1 positive	a	b	a+b
Model 1 negative	c	d	c+d
Column total	a+c	b+d	n

The null hypothesis and alternative hypothesis are:

$$H_0 : p_b = p_c$$

$$H_1 : p_b \neq p_c$$

The McNemar's test statistic is:

$$\chi^2 = \frac{(b - c)^2}{b + c}$$

χ has a chi-squared distribution. If the results are significant, it will reject the null hypothesis, in favor of the alternative hypothesis. Thus, the predictions from these two models are considered different. For deeper discussions of statistical significance testing and their

applications in machine learning, readers could refer to [12, 29, 32].

However, it is not very common to find research papers from the machine learning community use significance test. On one hand, a lot of significance test techniques have their limitations and might not be suitable for machine learning models [32]. On the other hand, the datasets involved in machine learning are usually very large and the training process might take several days if not weeks, this is especially true for deep neural network models. Thus, it is impractical to run this training process N time to get N numbers for the requirements of some significance tests. I provide McNemar's test in this dissertation, despite its limitations.

2.4. Deep Neural Network Based Models for Knowledge Tracing

I have introduced the basic ideas of deep neural networks and some popular architectures like RNNs and memory networks in the previous sections. Now, I will describe two popular deep neural network based knowledge tracing models. Deep Knowlegde Tracing (DKT) [107] and Dynamic Key-Value Memory Network (DKVMN) [149]. These two models have achieved better performance compared with conventional models like BKT and PFA. These two models also inspired a lot other research utilizing deep learning techniques in the learning science community.

2.4.1. Deep Knowledge Tracing

A knowledge tracing model uses historical activities to predict the future performance. In the simplest case, the historical activities is a sequence of correct/incorrect responses. Thus, it is natural to use recurrent neural network for knowledge tracing. Piech *et al.* [107] first proposed Deep Knowledge Tracing (DKT) model as shown in Figure 2.11. As we can see it is standard recurrent neural network. The RNN cell used, which is not shown in the figure could be the standard cell or LSTM. The input to this model is a one hot encoding of the skill id and the correctness of the problem. For example, if there are M skills, the size of the input vector will be $2 * M$. The authors tried other options, like encoding the skill id

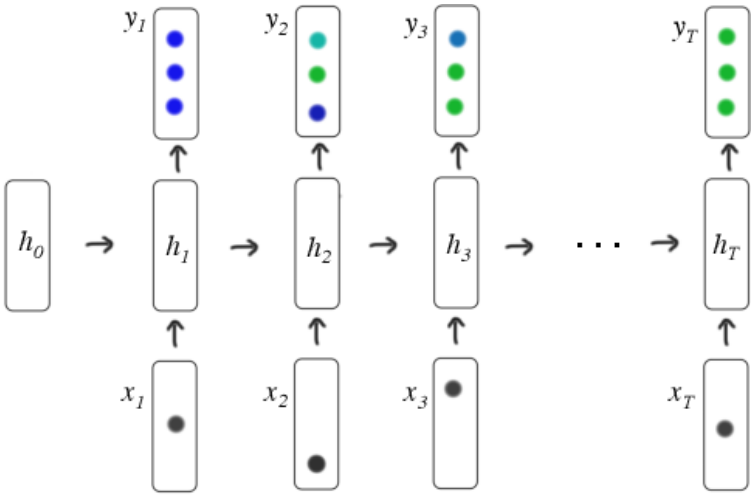

Figure 2.11. Deep Knowledge Tracing Model [107].

and correctness separately, but unsuccessful. The output layer of the model is a vector of probabilities of getting each skill correct. For example, the assistment 09-10 dataset has 124 skills. So the size of the output vector is 124, with each element corresponds to each skill. However, this output vector is not trained as a whole. For each student, the DKT models take the combination of the skill id and correctness from the previous problem to predict the next problem. The next problem id will be used to pick the corresponding element in the output vector, based on which the cost function is created. Also, the DKT model does not distinguish different students.

One argument of why DKT is so successful is because the hidden state h_t could capture the accumulated learning of a student and the relations between different skills. Even though each element of the output vector is designed to represent the probability of getting each skill correct. I found that these skills usually change in the same direction at the same time [33]. In other words, either most of these skills are correlated to each other, or the DKT model is only learning an overall 'ability' model instead of tracking each skill separately. I will go into the details in the next chapter.

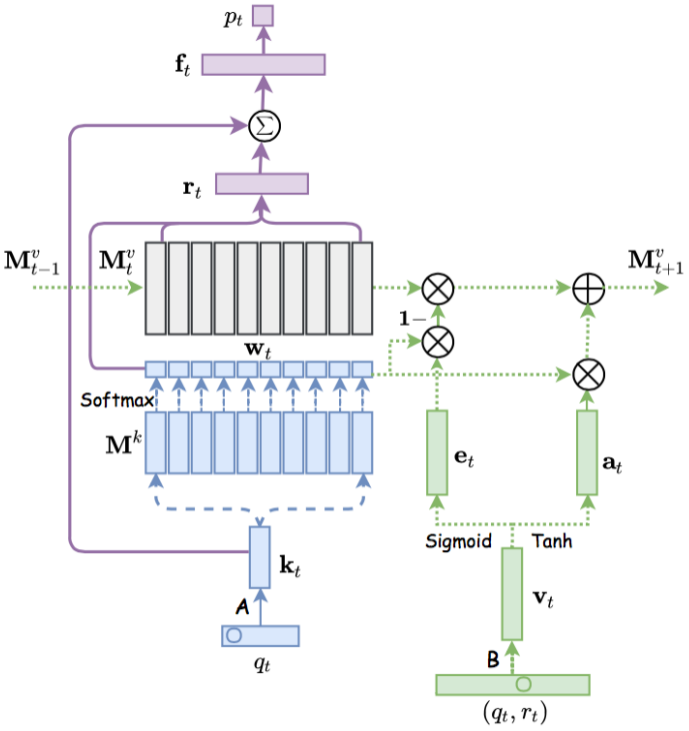

Figure 2.12. Dynamic Key-Value Memory Network for knowledge tracing [149].

2.4.2. Dynamic Key-Value Memory Network for Knowledge Tracing

Dynamic key-value memory network for knowledge tracing (DKVMN) [149] is based on the work from Weston *et al.* [135]. The architecture of DKVMN is shown in Figure 2.12. DKVMN has one static key memory M^k, which is the embeddings of all skills. The content of M^k does not change with time. DKVMN also has one dynamic value memory M^v_t for storing the current mastery level of corresponding skills. The content of M^v_t is updated after each response. There are two stages involved in the DKVMN model. In the read stage, a query skill q_t is first embedded to get k_t. Then a correlation weight is calculated:

$$w_t(i) = Softmax(k_t^T M^k(i))$$

The mastery level of skill q is thus calculated as follows:

$$\mathbf{r}_t = \sum w_t(i) M_t^v(i)$$

The authors concatenate the query skill q_t with r_t to get the final output p_t arguing that the difficult level of each skill might be different. The second stage is to update the memory network M_t^v. The embedding of the combination of the skill query q_t and the actual correctness r_t is used to create an erase vector $\mathbf{e}_t$ and an add vector $\mathbf{a}_t$. The new value matrix is updated using the following equations:

$$\tilde{M}_t^v(i) = M_{t-1}^v(i)[\mathbf{1} - w_t(i)\mathbf{e}_t]$$

$$M_t^v(i) = \tilde{M}_t^v(i) + w_t(i)\mathbf{a}_t$$

However, I tested DKVMN model on two different datasets and did not see improvements compared with DKT, though one may argue that DKVMN is more interpretable than DKT. I will compare my proposed models with DKT and DKVMN in chapter 5 and 6. In the next chapter, I will first discuss some limitations of these deep neural network based knowledge tracing models.

Chapter 3

TOWARDS INTERPRETABLE DEEP NEURAL NETWORK MODELS FOR STUDENT RESPONSE MODELING

How one deep neural network adapts its weights during training? Based on what strategies a decision is made? Is the learnt representation meaningful? With the wide success of deep neural networks, the requests for interpretable models are also increasing. These questions need to be seriously considered before deployment in critical domains like medical diagnosis, planning and control. But why interpretability is important to use? What is the exact definition of interpretable models?

Motivations of interpretability differ across research, but many argue that interpretability is a necessary condition for *trust* [71]. Given an input, can we trust the predictions? Machine learning models are trained using a training set. Our assumption is this training set will be a good representative of the real world data (classes distribution for example). However, what will happen if we have an input which is from a totally different distribution? For example, if we have built a model to classify different species of dogs, what will happen if we give this model a cat? *trust* is a subjective word, but it reflects some truth that we want our model to take actions that human beings will take. If the model behaves similarly as human beings when making decisions, we could relinquish control. In the dog classification case, humans will express great uncertainty when given an image of cat. Thus, we hope our model could also output great uncertainty (either output an uncertainty score together with the prediction, or use other mechanisms). This is a typical example of out of distribution problem. We could *trust* a model if it tends to make mistakes on inputs that human beings will also make mistakes and is accurate on inputs that human beings are accurate. However, this is often not guaranteed, especially for deep neural network models. Deep neural network models could be easily fooled [97]. For example, we could create some adversarial images

just by changing some pixel values to fool a convolutional neural network [126].

For the second question, unfortunately, as discussed in [18, 83] there is also no unique answer and interpretability is not a monolithic concept. May be it is more appropriate to ask what properties an interpretable model should have, or what extra information we could get from interpretable models? In Section 3.1, I will discuss the main properties an interpretable model should have. For a more detailed discussion, readers could refer to these works [18,83]. In Section 3.2, I use visualization to open the black box of DKT model, allowing a better understanding of inner workings.

3.1. Interpretable Deep Learning

What extra information we could get for these interpretable models compared with those 'uninterpretable' models? In this section, I discuss the main properties an interpretable model should have, taking the categorization from [83].

3.1.1. Transparency

Transparency is the opposite of black-boxness. There are several aspects about transparency [83]. **Simulatability** means take the input and all the parameters of the model, we could calculate the output manually in a reasonable time. If we tweak the parameters a little bit, we will have a good estimate about what will happen. In other words, we have a clear understanding of each computation step involved. However, this definition is vague and it seems simple models are equivalent to transparent models. **Decomposability** requires an intuitive explanation of the parameters and features. Consider the case of a linear model, a large weight usually indicates the importance of the corresponding feature. Both Bayesian Knowledge Tracing (BKT) [25] and Performance Factors Analysis (PFA) [102] models are designed in a way that each parameter has a semantic meaning. For example, the guess and slip parameter in the BKT model reflect the probability that a student could have a correct guess and make a mistake despite of mastery of a skill, respectively. BKT attempts to explicitly model these parameters and use them to infer a binary set of skills as mas-

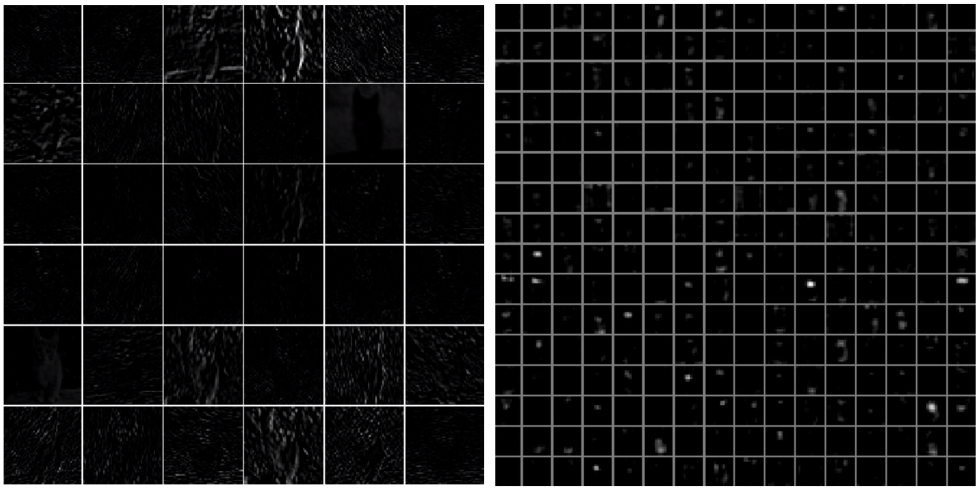

Figure 3.1. Activation of filter maps for first and 5th CONV layer of AlexNet when looking at a cat. With each square corresponds to one filter map [1].

tered or not mastered. However, more research works are needed to refine these notions of interpretability.

Both **Simulatability** and **Decomposability** are challenging for deep neural network models. A mediocre sized deep neural network could easily have millions of parameters, thus to calculate the output manually is nearly impossible. Besides, try to explain the meaning of each parameter is also impractical. Thus most of interpretable deep learning works focus on Post-hoc interpretability.

3.1.2. Post-hoc Interpretability

For deep neural networks, we usually take different approaches to *interpret* the learnt model. These approaches all try to extract extra information after one model has been trained. Thus, we call this kind of intrepretability Post-hoc interpretability. Krening *et al.* choose to train a separate model to output text explanations about the decisions [75]. However, this requires additional labeled data and we are adding another black-box in order to explain the original one. In computer vision community, visualization is a common tool used to understand convolutional neural networks. One option is to visualize the activations

of filters for different layers. Figure 3.1 shows the activations of filter maps for the first and 5th CONV layer of AlexNet when the input is an image of cat. Each square corresponds to one filter map. When we go deeper with more filters, the activations become quite sparse. We can also choose to visualize the weights [1]. However, these kinds of visualization is still not straightforward to explain.

We could also take a totally different approach. After trained the model, we fix the weights, but adjust the input image to enhance the activations of some class. By modifying the input image, we have a visual clue about what features the trained model is looking for. This is the technique behind Google's DeepDream Project [3]. Instead of modifying one image, we could also prepare a large dataset of images. Sending all these images through the trained network and rank them based on the activations of some neuron. We could check these top ranked images to have a better understanding of what this neural network is looking for [44].

Given one image, to elucidate what spatial area our classifiers are concentrating upon to classify an image as dog or cat, we could use the Gradient-weighted Class Activation Mapping (Grad-CAM) visualization technique [118]. Grad-CAM starts by calculating the gradients of the score for class c (before the softmax) with respect to the feature maps of the last convolutional layer. The gradients are then global average-pooled as weights. By inspecting these weighted activation maps, we can see which portions of the image have significant influence in classification.

3.2. Open the Black Box of DKT Model

The mechanisms of DKT are not well understood by the research community. That is, none of the parameters are mapped to a semantically meaningful measure or how the decisions are made for different skills, which diminishes our ability to understand the DKT model. There have been some attempts to explain why DKT works well but these studies treat DKT model more like a black box, without studying the state space that underpins the recurrent neural network [68, 140]. However, sometimes, a still understanding of how the

model works is as important as the performance. In this section, I take a Post-hoc approach, trying to use visualization to understand the inner workings of DKT model. My goal of this study is to inspire more research towards interpretable neural network based models for knowledge tracing and also provide directions for future research.

3.2.1. Experiment Setup

To investigate the DKT model, I perform a number of analyses based upon the activations within the recurrent neural network. I also explore different training protocols and clustering of the activations to help elucidate what is learned by the DKT model. In these analyses, I use the "ASSISTmentsData 2009-2010(b) dataset" which is created by Xiong *et al.* after removing duplicates [140]. Like Xiong *et al.*, I also use LSTM unit in this study. The mathematical elements that comprise each unit are given from equation 2.5 to 2.10.

For visualization purposes, I log the six intermediate outputs for each input during testing and concatenate these outputs into a single "activation" vector, $a_t = [f_t, i_t, \tilde{C}_t, C_t, o_t, h_t]$. I want to see how this activation vector changes when the model sees more and more inputs. In the DKT model, the hidden state of RNN, h_t is connected to an output layer y_t, which is a vector whose size equals to the number of skills. We can interpret each element in y_t as an estimate that the student would answer a question from each skill correctly, with larger positive number denoting that the student is more likely to answer correctly and more negative numbers denoting that the student is unlikely to respond correctly. Thus, a student who had mastered all skills would ideally obtain an y_t of all ones. A student who had mastered none of the skills would ideally obtain an y_t of all negative ones. A sigmoid function could be applied on top of these logits to convert them to probabilities between 0 and 1.

Deep neural networks usually work in high dimensional space and are difficult to visualize. The activation vector a_t has more than one thousand dimensions. Thus, I need to use dimensionality reduction techniques to identify clusters. T-distributed Stochastic Neighbor Embedding (t-SNE) [87] is a nonlinear dimensionality reduction technique that well suited

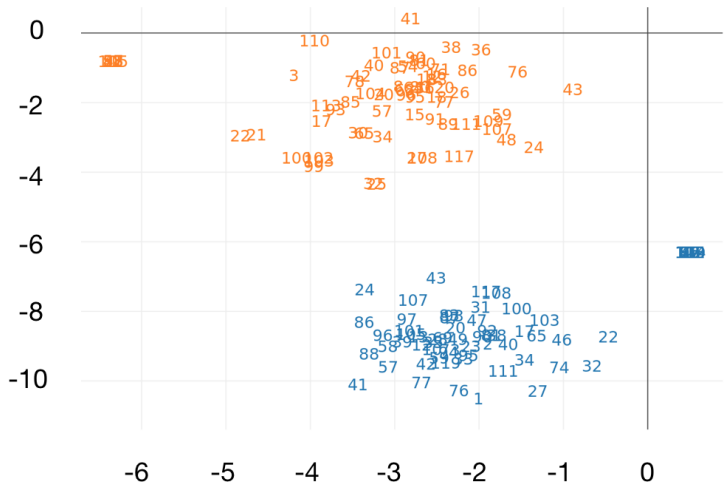

Figure 3.2. First two components of T-SNE of the activation vector for first time step inputs. Numbers are skill identifiers, blue for correct input, orange for incorrect input.

for this purpose. Figure 3.2 plots the first two reduced components (using t-SNE [87]) of the activation vector, a_t, at the first time step ($t = 0$) for a number of different students. The numbers in the plot are skill identifiers. I use color blue to denote a correct response and the color orange to denote an incorrect response. From reducing the dimensionality of the a_t vector for each student, we can see that the activations show a distinct clustering between whether the questions were answered correctly or incorrectly. We might expect to observe sub-clusters of the skill identifiers within each of the two clusters but we do not. This observation supports the hypothesis that correct and incorrect responses are more important for the DKT model than skill identifiers. However, perhaps this lack of sub-clusters is inevitable because we are only visualizing the activations after one time step—this motivates the analysis in the next section.

3.2.2. Prediction Vector Changes and Relation Among Skills

In this section, I try to understand how the prediction vector of one student changes as this student answers more questions from the question bank. This will help us understand the direct impacts an input has on the prediction vector. Figure 3.3 plots the prediction vector

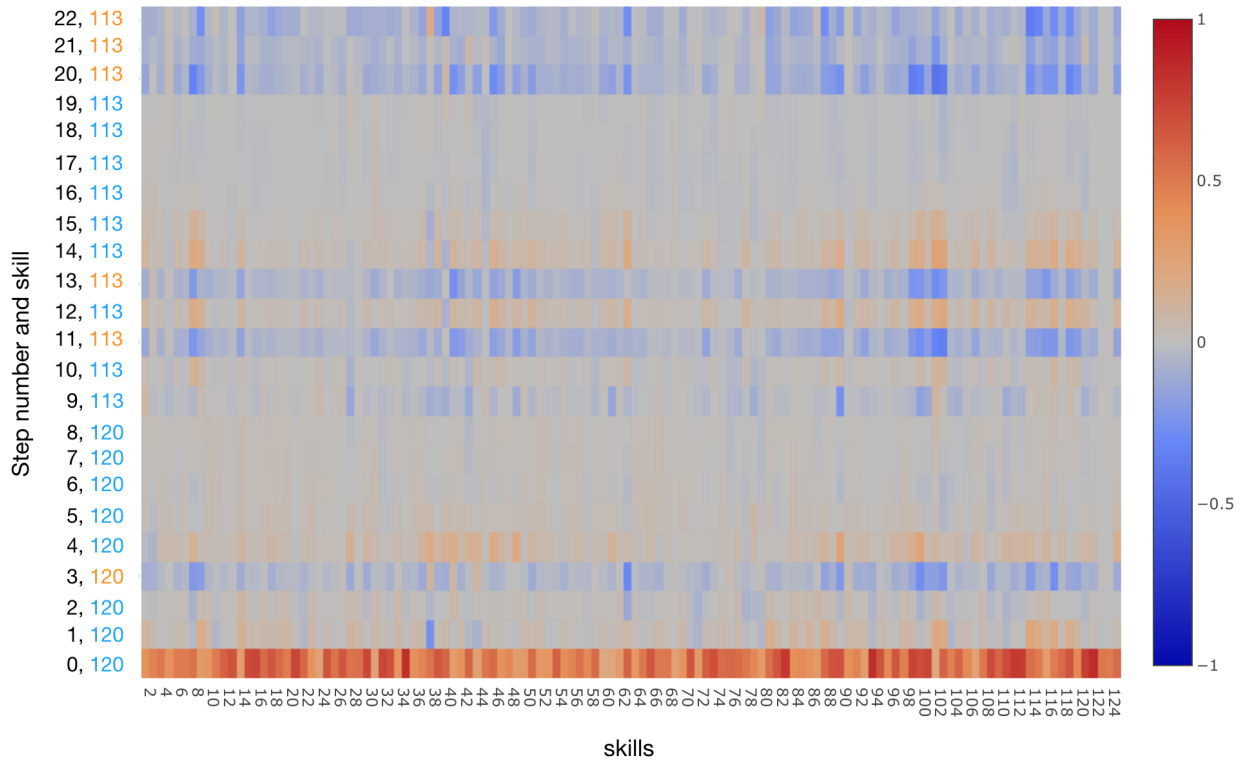

Figure 3.3. The prediction changes for one student, 23 steps, correct input is marked blue, incorrect input is marked orange.

difference (current prediction vector - previous prediction vector) for each question response from one particular student (steps are displayed vertically and can be read sequentially from bottom to top). The horizontal axis denotes the skill identifier and the color of the boxes in the heatmap denote the change in the output vector y_t. The first row in the heatmap (bottom) is the starting values for y_t for the first input. As we can see, if the student answers correctly, most of the values increase (warm color). On the other hand, when an incorrect response occurs, most of the predictions decreases (cold color). This makes intuitive sense. We expect a number of skills to be related so correct responses should add value and incorrect responses should subtract value. We can further observe that changes in the y_t vector diminish if the student correctly or incorrectly answers a question with the same skill several times repeatedly. For example, observe from step 14 to step 19, where the student

correctly answers questions associated with skill #113. Eventually the changes in y_t come to a steady state. However, occasionally, we can also notice, a correct response will result in decreases in the prediction vector (observe step 9). This behavior is difficult to justify from our experience, as correctly answering a question should not decrease the mastery level of other skills. Yeung *et al.* have similar findings when investigating single skills [145]. Observe also that step 9 coincides with a transition in skills being answered (from skill #120 to #113). Even so, it is curious that switching from one skill to another would decrease values in y_t even when the response is correct. From this observation, one potential way to improve the DKT model could be adding punishment for such unexpected behaviors (for example, in the loss function of the recurrent network).

3.2.3. Using Synthetic Data

From the previous section analysis, we see from step 14 to step 19, the student correctly answers questions associated with skill #113 and the changes in y_t diminish—perhaps an indication that the vector is converging. Also, from Figure 3.3, we see that for each correct input, most of the elements of y_t increase by some margin, regardless of the input skill. To have a better understanding of this convergence behavior, I simulate how the DKT model would respond to an *Oracle Student*, which will always answer each skill correctly. I simulate how the model responds to the *Oracle Student* correctly answering 100 questions from one skill. I repeat this for three randomly selected skills.

I plot the convergence of each skill using the activation vector a_t reduced to a two-dimensional plot using t-SNE (Figure 3.4 Left). The randomly chosen skills are #7. #8, and #24. As we can see, each of the three skills starts from a different location in the 2-D space. However, they all converge to near the same location in space. In other words, it seems DKT is learning one "oracle state" and this state can be reached by practicing any skill repeatedly, regardless of the skill chosen. I verified this observation with a number of other skills (not shown) and find this behavior is consistent. Therefore, I hypothesize that DKT is learning a 'student ability' model, rather than a 'per skill' model like BKT. To make

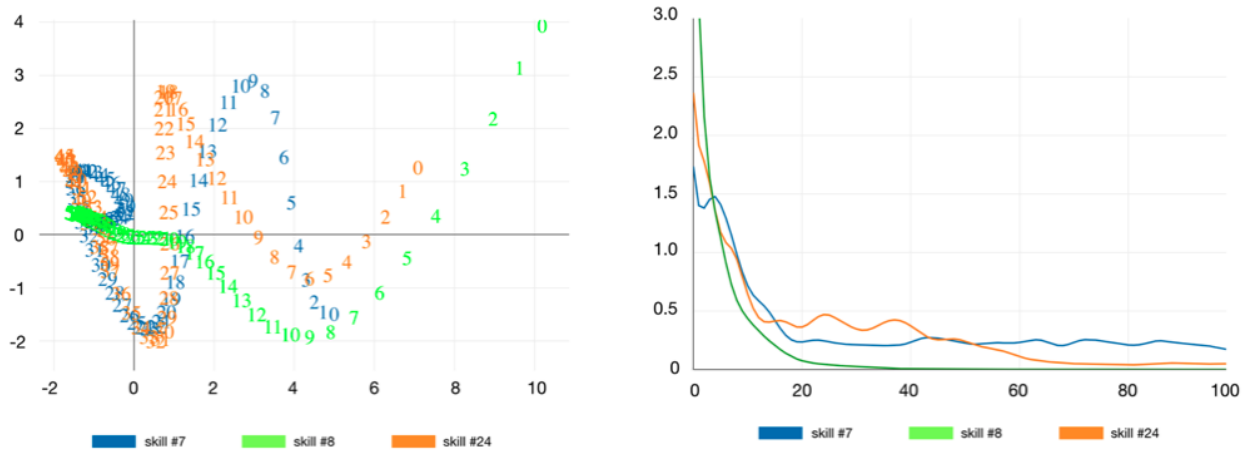

Figure 3.4. **Left:** Activation vector changes for 100 continuous correctness of randomly picked 3 skills **Right:** Activation vector difference of randomly picked 3 skills through time.

this observation more concrete, in Figure 3.4 right, I plot the euclidean distance between the current time step activation vector, a_t, and the previous activations, a_{t-1}, we can see the difference becomes increasingly small after 20 steps. Moreover, the euclidean distance between each activation vector learned from each skill becomes extremely small, supporting the observation that not only is the y_t output vector converging, but all the activations inside the LSTM network are converging. I find this behavior curious because it means that the DKT model is not remembering what skill was used to converge the network to an 'oracle state.' Remembering the starting skill would be crucial for predicting future performance of the student, yet the DKT model would treat every skill identically. I also analyzed a process where a student always answers responses incorrectly and found there is a similar phenomenon with convergence in an anti-oracle state.

Figure 3.5 shows the skills prediction vector after answering correctly 20 times in a row. We can see the predictions of most skills are above 0.5, regardless of the specific practice skill used by the *Oracle Student*. I argue that the DKT model is not really tracking the mastery level of *each skill*, it is more likely learning an 'ability model' from the responses. Once a student is in this oracle state, DKT will assume that the student will answer most of the questions correctly from any skill. I hypothesize that this behavior could be mitigated by

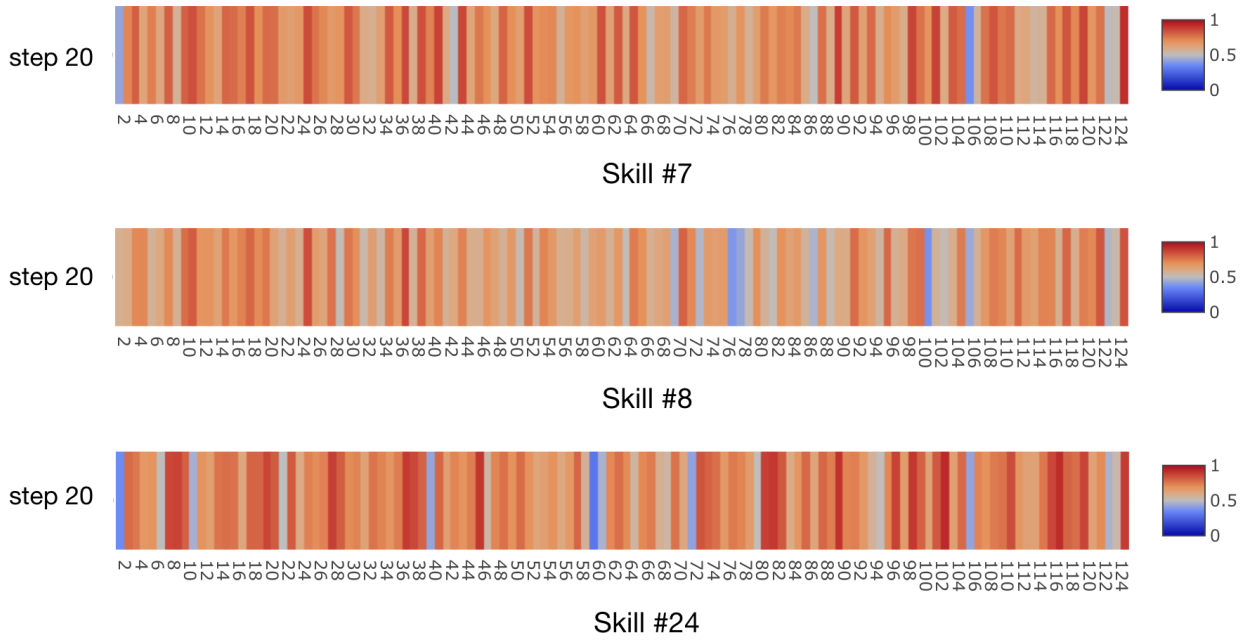

Figure 3.5. Prediction vector after 20 steps for skill #7, #8, #24.

using an "attention" vector during the decoding of the LSTM network [129]. Self attention in recurrent networks decodes the state vectors by taking a weighted sum of the state vectors over a range in the sequence (weights are dynamic based on the state vectors). For DKT, this attention vector could also be dynamically allocated based upon the skills answered in the sequence, which might help facilitate remembering long-term skill dependencies. I will discuss the application of attention mechanisms for knowledge tracing in chapter 6.

3.2.4. Temporal Impact

Recurrent neural networks are typically well suited for tracking relations of inputs in a sequence, especially when the inputs occur near one another in the sequence. However, long range dependencies are more difficult for the network to track [129]. In other words, the predictions of RNN models will be more impacted by recent inputs. For knowledge tracing, this is not a desired characteristic. Consider two scenarios as shown below: For each scenario, the first line is the skill numbers and the second line are responses (1 for correctness and

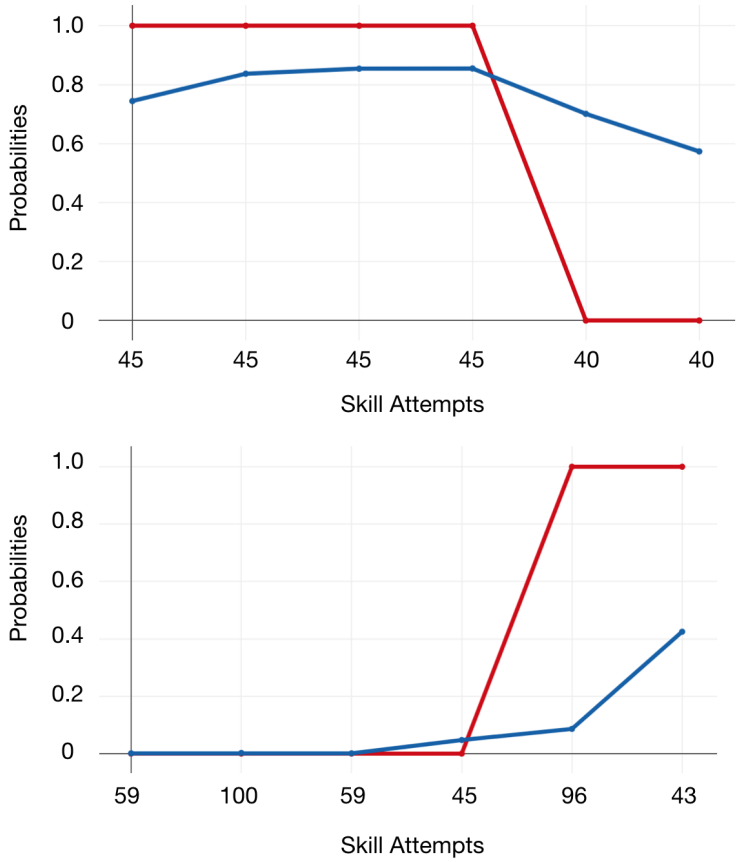

Figure 3.6. DKT predictions from two different students. The blue line is the prediction of correctness from DKT. The red line is the actual response correctness(1 or 0).

0 for incorrectness). Both two scenarios have the same number of attempts for each skill (4 attempts for skill #9, 3 attempts for skill #6 and 2 attempts for skill #24). Also, the ordering of correctness within each skill is the same (*e.g.*, 1, 0, 0, 0 for skill #9).

For models like BKT, there is a separate model for each skill. Thus, the relative order of different skills presented has no influence, as long as the ordering within each skill remains the same. In other words, for each skill the ordering of correct and incorrect attempts remains the same, but different skills can be shuffled into the sequence. For BKT, it will learn the same model from these two scenarios, but it may not be the case for DKT. The DKT model is more likely to predict incorrect response after seeing three incorrect inputs in a row because it is more sensitive to recent inputs in the sequence. This means, for the first

Scenario #1									
Skill ID	6	6	9	9	9	9	**24**	24	6
Correct	1	1	1	0	0	0	0	0	1

Scenario #2									
Skill ID	9	9	9	9	6	6	6	**24**	24
Correct	1	0	0	0	1	1	1	0	0

scenario, first attempt of skill #24 (in bold) will be more likely predicted incorrect because it follows three incorrect responses. For the second scenario, first attempt of skill #24 (in bold) is more likely to be predicted correct. Thus the DKT model might perform differently on the given scenarios.

Figure 3.6 gives two typical excerpts from the real dataset for two students. In the top example, after several correct inputs, the DKT model has a high probability of predicting the next item correct, regardless of the skill (70%). Similarly, in the bottom example, after several incorrect inputs, the DKT model has a low probability of predicting the next item correct (8%), regardless of the skill. That means, if a student has mastered an easy skill previously but then fails three attempts of more difficult exercises, the DKT would predict that the student would also fail the already mastered skill. I am only giving two samples here due to limited space, but this kind of behavior is universal across students, which I will talk more next. Again, I hypothesize that this behavior could be mitigated by using an "attention" vector that allows the DKT to use the whole weighted history as additional inputs.

Khajah *et al.* also alluded to this recency effect in [68]. In this study, I examine this phenomenon in a more quantitative way. I shuffle the dataset in a way that keeps the ordering within each skill the same, but spreads out the responses in the sequence. This change should not change the prediction ability of models like BKT. The results are shown in Table 3.1 and Table 3.2 using standard evaluation criteria for this dataset. All results are based on a five-fold cross validation of the dataset. When comparing DKT on the original dataset to the "spread out" dataset ordering, we see that the relative ordering of skills has

Table 3.1. Area under the ROC curve

	PFA	BKT	DKT	DKT (spread)	DKT (untrained)
09-10 (a)	0.70	0.60	0.81	0.72	0.79
09-10 (b)	0.73	0.63	0.82	0.72	0.79
09-10 (c)	0.73	0.63	0.75	0.71	0.73
14-15	0.69	0.64	0.70	0.67	0.68
KDD	0.71	0.62	0.79	0.76	0.76

Table 3.2. Square of linear correlation (r^2) results

	PFA	BKT	DKT	DKT (spread)	DKT (untrained)
09-10 (a)	0.11	0.04	0.29	0.15	0.25
09-10 (b)	0.14	0.07	0.31	0.14	0.26
09-10 (c)	0.14	0.07	0.18	0.14	0.15
14-15	0.09	0.06	0.10	0.08	0.09
KDD	0.10	0.05	0.21	0.17	0.17

significant negative impact on the performance of the model. From these observations, we see the behaviors of DKT is more like PFA which counts prior frequencies of correct and incorrect attempts other than BKT and the design of the exercises could have a huge impact on the model (For example, the arrangements of easy and hard exercises).

3.2.5. Is the RNN Representation Meaningful?

In a recently published paper, Wieting *et al.* [136] argue that RNNs might not be learning a meaningful state vector from the data. They show that a randomly initialized RNN model (with only W_o and b_o trained) can achieve similar results to models where all parameters are trained. This result is worrying because it may indicate that the RNN performance is due mostly to simply mapping input data to random high dimensional space. Once projected into the random vector space linear classification can perform well because points are more likely

to be separated in a sparse vector space. The actual vector space may not be meaningful. I perform a similar experiment in training the DKT model. I randomly initialize the DKT model and only train the last linear layer (W_o and b_o) that maps the output of LSTM h_t to the skill vector, y_t. As shown in Table 3.1 and Table 3.2, the untrained recurrent network performs similarly to the trained network.

3.2.6. Conclusion

In this chapter, I propose interpretable deep neural networks for knowledge tracing. To achieve transparency is usually hard for deep neural networks, thus most existing approaches try to interpret deep models in a post-hoc way. I visualized the activation vector a_t using dimensionality reduction technique and analyzed the behaviors of DKT through time. I have also analyzed the temporal sequence behavior of DKT using qualitative and quantitative analyses. I find that the DKT model is most likely learning an 'ability' model, rather than tracking each individual skill. Moreover DKT is significantly impacted by the relative ordering of skills presented. I also discover that a randomly initialized DKT with only the final linear layer trained achieves similar results to the fully trained DKT model. In other words, the DKT model performance gains may stem from mapping input sequences into a random high dimensional vector space where linear classification is easier because the space is sparse. This is a worrying conclusion because it means the underlying recurrent representation may not be reliable nor semantically meaningful. Several mitigating measures are suggested, including the use of a loss function that mitigates unwanted behaviors and the use of an attention model to better capture long term skill dependencies.

Chapter 4

UNCERTAINTIES IN STUDENT RESPONSE MODELING

Deep learning models are primarily trained using Backpropagation [113] and once trained, the weights are fixed. Thus, we can think of these models as deterministic functions. For each input x_i, there is always a corresponding output y_i. But what if we have an input x_i that is far from the distribution of the training data, how do we interpret the output? A real world example is from a 2016 car accident that the assisted driving system confused the sky with the bright side of a trailer [67]. Directly targeting this problem is difficult. A more feasible way is to make the model learn some uncertainty level about the prediction associated with the input data in a supervised or unsupervised manner.

In the previous chapter, I proposed interpretable deep neural network models for knowledge tracing and analyzed the behaviors of DKT using visualization. We have learnt that the DKT model may not actually track each skill, but only learns an oracle state. By opening the black box of DKT, I aim to have a better understanding of how decisions are made. In this chapter, I take a different approach and propose a practical way of alleviating the worries caused by the opaqueness of these deep learning based models, which is to model the uncertainty for each prediction. Thus, we could leave decisions to human experts if the uncertainty level is high. To investigate uncertainty modeling in DKT, I first examine a popular way of modeling data dependent uncertainties using Monte Carlo and show how it is insufficient to model variance in data. Second, I provide a way of incorporating uncertainties by regularizing the cross entropy loss function explicitly. At last, I evaluate the proposed method both in three different real datasets and in a more controlled way using synthetic data. Using synthetic data allows us to quantitatively understand the generated uncertainties. The results show that the proposed method provides comparable results to standard deep knowledge tracing models as well as meaningful prediction uncertainties.

4.1. Uncertainties in Deep Learning

In this section I will talk about how to incorporate uncertainties into deep models. I will mainly discuss two types of uncertainties: Epistemic uncertainty and Aleatoric uncertainty. Epistemic uncertainty is also called model uncertainty. It is caused by the fact that there are many models that could have generated the dataset in hand. This kind of uncertainty could be reduced by collecting more data. Aleatoric uncertainty is caused by measurements or observations thus is considered irreducible. In the real world, it can be difficult to differentiate these two types of uncertainties [30], such that hybrid uncertainties consisting of entangled epistemic and aleatoric factors exist. However, in this study, I do not explicitly model hybrid uncertainties, but focus more on the analysis of uncertainties in isolation.

4.1.1. The Uncalibrated Fact of Deep Neural Networks

Before diving into the discussion of two different types of uncertainties, there is still one question that needs to be answered. For a classification task, a deep neural network will output a probability distribution, reflecting the likelihood of different classes. And the class having the largest probability is chosen as the prediction. Why can't we just use this probability as a proxy of our uncertainty level? What exactly does the output of a deep neural network model tell you? From the frequentist perspective of probability theory [62], the output probability from a model reflects the true likelihood. For example, if a diagnosis system detects that a patient has cancer with the probability of 80%, then 80 out of 100 patients with similar symptoms indeed have cancer. If the output from a probabilistic model reflects the true likelihood, then we call this model is calibrated. For a classification task, the output logits from a neural network usually will be sent to a softmax function to get a probability distribution among different classes. One surprising recent finding is that modern deep neural networks are not well calibrated [51]. Figure 4.1 shows the confidence histograms and reliability diagrams for the 5-layer LeNet [79] and 110-layer ResNet [55] on the CIFAR-100 dataset [76]. The average confidence of the ResNet is mucher higher than the average accuracy. Thus, we may not use the output probabilities as a proxy for the measurement of

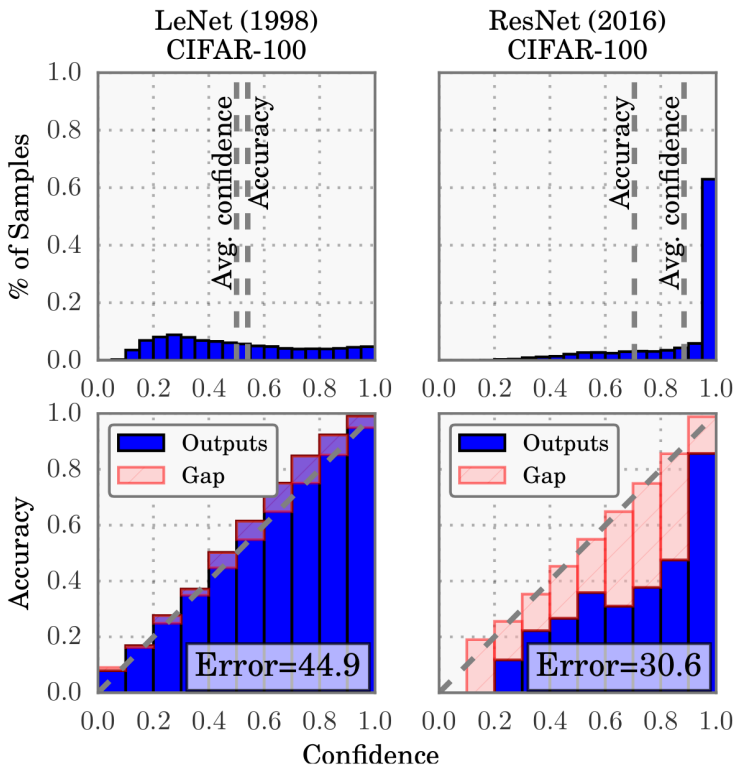

Figure 4.1. Confidence histograms (top) and reliability diagrams (bottom) for a 5-layer LeNet and a 110-layer ResNet on the CIFAR-100 Dataset [51].

uncertainty. For a detailed discussion about the possible causes of uncalibrated outputs and ways to fix that, readers could refer to this work [51].

4.1.2. Epistemic Uncertainty

If we view a neural network as a probabilistic model $P(y|x, \mathbf{w})$. Given training data $\mathcal{D}$ = {(x^i, y^i), i = 1,...,N}, we could get the parameter $\mathbf{w}$ using maximum likelihood estimate (MLE):

$$\mathbf{w}^{MLE} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^{N} \log P(y^i|x^i, \mathbf{w}) \tag{4.1}$$

Regularisation term is not shown here for simplicity. Global optimal is usually not accessible for deep models. Iterative methods like gradient descent could be used to get a

local optimal, such local optimal could usually give satisfying results in practice. Bayesian Deep Learning assumes a distribution over the weights of a neural network. Given the training data $\mathcal{D}$, we could get the posterior distribution of weights $P(\mathbf{w}|\mathcal{D})$. The predictive distribution is given by $P(\hat{y}|\hat{x}) = \mathbb{E}_{P(\mathbf{w}|\mathcal{D})}[P(\hat{y}|\hat{x}, \mathbf{w})]$ for a prediction $\hat{y}$ of a test item $\hat{x}$. There are several works that tried to incorporate uncertainties into deep models [67] [43] [48]. They all tackle this problem using a bayesian approach. However, integrating over all $\mathbf{w}$ is usually intractable and approximate approaches are used in practice. A lot of approximation algorithms have been proposed.

The extensive study of bayesian neural network could date back to the 90s [88] [94]. In his Ph.D thesis, Neal discussed the classes of prior distributions of weights when the network width goes into infinity. He solved the integration of posterior using hybrid Monte Carlo. However, these works in the early days used shallow networks. When network gets deeper, new challenges come.

Graves *et al.* [48] took the variational inference approach which approximates the posterior distribution with a tractable distribution $q_\theta(\mathbf{w})$. The optimal distribution $q_\theta(\mathbf{w})$ is choosen by minimizing the Kullback-Leibler(KL) divergence between $P(\mathbf{w}|\mathcal{D})$ and $q_\theta(\mathbf{w})$ defined by:

$$KL\{q_\theta(\mathbf{w})||p(\mathbf{w}|\mathcal{D})\} := \int q_\theta(\mathbf{w}) \log \frac{q_\theta(\mathbf{w})}{p(\mathbf{w}|\mathcal{D})} d\mathbf{w} \tag{4.2}$$

Specificlly, Graves *et al.* discussed using both delta approximating distribution and factorised Gaussian distribution in their work.

Gal [43] ties approximate inference in bayesian models to stochastic regularisation techniques like dropout [120], thus avoiding the necessity to modify the original model.

4.1.3. Aleatoric Uncertainty

All the above works I have talked about tried to approximate the posterior distribution in one way or another. Epstemic uncertainty could be derived by sampling the weights of

the network. On the other hand, aleatoric uncertainty could be derived by applying some noise distribution to the output of the network. Kendall *et al.* talked about applying a Gaussian distribution to the output, the variance σ_i of which will also be an output of the network [67].

$$\hat{x}_{i,t} = \mathbf{f}_i^{\mathbf{W}} + \sigma_i^{\mathbf{W}} \epsilon_t, \epsilon_t \sim \mathcal{N}(0, I) \tag{4.3}$$

When applied to a classification task. They sample the logits T times and using the following loss function:

$$\mathcal{L}_x = \sum_{i=1} \log \frac{1}{T} \sum_t \exp(\hat{x}_{i,t,c} - \log \sum_{c'} \exp \hat{x}_{i,t,c'}) \tag{4.4}$$

Despite the fact the number T could be large, the whole process is still efficient, because for each input, there is only one pass through the network. The sampling happens after we get $\mathbf{f}_i^{\mathbf{W}}$ and σ_i

However, when I directly apply their loss function for classification task, the output uncertainty is not meaningful. Based on Kendall's work, in this study, I introduce a regularizer, which could provide much better explanation about the output as well as maintaining good performance.

4.2. Uncertainties for Knowledge Tracing

In the previous section I introduced two kinds of uncertainties: Epistemic uncertainty and Aleatoric uncertainty. I now demonstrate how to incorporate these uncertainties into a deep knowledge tracing model. I will evaluate these uncertainties separately and my main focus will be on the aleatoric uncertainty. While these two kinds of uncertainties could be combined in one model without changing the optimization process, I observed that modeling epistemic uncertainties is not as effective as modeling aleatoric uncertainties in the context of knowledge tracing. This observation is consistent with the results from [67] despite coming from a different domain. The architecture of the model in this study is shown in Figure 4.2.

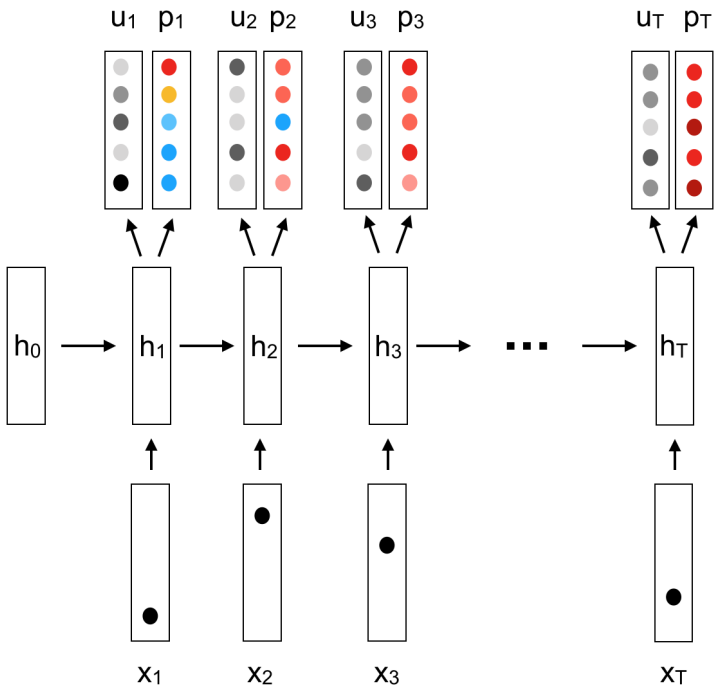

Figure 4.2. Architecture of Deep Knowledge Tracing model with uncertainties.

The input to this model is one hot encoding of the correctness and the corresponding skill id. For each output vector p_t, containing the probabilities of getting different skills correct, there is a corresponding uncertainty vector u_t, which gives the uncertainty level for each predicted skill.

4.2.1. Aleatoric Uncertainty

Data dependent uncertainty or aleatoric uncertainty is caused by noise in the observations or measurements and is considered irreducible (other than via improvements in the measurement device). In the context of knowledge tracing, one can think of these uncertainties as being caused by factors like the difficult level of an item, corresponding guess-ability of an item, etc. For example, items that are too easy or too difficult will have lower uncertainties because students are more likely to get them correct or incorrect. However, for items with difficulty levels in the middle range, the uncertainties might be higher. Fixed weights

for a deep neural network results in a deterministic function, thus if input x_i has normally distributed noise, the output should also have normally distributed noise. In previous works, to model the aleatoric uncertainty Kendall *et al.* [67] proposed for the model to output a variance σ_i for each prediction $\mathbf{f}(x_i)^{\mathbf{W}}$. Then $\hat{y}_i$ is calculated by:

$$\hat{y}_{i,t} = \mathbf{f}(x_i)^{\mathbf{W}} + \sigma_i^{\mathbf{W}} \epsilon_t, \epsilon_t \sim \mathcal{N}(0, I) \tag{4.5}$$

Here t denotes the time when a sampling occurs. This method is a combination of supervised learning and unsupervised learning because we are not given a label for the variance. The expected value of $\hat{y}_i$, however, is often intractable.

$$E_{\hat{y}_i \sim \mathcal{N}(\mathbf{f}(x_i)^{\mathbf{W}}, \sigma_i^{\mathbf{W}})}[\hat{y}_i] \tag{4.6}$$

Therefore a number of researchers use Monte Carlo methods to approximate this value. Let's define the binary cross entropy loss for instance i as:

$$\mathcal{L}_{bce}(y_i, x_i) = -[y_i \cdot \log \sigma(x_i) + (1 - y_i) \cdot \log(1 - \sigma(x_i))]$$

Here σ is the sigmoid function that compresses the raw logit output to (0, 1). Thus, we have the loss function proposed by Kendall *et al.* [67]:

$$\mathcal{L}_{kendall} = \sum_i \mathcal{L}_{bce}(y_i, \frac{1}{T} \sum_t \sigma(\hat{y}_{i,t})) \tag{4.7}$$

where T is the sampling number, I use 100 in the experiment.

4.2.2. A Theoretical Analysis of Kendall's method

In this section, I motivate why using Kendall's loss function directly could not produce sensible variance. Figure 4.3 shows a sigmoid function and its first derivative:

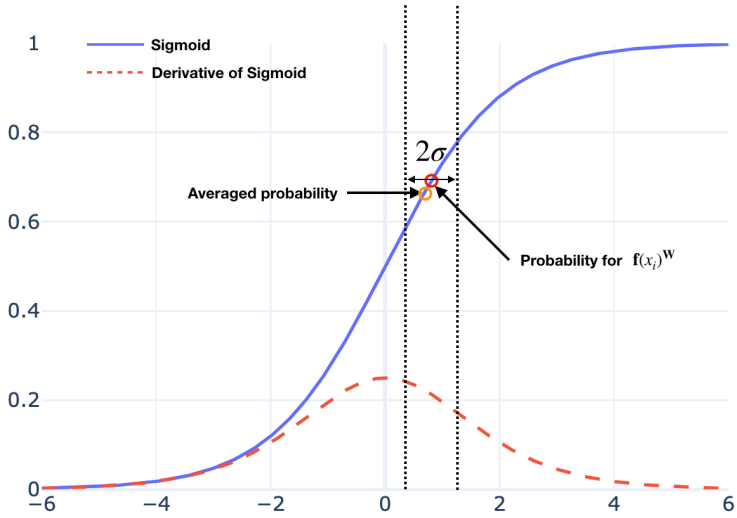

Figure 4.3. Demonstration of sampled logit and its relation to output variance.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.8}$$

$$\frac{\partial}{\partial x}\sigma(x) = \sigma(x)(1 - \sigma(x)) \tag{4.9}$$

For a binary classification problem. Let's first consider the case when the output logit $\mathbf{f}(x_i)^{\mathbf{W}}$ is larger than 0. And we have its corresponding probability $\sigma(\mathbf{f}(x_i)^{\mathbf{W}})$ as shown in Figure 4.3. The first derivative of the sigmoid function on the left side of this logit is larger than that on the right side. Because our noise has 0 mean and variance σ_i^W. If we sample the logit T times, the averaged probability will be less than the probability $\sigma(\mathbf{f}(x_i)^{\mathbf{W}})$ when the logit is greater than 0:

$$\frac{1}{T}\sum_t \sigma(\hat{y}_{i,t}) < \sigma(\mathbf{f}(x_i)^{\mathbf{W}}) \tag{4.10}$$

The larger the output variance, the smaller the averaged probability will be. Thus, to minimize the loss function when target is 1, as shown below:

$$- \log(\frac{1}{T}\sum_{t}\sigma(\hat{y}_{i,t})) \tag{4.11}$$

The training process will make the output variance small, thus to make $\frac{1}{T}\sum_t \sigma(\hat{y}_{i,t})$ as close to $\sigma(\mathbf{f}(x_i)^{\mathbf{W}}$ as possible. This makes intuitive sense that when you have a correct prediction, the variance should be small. But if you have an incorrect prediction (the correct label is 0 in this case). Then it will increase the variance to make the sampled probability as close to 0.5 as possible. However, Kendall's approach does not distinguish the difference of logits as long as they are all correct. In other words, if the correct label is 1, the model will put a small variance for both a logit of 0.6 and logit of 0.9. What we want is we want our model to output more variance when its a 0.6 than 0.9. Because 0.9 is a closer estimate to the true label of 1. Similarly, for values when the logit is below 0, the output probability will be increased, making the average probability closer to 0.5. This will similarly incentivize the model to place a small variance on any correctly labeled logit below 0.

Our goal is to create a model that has high confidence level (low variance) for predictions with increased accuracy. Conversely, the model should make predictions with low confidence (high variance) when predictions may not have high accuracy (at least not as high as these with high confidence level). I achieve this behavior by introducing the following regularization. *This regularization term is the most important aspect of the proposed method*:

$$\mathcal{L}_r = \left|\log\left(\gamma\mathcal{L}_{bce}(\mathbf{f}(x_i)^{\mathbf{W}}, y_i)/\sigma_i\right)\right| \tag{4.12}$$

The regularization penalizes the loss function to a lesser extent when the prediction is incorrect but variance is large. Thus, the undistorted loss and variance change in proportion. We also do not want the variance values to grow without bound, so I introduce an L2-regularization term:

$$\mathcal{L}_g = ||\sigma_i^{\mathbf{W}}||_2 \tag{4.13}$$

The final loss function then becomes:

$$\mathcal{L}_{proposal} = \sum_{i=1}[\mathcal{L}_{bce}(\mathbf{f}(x_i)^{\mathbf{W}}, y_i) + \lambda\mathcal{L}_r + \beta\mathcal{L}_g] \tag{4.14}$$

γ is a hyperparameter to make the value

$$\mathcal{L}_r = \gamma\mathcal{L}_{bce}(\mathbf{f}(x_i)^{\mathbf{W}}, y_i)/\sigma_i$$

close to 1. λ and β are two hyperparameters that control the relative tradeoff between these two regularization terms.

4.2.3. Epistemic Uncertainty

Epistemic uncertainty is also called model uncertainty. It is caused by the fact there are numerous models that could have generated the dataset in hand. This uncertainty could be reduced by collecting more data. The bayesian approach for modeling uncertainties for deep learning models is to assume the weights of these models are not fixed, but sampled from a distribution. The final prediction is generated by integrating overall all possible weights, which is apparently intractable. There are a lot of works proposed to approximate this prediction. Gal *et al.* [43] proposed using stochastic regularization techniques like dropout. The key insight is to turn on dropout during the testing phase. For a classification task, prediction probability is approximated by:

$$p(y = c|x, \mathcal{D}) = \frac{1}{T}\sum_{t}^{T}\text{softmax}(\mathbf{f}(x)^{\mathbf{W}_t}) \tag{4.15}$$

T is the total number of sampling. The uncertainty can be summarized using the entropy $H(p) = -\sum_c p_c \cdot \log p_c$ [67].

The biggest advantage of using dropout is that we do not need to modify the original model. Thus, we can apply this technique to almost any deep neural network model. However, I also observed that, when dropout is turned on during testing, the performance

worsens slightly. These results are discussed in more detail in the results section.

4.2.4. Model Training

Deep knowledge tracing model uses recurrent neural network with LSTM cell [107]. The mathematical elements that comprise each unit are given from equation 2.5 to 2.10. The architecture of our network is shown in Figure 4.2. Different from Deep Knowledge tracing model, we have two output vectors decoded from the hidden state: the prediction vector and uncertainty vector. the model is trained end to end using the Adam optimizer. I use batch size 32, hidden state size 200 and the learning rate is set to be 0.1. The model is implemented using pytorch 1.1.0 and trained on a GTX 1080 node.

4.3. Datasets

I evaluate the proposed methodology on several datasets and compare it with the model proposed by Kendall *et al.* [67]. The specifics of each datasets are given in Table 5.3. I have removed students with less than three attempts. Despite the Assistment skill builder 09-10 and 14-15 datasets, KDD 2010 dataset, I also use the following synthetic dataset:

Synthetic-5: To evaluate the proposed methodology in a more controlled way, I created a synthetic dataset similar to the one used in the original DKT model [107]. I construct this synthetic data as follows: I assume each student has an ability α and each question has a difficulty β. The probability of a student with α ability getting a problem with difficult β correct is modeled using Item Response Theory [40].

$$p(correct|\alpha, \beta) = c + \frac{1-c}{1+\exp(\beta - \alpha)} \tag{4.16}$$

Where c is the guess probability (usually set to 0.25 if there are four choices, 0.2 if there are five). Note here I am not using the dataset directly from [107], but created my own, such that I would be able to access the probabilities of generating the responses. This is important to quantitatively evaluate the uncertainties. I am using this synthetic data only

Table 4.1. Dataset Summary

	# Records	# Students	# Records Per Student	# Skills
Assistment 09-10	338,001	3,884	87	124
Assistment 14-15	822,938	19,523	42	100
KDD 2010	510,749	557	916	100
Synthetic-5	200,000	4000	50	5

for evaluating the proposed methodology—I am not proposing a mature assessment model. The pseudo code used to generate this dataset is given in Algorithm 1.

4.4. Results

The goal of this study as mentioned above is to build a model capable of providing a reasonable confidence level for each prediction. If we interpret the output variances as a measure of confidence level, lower variance should signify high confidence. I will first empirically show why Monte Carlo sampling alone may not help learn the data dependent variance. Then, I will compare the aleatoric uncertainties and epistemic uncertainties generated using the proposed method with Kendall's. Finally, I will investigate these uncertainties quantitatively using synthetic data.

4.4.1. Monte Carlo Sampling for Variance

As previously discussed, if input x_i is drawn from normally distributed noise, the output is often also normally distributed. Thus, we can model the aleatoric uncertainty by making the model output a variance σ_i for each prediction $\mathbf{f}(x_i)^{\mathbf{W}}$ [67]. Then $\hat{y}_i$ is calculated by:

$$\hat{y}_{i,t} = \mathbf{f}(x_i)^{\mathbf{W}} + \sigma_i^{\mathbf{W}}\epsilon_t, \text{ where } \epsilon_t \sim \mathcal{N}(0, I) \tag{4.17}$$

Where t denotes the time when a sampling occurs. I investigate this methodology using Monte Carlo simulation and find that it does not produce a meaningful data dependent variance. Figure 4.4 shows the loss and output variance for each batch over time (I use batch

Algorithm 1 Synthetic Data Generation

c: guess probability, set to 0.25
l: learning rate, set to 0.1
q_id: an array for 50 skills
q_beta: an array of 50 difficult levels for corresponding skills
q_mean_beta: is an array storing the mean difficult level for 5 skills

Initialization: *q_id* = [], *q_beta:* = [], draw 5 random numbers from normal distribution with mean 0 and standard deviation 1.0 and put them into array *q_mean_beta*

```
# Get the difficult level for 50 questions
for i from 1 to 50 do
    current_skill = random.uniform(0, 4)
    q_id.append(current_skill)
    current_skill_beta = random.normal(q_mean_beta[current_skill], 1.0)
    q_beta.append(current_skill_beta)

# Generate 4000 students
for i from 1 to 4000 do
    current_student_alpha_mean = radnom.normal(0, 1.0)
    current_student_alpha = np.random.normal(current_student_alpha_mean, 1.0, 5)
    for i from 1 to 50 do
        calculate p_correct using Equation (8)
        update the corresponding student alpha by learning rate l
```

size of 32 and plot the average variance). Distorted loss is calculated using the noisy logit, $\hat{y}_i$, and the target. Undistorted loss is calculated using the raw logit, $\mathbf{f}(x_i)^{\mathbf{W}}$ and target. As we can see, both the distorted loss and undistorted loss decrease with time. However, the output variance rarely changes over all training iterations and batches. This strongly implies that the model does not learn any data dependent variance, regardless of training iterations. I repeat the experiments with T=100, T=500 and T=1000 and the result is the same. This result is perhaps not very surprising—we cannot rely on an unsupervised output to learn a meaningful data dependent variance. Instead, we must give the model some useful information to guide the learning process. I achieve this by explicitly regularizing the loss function.

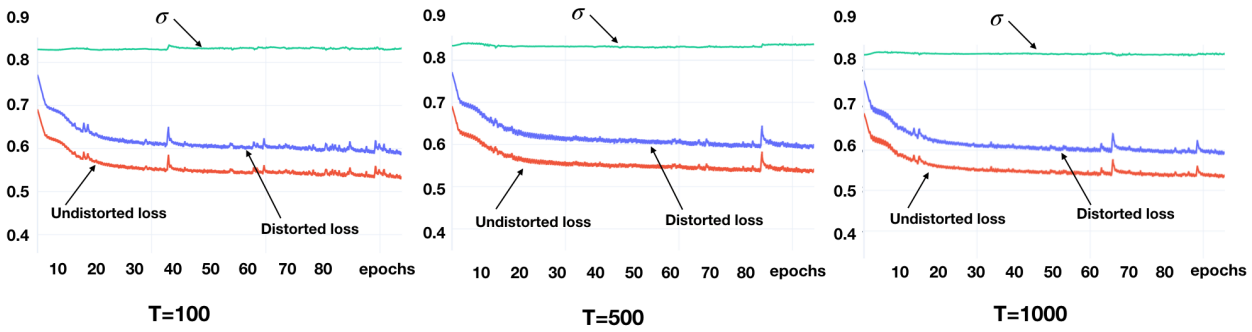

Figure 4.4. Monte Carlo Sampling for different T, distorted loss is calculated using the noisy logit ($\hat{y}_i$) and the target. Undistorted loss is calculated using the raw logit. ($\mathbf{f}(x_i)^{\mathbf{W}}$) and target.

4.4.2. Aleatoric Uncertainty

With aleatoric uncertainty, we are modeling the effect of noise level of the input data on the model output. In the context of knowledge tracing, this noise is a combination of human factors like guess, slip, *etc.* and the difficulty level of the problem itself. Intuitively, mediocre students answering medium difficulty level problems should exhibit high uncertainties. On the other hand, very talented or poor performing students should exhibit lower uncertainties. To capture this behavior, I divide the variance values into different percentiles as shown in Table 4.2 and Table 4.3. In these data, performance is categorized by the aforementioned percentiles. As we can see, using the proposed methodology, the proposed model achieves the best performance when the the value of the variance is less than 25^{th} percentile. The performance and variance percentile groupings follow each other almost monotonically—as the percentile increases, the performance worsens. If the DKT model is calibrated, we may just use the output logits as a proxy for uncertainties and should be able to see similar patterns. I did not further investigate this and leave it for future research. On the other hand the performance of Kendall's method is not consistent across percentiles. I believe, since the normal distribution is symmetric, the added punishment from Kendall's method could either increase or decrease the original logit with the same probability. On the other hand, the proposed method produces low variance for small undistorted loss and large variance for

Table 4.2. Area under the ROC curve (AUC), Grouped by Different Aleatoric Uncertainty

	Assistment 09-10		Assistment 14-15	
	Proposal	Kendall's	Proposal	Kendall's
Overall	0.7499	0.7554	0.6905	0.6856
σ <25%	**0.7854**	0.7512	**0.7132**	0.6920
σ <50%	0.7646	0.7439	0.7005	0.6874
σ >50%	0.7227	0.7454	0.6684	0.6837
σ >75%	0.7165	0.7558	0.6625	0.6827
DKT (Baseline)	0.7483		0.6908	
	KDD 2010		Synthetic-5	
	Proposal	Kendall's	Proposal	Synthetic
Overall	0.8125	0.7966	0.7410	0.7440
σ <25%	**0.8233**	0.7923	**0.7614**	0.7522
σ <50%	0.8146	0.7921	0.7480	0.7469
σ >50%	0.8101	0.7999	0.7090	0.7309
σ >75%	0.8123	0.7999	0.6925	0.7235
DKT (Baseline)	0.8003		0.7433	

large undistorted loss. The punishment will be large if the undistorted loss and variance diverge from each other.

4.4.3. Epistemic Uncertainty

Epistemic uncertainty is caused by the limited number of data points. There might be numerous processes that could have generated the dataset we have. And, it is understood that most epistemic uncertainty can be explained away with more data.

I evaluate epistemic uncertainty on three different models and the results are shown in Table 4.4 and 4.5. The training process is the same as for aleatoric uncertainties. The difference is in the testing process. During testing, I turn on dropout and sample the logit T times for each prediction. The entropy is used as our uncertainty. As we can see modeling

Table 4.3. Square of linear correlation (r2) for Different Aleatoric Uncertainty

	Assistment 09-10		Assistment 14-15	
	Proposal	Kendall's	Proposal	Kendall's
Overall	0.1895	0.1828	0.0956	0.0917
σ <25%	**0.2708**	0.1946	**0.1127**	0.0982
σ <50%	0.2213	0.1813	0.1017	0.0930
σ >50%	0.1500	0.1829	0.0787	0.0893
σ >75%	0.1419	0.2017	0.0756	0.0871
DKT (Baseline)	0.1860		0.0944	
	KDD 2010		Synthetic-5	
	Proposal	Kendall's	Proposal	Kendall's
Overall	0.2540	0.2293	0.1460	0.1496
σ <25%	**0.2792**	0.2252	0.1410	**0.1606**
σ <50%	0.2578	0.2237	0.1382	0.1502
σ >50%	0.2499	0.2338	0.1199	0.1377
σ,>75%	0.2623	0.2351	0.1068	0.1318
DKT (Baseline)	0.2339		0.1499	

epistemic uncertainties does not always improve the performance. It is not as effective as modeling aleatoric uncertainties, which is consistence with the observations from [67].

4.4.4. Synthetic Data

When uncertainties are evaluated in tasks like image segmentation, subjective judgement could be used. Although, such subjective judgement is useful for evaluating a single model, to compare different models, we need a more quantitative method. Thus, I created a synthetic dataset, in which I have access to the actual probabilities that generated the student responses—that is, we know the c, α, and β values in Equation 4.16. Recall that the α parameter corresponds to student ability, with higher values denoting greater ability. The pseudo code used to generate these student responses is shown in Algorithm 1. I assume

Table 4.4. Area under the ROC curve (AUC) for Different Epistemic Uncertainty

	Assistment 09-10			Assistment 14-15		
	Proposal	Kendall's	DKT	Proposal	Kendall's	DKT
Overall	0.7438	0.7468	0.7481	**0.6873**	**0.6894**	**0.6902**
$H(p)$ <25%	**0.7648**	**0.7794**	**0.7862**	0.5996	0.5960	0.5973
$H(p)$ <50%	0.7389	0.7440	0.7537	0.6278	0.6269	0.6162
$H(p)$ >50%	0.6492	0.6473	0.6460	0.6064	0.6120	0.6165
$H(p)$ >75%	0.5866	0.5822	0.5800	0.5619	0.5741	0.5843
	KDD 2010			Synthetic-5		
	Proposal	Kendall's	DKT	Proposal	Kendall's	DKT
Overall	**0.7976**	**0.8128**	**0.8095**	**0.7423**	**0.7421**	**0.7373**
$H(p)$ <25%	0.7658	0.7944	0.7870	0.6234	0.6236	0.6344
$H(p)$ <50%	0.7396	0.7678	0.7643	0.6615	0.6609	0.6556
$H(p)$ >50%	0.7037	0.7165	0.7141	0.6349	0.6351	0.6294
$H(p)$ >75%	0.6358	0.6459	0.6448	0.5808	0.5847	0.5811

students with high mastery level (large α) should behave consistently and are therefore easier to predict (compared with students in the middle range of ability). I hope the proposed model could give low variance outputs on these students because the model can be more certain about this consistent behavior. However, the relationship is not so simple. When a student starts responding to questions, the model should output high variance because it knows little about the student. As the student undertakes more and more attempts, the model should become more familiar with the student and the uncertainties should decrease.

Figure 4.5 illustrates this behavior by plotting the average uncertainties during the 50 exercises that a student undergoes. The average uncertainty for each exercise is plotted for the top performing students (denote as large α values in the key). These students are the top 25% performing students in our synthetic dataset. Similarly, mid-range performing students are selected from the α values in the 25% to 50% percentiles. All students work on the same 50 exercises in the same order, which means that the β parameters are identical across all

Table 4.5. Square of linear correlation (r2) for Different Epistemic Uncertainty

	Assistment 09-10			Assistment 14-15		
	Proposal	Kendall's	DKT	Proposal	Kendall's	DKT
Overall	0.1806	0.1839	0.1864	**0.0909**	**0.0951**	**0.0960**
$H(p)$ <25%	**0.3401**	**0.3626**	**0.3813**	0.0205	0.0165	0.0137
$H(p)$ <50%	0.2457	0.2556	0.2712	0.0421	0.0396	0.0258
$H(p)$ >50%	0.0701	0.0674	0.0661	0.0342	0.0399	0.0432
$H(p)$ >75%	0.0229	0.0205	0.0196	0.0121	0.0174	0.0240
	KDD 2010			Synthetic-5		
	Proposal	Kendall's	DKT	Proposal	Kendall's	DKT
Overall	0.2261	0.2561	0.2475	**0.1478**	**0.1471**	**0.1433**
$H(p)$ <25%	**0.2643**	**0.3550**	**0.3373**	0.0141	0.0142	0.0174
$H(p)$ <50%	0.1933	0.2560	0.2423	0.0383	0.0377	0.0363
$H(p)$ >50%	0.1272	0.1451	0.1407	0.0553	0.0557	0.0526
$H(p)$ >75%	0.0563	0.0654	0.0651	0.0196	0.0216	0.0202

students. Thus, we can just look at the α values. I use α values from 25% to 50% as the medium values to compensate the fact that one student could always guess the item correct with probability 25% (parameter c in Equation 4.16). From Figure 4.5 left, we can see the average uncertainties both decrease for both groups of students. However, the group with large α decrease much faster, which is consistent with our intuition. These high performing students can have their responses predicted with greater certainty than mid range students. This behavior is exactly what is expected for an uncertainty measure. Figure 4.5 right plots the uncertainties for Kendall's method. However, these uncertainties do not follow any consistent pattern. This strongly supports the hypothesis that the proposed method captures aleatoric uncertainty in the dataset, whereas Kendall's method does not.

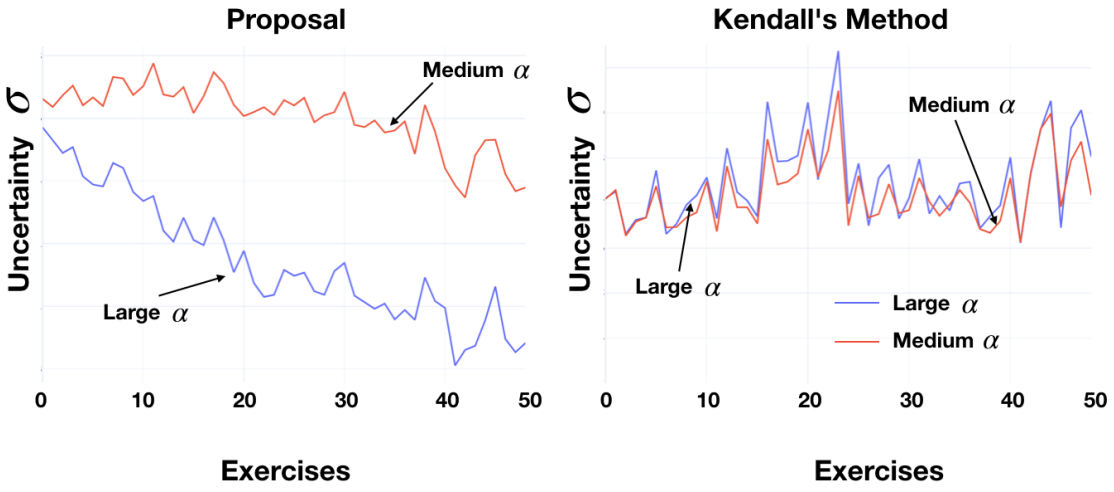

Figure 4.5. Average uncertainties change with 50 exercises attempts for our proposed method and Kendall's method for high performing students and mid-range performing students. Notice that our proposed method becomes more certain as a students responds to more questions. Moreover, for high performing students, this uncertainty can be significantly reduced.

4.5. Discussion

In this chapter, I proposed a way of incorporating prediction uncertainties into deep neural network models in the context of knowledge tracing. I first empirically showed that Monte Carlo sampling alone may not help learn the data dependent variance. Next, I proposed the methodology by explicitly regularizing the loss function to incentivize expected behavior. I evaluated our methodology on three different real datasets. The results show that the proposed model can provide comparable results as well as a confidence level for each prediction. To further quantitatively understand our model, I created a synthetic dataset with known uncertainty patterns. With access to the simulated data, we were able to understand the generated uncertainties in a quantitative way. I also showed that our model follows the expected behavior on the synthetic data, whereas other methods do not. While the results are promising, there is considerable room for further improvement in this research space, which I leave for future work.

Chapter 5

INCORPORATING MORE MODALITIES FOR STUDENT RESPONSE MODELING

We see, hear, smell and touch the surroundings in this world. We combine all these information from different channels to better understand what is happening. For instance, we use both voice and gestures to communicate with each other. Now, the explosion of the number of different sensors, especially those embedded in mobile devices, greatly extends our ability to feel the world. A machine learning problem is unimodal, if it only involves one modality. In this study, I will use the same definition of modality as used in [111]. When observing a phenomenon or system using multiple sensors, the output of each sensor is termed a modality. I will loosely use modalities as features extracted either manually or learnt by models. Conventional object recognition classifiers only involve images, natural language processing only deals with voice signal. To accomplish truly machine intelligent, we must take multiple modalities into consideration. There are at least three benefits [11] that multimodality could provide. First, all modalities observed the same phenomenon that may result in more robust predictions. Second, multi modalities may provide complementary information. Third, a multimodal system could still work if one of the modalities is missing. Numerical empirical results have shown that adding more modalities does improve the performance [96, 109, 121].

In this chapter, I will first present the EduAware system which is a collaborative work with Doyle *et al.* [36] in section 5.1. EduAware uses multimodality features from the tablet to predict student performance. In section 5.2, I will briefly introduce different ways of fusing mutli modalties. Then, I will review some works that use multimodality for knowledge tracing in section 5.3. We will see, the way they use these multi modals are still the legacy from conventional methods (features are extracted manually and simply concatenated). Simply concatenation does provide a way for multimodal fusion, however, it does not fully exploits the hierarchical representation ability of deep neural networks. Besides, it is known that

simply concatenation makes it hard to learn intra-modal relations [111]. At last, I will explain how to apply NAS for multimodal fusion.

5.1. The EduAware Learning Module

In this section, I will present a context-aware, tablet-based learning module for adult education. Specifically, I focus on adult education in healthcare–teaching learners to perform a medical screening procedure. Previous works on performance prediction were mostly evaluated on Learning Management Systems (LMS), Massive Open Online Courses (MOOC), or web based Intelligent Tutoring Systems (ITS) [53, 60, 95, 106, 142]. The popularity of mobile devices like smart phones and tablets extends the realm of education and provides new interaction features. Based upon how learners navigate through the learning module (e.g., swipe-speed and click duration, among others), I use machine learning to predict what comprehensive test questions a user will answer correctly or incorrectly. Compared with other context aware learning applications, this is the first time tablet-based navigation gestures have been used to support learning assessment. Figure 5.1 shows a participant using the learning tool.

The interactions performed with tablets, like swipe speed and click duration, could provide a rich feature set to use for characterizing student performance. Recent research indicates that keystroke dynamics (such as mouse movement, keystroke, etc.) can reveal aspects of the cognitive state of a user [14, 130], motivating its possible inclusion for educational applications. **In particular, I ask: is there a relationship between tablet-based interactions and learners' answer responses that reveals how a learner will respond to assessment questions?** A user study involving 21 participants was conducted, showing the system can predict how users respond to test questions with 87.7% accuracy without user-specific calibration data. This is compared to item response theory methods which achieve about 70% accuracy. I also investigate which attributes are most responsible for making predictions.

Figure 5.1. A participant using the EduAware learning tool.

5.1.1. Summary of Tablet Learning Modules

The learning module focuses upon instructing individuals to perform cervical cancer screening (VIA). This learning process is chosen because there is a vast shortage of trained public health workers that understand the screening procedure. Future work will investigate how EduAware can promote online learning for health workers. The current study, however, focuses on contributions in task prediction. This work could be easily extended for knowledge tracing.

Iterative methods were used for selecting material from the popular courses to be included in the learning tool [20, 73, 103]. The tool is broken into 14 sections as numbered in Figure 5.2. These numbered blocks are included in each description below:

- **Introduction:** Learn about why cervical screening is important and how to navigate tablet application. Comprised of items 1-4 in Figure 5.2.
- **Module 1:** Learn how to calculate sensitivity and specificity and understand why they are important. Comprised of items 5-6.
- **Module 2:** Learn the potential of a screening program to reduce cancer deaths and improve the lives of women. Comprised of items 7-8.

Figure 5.2. A flowchart of the learning tool with callouts of various modules.

- **Module 3:** Understand how the VIA screen-and-treat approach works and why it benefits low resource areas. Comprised of items 9-10.
- **Identifications:** (Video) How to identify precancerous cervical lesions and judge severity. Comprised of items 11-12.
- **Comprehensive Test:** Have an understanding of sensitivity, specificity, the necessity of scalable screening procedures, the specifics of the screen-and-treat approach, and have the ability to identify the severity of precancerous cervical lesions with high confidence. Comprised of item 13.
- **Exit Survey:** An exit survey and interview is conducted to collect feedback. Comprised of item 14.

Self-assessments from module 1 to 3 all have multiple choice questions while in identification exercises, users are presented with images of cervixes and they must distinguish as negative, positive, or positive invasive. The comprehensive test module contains 22 questions covering all material from the learning modules. It is these comprehensive test questions that I seek to predict user responses to with EduAware. All questions are new to the users, but cover some similar content to questions from the self-assessments and cervical lesion identification exercises. Unlike the self-assessments, the comprehensive test does not provide feedback on the correctness of the answers.

5.1.2. Tablet-based Features

The learning app is designed in a way that encourages natural interactions. For example, the sliding page transition between quiz questions and response feedback is controlled by adaptive animations–the user can swipe quickly or slowly and the interface will move with their swipe speed (i.e., direct manipulation). For each question in the self-assessments, 6 features are collected. All features are continuous valued except where noted and are grouped as follows:

- If the self-assessment response is correct (binary). Abbreviated: *Res.*
- Response navigation features (not specific to tablet). Abbreviated: *Navi.*
  - the cumulative time a learner spends on the quiz for each module (the time passed since the first quiz loads until the user selects the response).
  - the time a learner spends on answering a self-assessment question.
  - the time a learner spends reviewing response feedback to a self-assessment.
- Tablet interactions, Abbreviated: *Tablet*
  - the button tap duration for selecting the response (i.e., how long a user kept his/her finger on a button when answering).
  - the review tap duration (i.e., how long a user kept his/her finger on a the "continue" button when navigating).

- Aggregated features of the learning materials, not including quiz parts. Abbreviated: *Agg.* (i.e., Averaged swipe time, time spent on each module).

- Bag of gestures: gestures that participants perform on the screen that are not related to navigation including swipes, taps, failed swipes, long presses, video fast-forwarding, rewind, play, and pause. Abbreviated: *Bag*

These divisions above will help us have a more fine grained understanding of which features are most predictive in later analysis. The final set of features (*Bag*), are eight tablet interactions that are not directly related to navigation on the tablet. They occur infrequently and are logged throughout the learning module. Because they are sparse features, they are aggregated into an eight element count vector of the number of times they occurred (similar to bag-of-words feature extraction typically used in text analysis [65]).

5.1.3. Study Design

21 participants were recruited by word of mouth, email, class announcements, and flyers. All participants were volunteers and were compensated for their time. All experimental procedures were IRB approved. Each timeslot was scheduled for one hour and each participant was given an Apple iPad and headphones used to eliminate audible distractions and for video media as shown in Figure 5.1. All experiments were conducted in a controlled room.

The demographics of all participants are described in Table 5.1. No participants reported any learning disabilities. Additionally, all participants were asked to describe any experience they had with online learning, cancer knowledge, medical experience. Researchers coded the responses as *no*, *some*, or *extensive*. All coding was performed by two researchers independently. Both researchers coded responses identically, revealing that the responses were straightforward to interpret.

The scores of the comprehensive test range from 50% up to 77%. The average score is 63% and median is 64%. To assess the difficulty of the items in the learning module and further analyze performance I use item response theory (IRT) [128]. After calculating the difficulty of each question, the ability of each person can also be estimated directly from the

Table 5.1. Participant Demographics

Participant Demographics	
Race	White=16, Asian=4, Hispanic/Latino=1
Gender	Female=15, Male=6
Education	College Enrolled=16, BS=4, MS=1
Previous Experience with:	
Medicine	Extensive=0, Some=6, No=15
Cancer	Extensive=0, Some=5, No=16
Online Learning	Extensive=0, Some=7, No=14

IRT model. This is typically known as person-ability and part of the joint estimation of the model. I use maximum likelihood estimation to assess the fit and expectation maximization to optimize the parameters [127].

Explanation (item): Figure 5.3 (right) shows an item response map (also called a Wright-map [138]) summarizing the IRT model. I combine the responses of the self-assessment questions and the comprehensive test questions, resulting in 18+22=40 questions. The map shows a dense display of these combined questions on the x-axis versus the estimated difficulty on the y-axis for each question. The y-scale is arbitrary, but negative values denote easier questions and positive values denote harder questions. The points are color-coded based upon if they came from the self-assessments or the comprehensive test.

Result (item): In the self-assessment, multiple-choice questions are relatively unchallenging to participants as indicated by the negative values for each question. About half of the lesion identification questions are fairly challenging in the assessment (positive values), although the remaining half are about as difficult as the multiple-choice questions. In the comprehensive test, the information is more challenging, partly due to a multiple-choice question (regarding interpretation of screening sensitivity) and the identification exercises. Participants have considerable difficulty with lesion identification, especially Q16, which had widespread lesions that decreased contrast to healthy tissue.

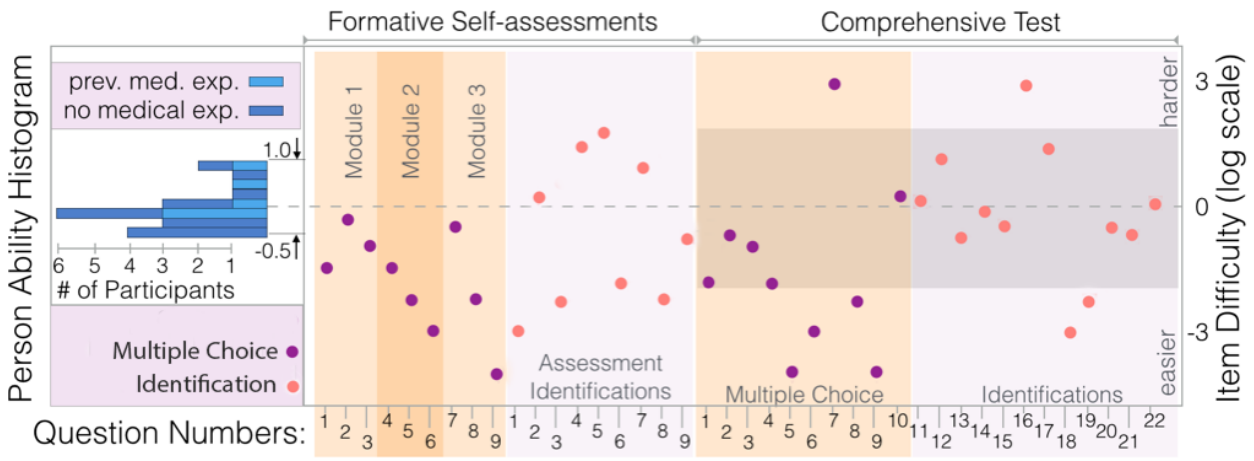

Figure 5.3. A Wright Map summarizing the item difficulty and person-ability as estimated by our IRT model. The dark band corresponds to questions selected for prediction analysis.

Implication (ability): For predicting responses, we should not include questions that are overly easy or overly difficult. Questions that are too difficult or too easy will be relatively easy to predict user response (i.e., always predict wrong or always predict correct). However, test questions with difficulties nearer to zero are less likely to be predicted by chance because users respond to them correctly or incorrectly with about the same frequency. The dark band in the Wright Map denotes which questions are most difficult to predict from the comprehensive test: Q1-Q4, Q10-Q15, Q17, and Q20-Q22. In analysis, I mainly focus on these questions because they are the most challenging to predict.

Explanation (ability): The person-ability (Figure 5.3 left) is plotted on the same y-axis scale as the item difficulty, as a histogram binned upon the number of participants with the corresponding person-ability score. Negative values can be interpreted as less capable learners. Positive values are more capable learners. The individuals are also grouped by whether they had any previous medical experience, denoted by color stacking in the histogram.

Result (ability): When grouping the person-ability histogram by prior medical experience, those who had medical experience scored marginally higher than those who did not. Even so, based on a t-test of the two groups, we cannot say that the means are different

($p = 0.11$). However, because the p-value is borderline a larger sample might be warranted to understand if a real difference exists. Regardless of statistical significance, the impact of having previous medical experience appears to be marginal, if any difference does exist. When comparing ability with whether the participant has prior cancer knowledge, we cannot say the means are statistically different ($p > 0.5$).

Implication (item): The overall performance on the module is fairly poor. Regardless of medical experience, the participants struggled to answer questions correctly. In particular, the visual identification questions are quite difficult for the participants. However, this module is only one of many involved in teaching individuals about precancerous lesions and screening–it is recommended that individuals see 50-100 images before they are prepared to diagnose a human cervix based on inspection [24].

5.1.4. Predicting Student Responses

To compare the proposed method with IRT models, an analysis of IRT using this data is conducted. I calculated the accuracy of Rasch, 2PL, 3PL models using similar methods from [63]. All models are cross validated using leave-one-participant-out cross-validation, where no user-specific data is used for training when that user is in the test fold. Table 5.2 describes the performance of these different models. Results are summarized for all questions (Table 5.2 right) and for the "Hard to Classify" subset of questions described in the previous section (Table 5.2 left). I report the average accuracy and the average F1-score of these models. These metrics are averaged across all 21 participants and across all questions. In this context, accuracy can be defined as the percentage of occurrences where the model correctly predicted that the user would answer a response as correct/incorrect. F1-score is similar to accuracy, but places emphasis on the positive class predictions (*i.e.*, a weighted average of precision and recall). I define the positive class in our examples as when the learner incorrectly answers a question, thus placing more emphasis on the ability of the model to predict errors from the learner. Both accuracy and F1-score in Table 5.2 result in similar conclusions about comparing each model.

Table 5.2. Model accuracy and F1 score for various methods of predicting student responses.

	Subset Questions (N=14)		All Questions (N = 22)	
Model	Accuracy	F1-Score	Accuracy	F1-Score
Majority Classifier	67.3%	0.731	77.2%	0.826
Rasch	67.3%	0.731	77.2%	0.826
2PL	67.6%	0.726	76.4%	0.815
3PL	69.3%	0.747	77.4%	0.827
EduAware (Res. only)	70.4%	0.737	79.2%	0.833
EduAware (All features)	**87.7**%	**0.892**	**90.2**%	**0.922**

Among these IRT models, the best performance I get is from 3PL model, having the accuracy of 69.3% and 77.4% compared to a simple majority classifier with accuracy of 67.3% and 77.2%. One interesting observation here is that the Rasch model has the same accuracy as the majority classifier. I hypothesize this is because discrimination and guess-ability are meaningful for this task, which the Rasch model does not consider. The performance of EduAware is superior to the IRT models and the difference is statistically significant.

I train a separate model for each question in the comprehensive test. The features for each model are selected from data logged during all the self-assessments. As mentioned in the feature extraction section, I investigate 6 features for each self-assessment question. This results in a total of (18 questions) x (6 features) = 108 features to use for prediction. All features are preprocessed to have zero mean and unit standard deviation. 7 aggregated features and 8 bag-of-gestures features (discussed previously) are also collected. These features are not included in our results presentation because they never provided any performance improvements.

The machine learning model for a single question uses 108 features to predict if a user answers the question correctly. Because we have 108 features and 21 training examples, this prediction problem is clearly under-constrained. As such, I impose strict feature selection and cross-validation techniques to guard against over-fitting. First, I employ less expressive algorithms, opting to use a model with linear decision boundaries: logistic regression. Second,

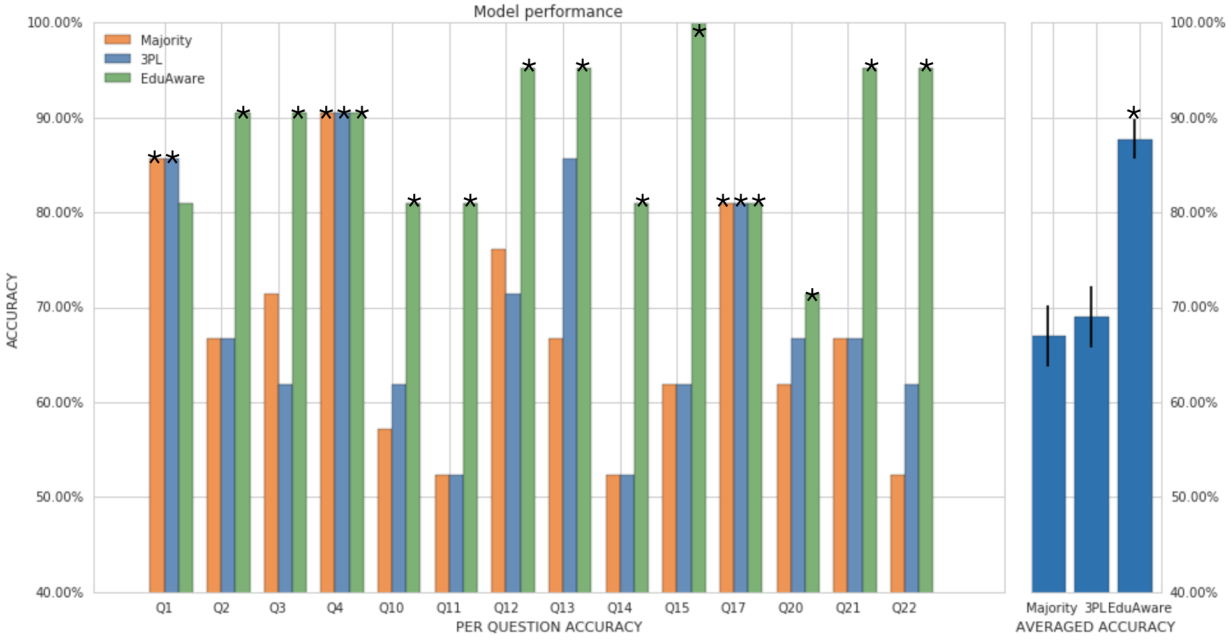

Figure 5.4. Model performance in predicting each subset question, paired t-test t = 4.9873, $p < 0.001$ for 3PL model, t = 5.3033, $p < 0.001$ for majority classifier.

I use recursive feature elimination (RFE) [19] to iteratively eliminate coefficients in the linear models with small magnitude. Finally, to guard against over-fitting I employ leave-one-participant-out cross-validation. RFE is run inside this cross-validation loop so that no data snooping is possible across folds. That is, I use a nested cross validation where RFE parameters are selected and trained in the inner loop and then applied to the left-out participant in the outer loop. While this means that the chosen features might be different for each participant, I have noticed that the selected features for a specific question are similar across participants. That is, the highest magnitude features in the models are almost always identical and never are the feature sets disjoint across participants. Because I use leave-one-participant-out cross-validation, the evaluation mirrors a scenario where no user-specific training data is used–that is, no user-specific calibration is required.

Figure 5.4 shows the accuracy of EduAware in predicting responses for comprehensive test questions in the subset of "Hard to predict" (N=14 questions). I also show accuracy for

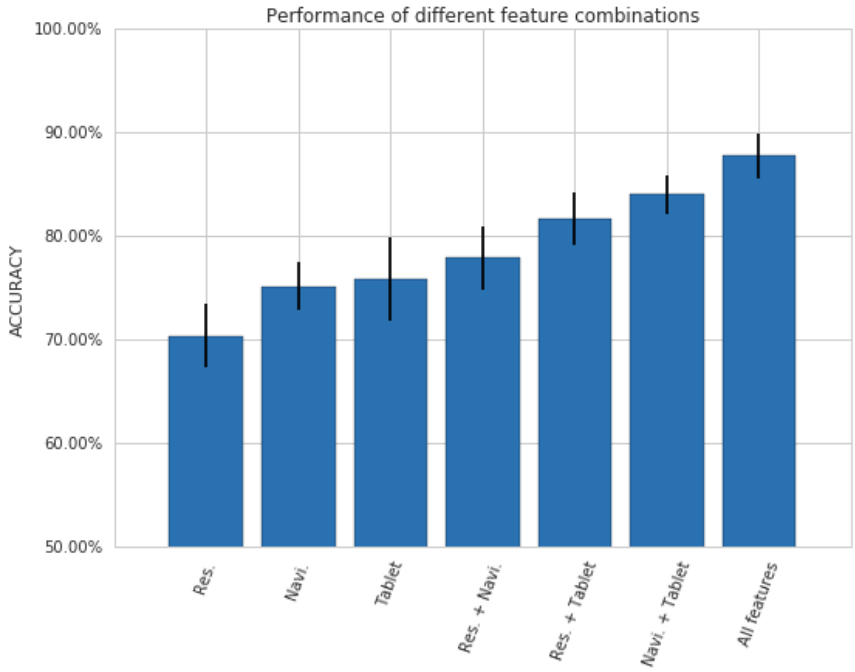

Figure 5.5. Sorted accuracies of different feature subsets and combinations of feature subsets. Error bars signify the 95% interval.

the 3PL model and majority classifier. The majority classifier simply assumes that the user will answer the question the way most other students answered the question. Statistically superior performance is calculated by paired t-test with $p < 0.01$ designating rejection of the null-hypothesis. The statistically best performing algorithms are denoted with an asterisk. For all but three questions (Q1, Q4, Q17), EduAware is the statistically superior performer. For Q4 and Q17, EduAware is one of the statistically best performers.

5.1.5. Which Feature Subsets Are Most Important?

In this analysis, I use the following subset of features:

- *Res.*: using self-assessment responses only, 18 features
- *Tablet* features: these features are specific to tablet, 18x2=36 features
- *Navi.* features: like time spent, review time etc, 18x3=54 features

- Using various combinations of the above subsets. For example, *Res.+Tablet* or *Navi.+Tablet*.

Figure 5.5 shows the accuracy for each subset of features and combination of subsets. Using responses only results in performance that is only slightly better than IRT. The addition of response navigation interactions does improve performance, but tablet gestures provide a substantially greater improvement.

It is interesting that tablet navigation gestures can better predict performance on the comprehensive test than how participants responded to self-assessments. It may be that these features capture the confidence with which a user responds, whereas response data does not. Finally, Figure 5.5 shows that each subset of features add new independent information to the model, as indicated by the improved performance when using all features. This indicates that the interaction of all the features subsets is important to capture, not one single element. However, it is still unclear which of the features are most responsible for prediction. This will be addressed in the next analysis.

5.1.6. What Specific Features Are Most Important?

One method of interpreting features is to review which features are selected by RFE iteration. I seek to answer, of the six different types of features, which features are chosen the most? However, because I use RFE in a cross-validation loop and this loop is run separately for every question and every held-out user in the test fold, there are about (21 users) $\times$ (14 questions) = (294 groups) of selected features. To keep the results tractable, I summarize which of the six feature types were selected for each question on the comprehensive test. Recall that some questions were thrown out because users always responded correctly or incorrectly. I only report results on the "Hard to predict" comprehensive test questions. Figure 5.6 summarizes the features selected and what percentages of RFE trials in which they were selected. That is, for each question, I show a stacked bar plot of the percentage of runs for which the feature was selected. For example, for question 1, the model only employed features related to review time, but question 14 used information from all six feature types.

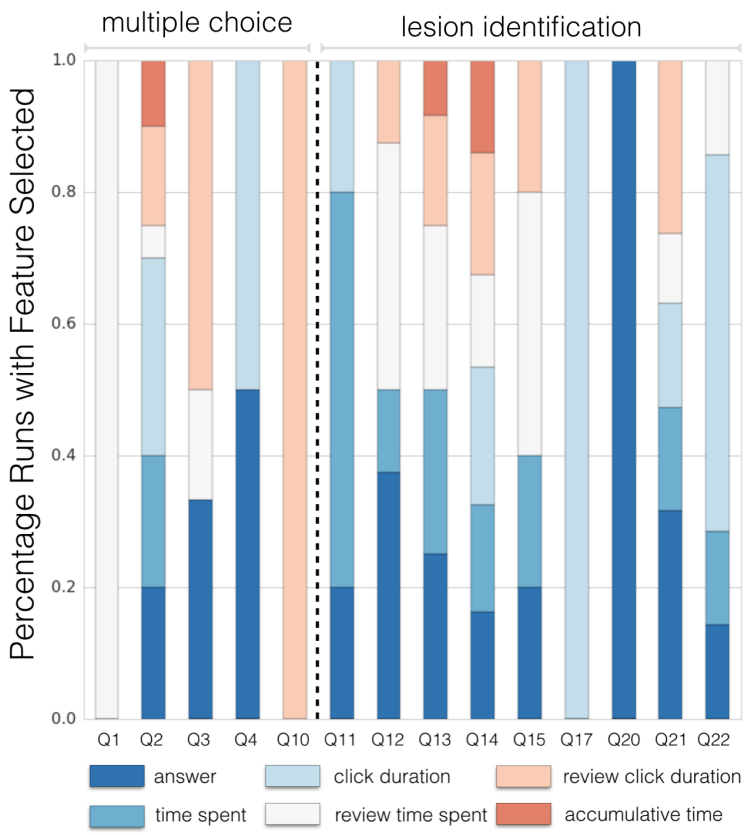

Figure 5.6. Feature categories selected by RFE for each comprehensive test question.

The comprehensive test multiple-choice questions (Figure 5.6, Leftmost 5 columns) tend to be predicted using the review tap duration that participants use to move to the next question, the responses for each questions(correct or not) and the total time spent on review. For identification questions (Figure 5.6, rightmost 9 columns), the features selected are more evenly distributed except Q17 and Q20. Another observation is the accumulative time is seldom selected, it could due to that important information are already been captured by features like time spent, review time spent. These observations have several interesting implications. It is interesting to see click duration and review click duration are so often chosen. One explanation could be the time spent on clicks reflects some confidence level of the participant. Interestingly, the questions that EduAware performs poorly correspond to the questions with only one or two features selected—Q1, Q4 and Q17. This highlights the

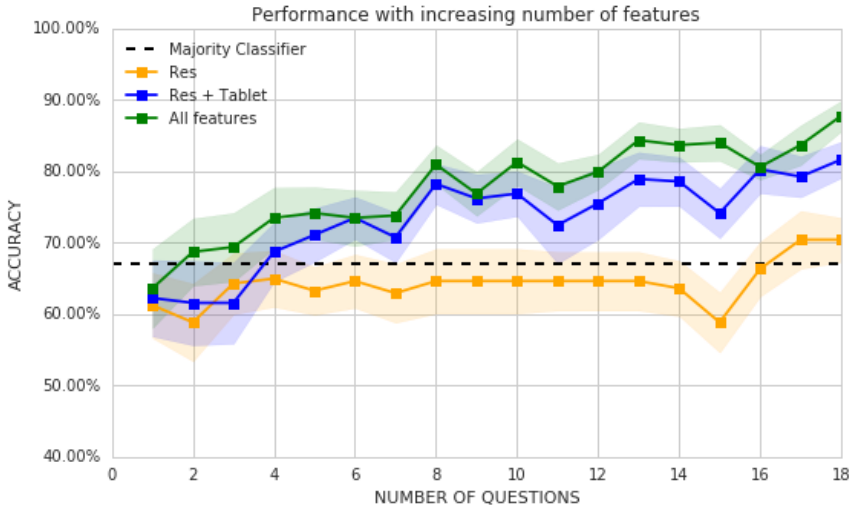

Figure 5.7. Performance of EduAware versus the number of questions used to train the system for each participant.

importance of a multi-modal approach toward inferring user performance (and possibly user context). This may imply that multiple features are essential for capturing which users are confused by the material when variability in the answers is expected.

5.1.7. How Many Self-assessment Exercises Are Required?

In previous analyses, I used features from all 18 self-assesment exercises. In this analysis, I investigate how many self-assessment exercises students need to conduct before the EduAware module can predict their responses on the comprehensive test. To explore this, I add features from each self-assessment gradually, one at a time. The self-assessment features are added chronologically. All cross validation and feature selection methods remain the same. For this analysis, I also investigate three different feature subsets:

- Using only responses (*Res.*).
- Using responses and Tablet interactions (*Res.+Tablet*).
- Using response, Tablet, and Generic interactions (*All features*).

Figure 5.7 shows the accuracy of the models versus the number of self-assessment exercises

used to as training. The area plots around each curve correspond to standard error in the mean. Using responses only does not appear to perform better than a majority classifier until at least 17 of the 18 self-assessments are answered. For the other subsets, as we add more questions, the accuracy improves and stabilizes at 80% after 9 questions. When using all features, there is another jump in accuracy once 16 of the 18 questions are included and again when all questions are included. This implies that a multi-modal approach captures more relevant features to the prediction task. Furthermore, this improvement over using only responses is apparent early in the self-assessment. For example, once four questions are completed, there is a clear separation in the performances. Even so, EduAware cannot achieve 87.7% performance until all the self-assessment examples are used for training. It is also unclear if more training exercises would further increase the performance of EduAware, or if the performance is beginning to plateau.

5.1.8. Conclusion

I present EduAware, an adult learning module that can determine what comprehensive test questions users will answer correctly based upon their earlier responses, navigation timing, and tablet specific gestures. I have shown the importance of modeling navigation behavior and touch gestures in addition to the responses from periodic self-assessments. In particular, I have answered our original question: navigation gestures significantly increase the ability of a system to predict learner responses. These contextual features can be powerful predictors of student performance on a comprehensive test. I evaluated the efficacy of the tool for 21 users, showing that near 87.7% of all user responses could be predicted. I trained different machine learning models and performed analysis on which features are most predictive. Furthermore, I explored adjusting the number of exercises employed in training. One limitation of this study is that the dataset is small (21 participants). In the future, I hypothesize that EduAware could be used to provide just-in-time interventions that increase retention for intelligent tutoring systems by predicting which questions a specific user will struggle with, and then providing additional assistance.

5.2. Multimodal Fusion

The models used in the EduAware learning app (Logistic regression , SVM) are very simple. Thus, I didn't consider the problem of multimodal fusion. In this section, I review different ways of multimodal fusion, preparing for the discussions in the following sections. Prior to the appearance of Deep Learning, there are mainly two ways that multi modalities could be used in a machine learning model. The first one is to combine different modalities at the very beginning. For example, simply concatenate different modalities into one big vector. We hope, the model could learn all useful information from this vector. This method is also called early fusion. Another way is to train a different model for each modality, then combine the outputs from these models to make a final decision. This is similar to the ensemble modeling and also known as late fusion or decision level fusion. Deep neural networks can learn hierarchical representations, thus making intermediate fusion possible.

5.2.1. Early Fusion

Figure 5.8 (a) shows an example of early fusion. For early fusion, either raw data or manually extracted features could be used. However, using raw data could be quite challenging since data from different modalities might have different sampling rate. Besides, some data might be continuous, while others are discrete. Thus, some pre-processing steps must be taken before they could be merged. Automatically learnt representation could also be used. For example, we could use a fully connected layer in deep neural networks to learn an embedding for each modality. We assume these features from different modalities contain complementary information. However, in practice, some features might be just noise. In such cases, feature selection techniques could be used [19]. Based on some performance metrics, features are ranked from the most relevant to least relevant and those least relevant features will be discarded. We have seen the use of Recursive Feature Elimination (RFE) [52] technique in the EduAware learning module to keep the most relevant features for student performance prediction. Feature selection is a huge research field by its own, interested readers could refer to this survey [19].

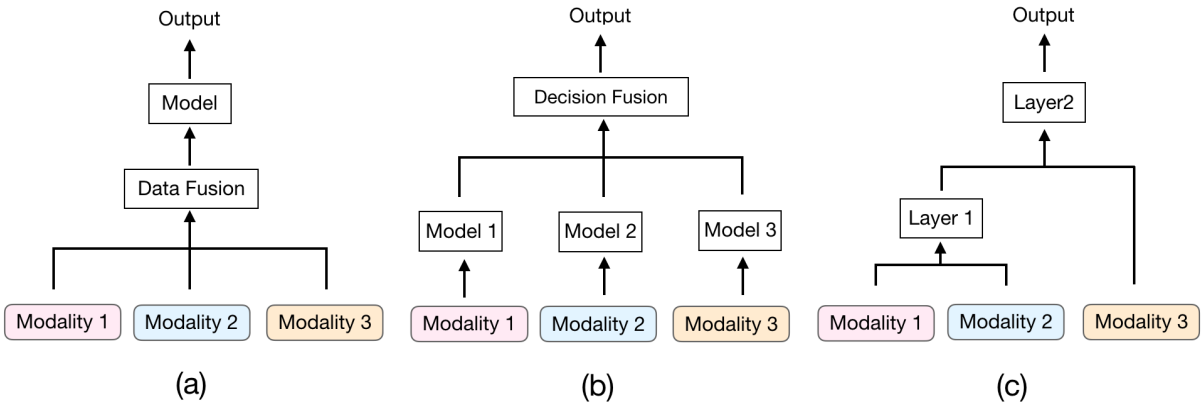

Figure 5.8. Different levels of fusion. (a) Early level fusion, (b) Late level fusion, (c) intermediate level fusion.

Another potential problem for the early fusion is the final combined feature vector might be very big, thus suffering the curse of high dimensionality. In such cases, dimensionality reduction techniques like Principal Component analysis (PCA) or t-SNE could be used. Zhang *et al.* [150] use Auto-Encoder [57] to learn a concise representation of the concatenated modalities for knowledge tracing.

5.2.2. Late Fusion

Figure 5.8 (b) shows an example of late fusion. Late fusion or decision level fusion trains a separate model for each modality and does not have most of the problems early fusion has. For example, in a video classification task, we could use a RNN for the acoustic data and a CNN for visual data. The acoustic data and visual data could have different sampling rate. Then we could combined the decisions from these two models using a fully connected layer. The advantage of late fusion is it allows the building of a specific model for each modality that is supposed to maximise the utilization of that modality. Decision level fusion may perform well when different modalities are highly heterogeneous. Different models usually make different assumptions about the data and may make different errors. Thus, combining the decisions at the very end might be beneficial. The idea of late fusion is similar to ensemble

modeling.

5.2.3. Intermediate Fusion

Neither early fusion nor late fusion is proved to perform better than the other in all situations. Which methodology to use highly depends on the application. The reason why deep learning is so successful on perceptual tasks is its ability to learn hierarchical representations. Features are learnt from data automatically instead of manually crafted. Thus, it seems a natural next step to use deep learning to learn a representation from multi modalities. Figure 5.8 (c) shows an example of intermediate fusion. As we can see intermediate fusion allows features from different modalities to be gradually integrated. Empirical results have shown the advantages of intermediate fusion. However, to use intermediate fusion, we still need to decide the fusion architecture first (which modalities to fuse at which level). Such kind of intermediate fusion using deep learning is also called deep multimodal learning [111]. Deep multimodal learning has been applied to a lot of applications including human activity recognition and medical applications, etc. Radu *et al.* [109] conducted a study of human activity and context recognition using sensor data collected from mobile devices.

One of the research focuses on deep multimodal learning is models that could enforce inter-modality and intra-modality relationship. Learning a shared representation across many modalities is especially helpful, in case, some of which are missing due to sensor failures or other reasons. Another focus is multimodal fusion structure search. As I have mentioned, even though deep learning allows features to be gradually fused. Human experts need to decide beforehand the fusion structure. Neural architecture search (NAS) has been well explored for unimodal tasks and we have already seen several models found using NAS that perform better than those manually crafted [152]. It seems a natural step to apply NAS for tasks involving multimodality. The search process usually starts by extending a seed architecture with new connections or replacing old activation functions with new ones. Thus, NAS is basically a discrete optimization problem. We could use heuristic search, revolutionary algorithms or reinforcement learning for this purpose. In this study, I mainly focus

on reinforcement learning and heuristic search.

5.3. Multimodality for Knowledge Tracing

In this section, I describe knowledge tracing in the context of multimodality. I will first introduce the related works. Then, I present my work exploring the best fusion architecture of different modalities for knowledge tracing using NAS. I will first discuss using NAS in the simplest situation, in which only one modality is involved. Then I extend that work to multimodality cases. I will mainly talk about three different ways of doing mulitmodal fusion using NAS. Current metrics used for knowledge tracing include area under the curve (AUC) and r squared (r^2). However, both these two metrics could not measure how one model performs with time. Thus, I propose using weighted AUC that could measure how one model performs after it sees more and more responses.

5.3.1. Related Work

The EduAware learning module used a small dataset (21 participants), thus conventional machine learning models like Logistic regression, SVMs [26] were used. The popularity of platforms like CMU datashop [2] allows the collection of data from thousands of students with millions of transactions. Such amount of data allows the possibility of using deep learning models for knowledge tracing. These platforms or Intelligent Tutoring Systems usually not only log the correctness of a response, but also things like how many attempts this student has tried, what is the result of the first try, how much time was spent on an item, etc. DKT model only uses the correctness as input. In this section, I will discuss two works that use both multimodality and deep learning for knowledge tracing. However, they only evaluated simple concatenation of different modalities, neglecting the hierarchical representation learning ability of deep neural networks.

Based on the DKT model, Zhang *et al.* [150] tried including more features like *response time*, *attempt count*, *first action*, etc. These features are logged by most intelligent tutoring systems. Figure 5.9 shows the overall architecture of their model. The only difference from

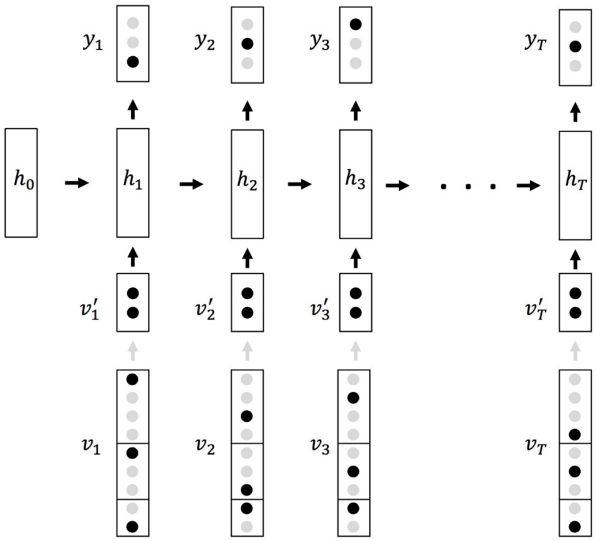

Figure 5.9. DKT Model with more features [150].

the DKT model is the input. Features are first discretized if they are continuous. Then these features are concatenated to form one input vector. However, this input vector is very big. Thus, they used Auto-Encoder [57] to learn a representation with much lower dimensionality, which is the v' as shown in the Figure 5.9. They evaluated this model on different datasets and the results show improved performance. However, the way they incorporate these features are pretty naive (simply concatenation). I believe more mature fusion techniques could be used to further improve the performance.

Yang *et al.* [143] proposed a similar work. However, instead of discretizing features first, they used a decision tree which could handle both continuous and discrete input. The decision tree will first take in all the features and then output a prediction about the next response. The predicted response is combined with the actual response to form a 4 bit one-hot encoding (1000, 0010, 0100, 0001 representing true positive, false positive, false negative and true negative respectively), as shown in Figure 5.10. This one hot encoding then is concatenated with the original one hot encoding of correctness and exercise tag to form the final input. They evaluated this technique on several datasets and the results also show improved performance compared with using only the correctness as input.

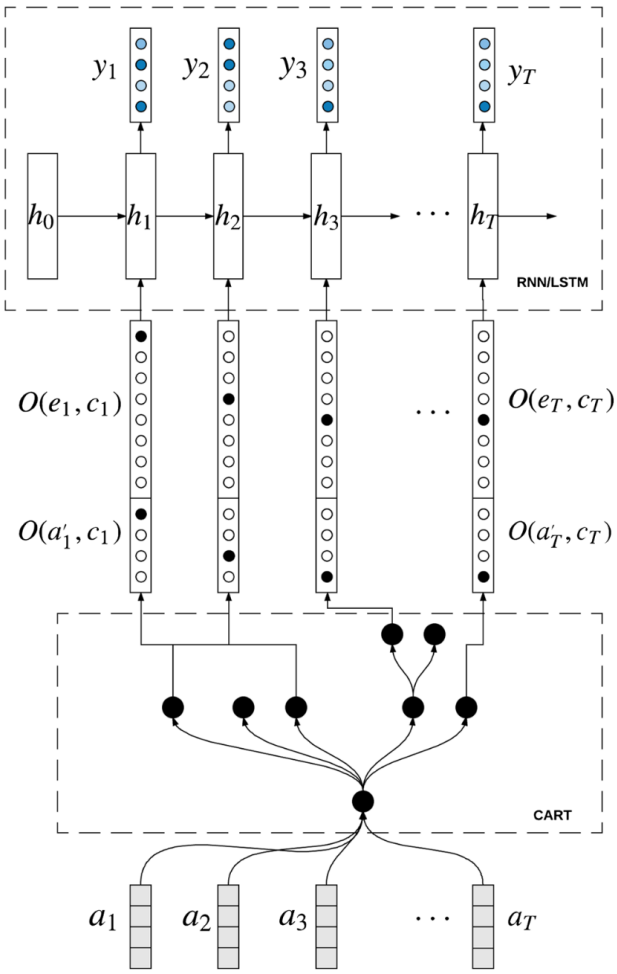

Figure 5.10. DKT Model with more features and decision tree classifier [143].

5.3.2. Recurrent Cell Search in The Case of One Modality

Before we move on to discuss more complicated situations using NAS for multimodality, let's first consider the case where there is only modality involved. Recurrent cell architectures like LSTM, GRU were proposed and evaluated in the context of sequence to sequence modeling (for example, neural machine translation). Although LSTM has achieved good performance for the task of knowledge tracing, we do not know if its architecture is optimal for knowledge tracing. Thus, I propose using NAS for automatic recurrent cell design for knowledge tracing. I take similar approaches as in [105] and the search space I use is similar to the one shown in Figure 2.9. The process of neural architecture search could easily take

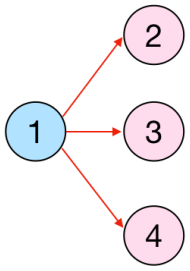

Figure 5.11. Discovered recurrent cell for knowledge tracing.

days or weeks, even on GPU nodes. Thus, I only use four nodes in our global graph in the experiments. To further reduce training time, I only train on sequences where students have fewer than 200 attempts. At each iteration, the controller samples a sub-graph representing one cell. After selection, a recurrent neural network with this cell is built and trained for one epoch. I split the dataset into two parts: 80% is used for training and 20% used for validation. After one epoch training, I calculate the reward using the validation dataset. The reward is inversely proportional to the validation loss. The cell discovered from NAS is shown in Figure 5.11. The output is calculated using the following equations:

$$
\begin{aligned}
c_1^t &= \text{sigmoid}(x^t \cdot W_0^{(x,c)} + h_0^{t-1} \cdot W_0^c) \\
h_1^t &= c_1^t * \tanh(x^t \cdot W_0^{(x,h)} + h_0^{t-1} \cdot W_0^h) + (1 - c_1^t) * h_0^{t-1} \\
c_2^t &= \text{sigmoid}(h_1^t \cdot W_{1,2}^c) \\
h_2^t &= c_2^t * \text{ReLU}(h_1^t \cdot W_{1,2}^h) + (1 - c_2^t) * h_1^t \\
c_3^t &= \text{sigmoid}(h_1^t \cdot W_{1,3}^c) \\
h_3^t &= c_3^t * \text{ReLU}(h_1^t \cdot W_{1,3}^h) + (1 - c_3^t) * h_1^t \\
c_4^t &= \text{sigmoid}(h_1^t \cdot W_{1,4}^c) \\
h_4^t &= c_4^t * \text{ReLU}(h_1^t \cdot W_{1,4}^h) + (1 - c_4^t) * h_1^t
\end{aligned}
$$

Here we have leaf nodes 2, 3, and 4. Their outputs are averaged and used as the final

Table 5.3. Dataset Statistics

	Records	Students	Skills
Assistment 09-10	338,001	3,884	123
KDD 2010	510,749	557	100

Table 5.4. Results Comparison

		AUC	R2
Assistment 09-10	LSTM	0.7378	0.1705
	NAS	**0.7411**	**0.1747**
KDD-2010	LSTM	0.8103	0.2462
	NAS	**0.8119**	**0.2545**

output. Thus

$$\text{output} = (h_2^t + h_3^t + h_4^t)/3$$

The dataset statistics are summarized in Table 5.3 and the results are shown in Table 5.4. As we can see, our found cell performs better than the LSTM cell on these two datasets, even though the improvement is not that large. I did McNemar's test on the predictions from these two models and the results are significantly different ($p < 0.01$).

After I derive the cell architecture, I train it from scratch using all training data available. However, I find that the model performance is slightly worse compared to the performance in the model search process. This might be caused by the fact I only used 20% data for validation and I picked the architecture with highest performance for this 20% data. In other words, our found architecture may overfit this 20% data. Another hypothesis is that the sub-model sampling process may act as a form of regularization. To test this hypothesis, I divide the training process into two phases. In the first phase, besides training our found cell, there is a probability (0.9) to sample a random sub-graph from the global search space. This probability decreases with epochs. In the second phase, I fix the model to be the one found as shown in Figure 5.11. This training process indeed improves the results. I am not certain if this type of regularization is suitable for other classification tasks and leave this

for future work.

This initial analysis of using NAS for automatic RNN cell design is encouraging. To reduce the training time, I used a relative small search space and was able to find at least as good cell architecture as LSTM for knowledge tracing. I also tried to increase the search space by including two more layers. However, I did not see improvements. That means more complicated architectures do not always improve performance. I hope this work has proved the promising potential of using NAS for knowledge tracing and will inspire more research work in this direction.

5.3.3. Fusion Structure Search for Multimodality

We have seen using NAS could help us design better cell architecture for knowledge tracing in the previous section. Now, let's consider the case of multimodal fusion. Existing works that utilize multimodality for knowledge tracing only considered the case of simple concatenation. It is known that simple concatenation makes it harder for the model to learn intra modality relations. Besides, not all modalities are useful for improving predictions and some of them may be just noise. This is particularly true in the case of knowledge tracing. All existing intelligent tutoring systems try to log as many information as possible. And these logged features are highly heterogeneous. However, it is unknown if all the logged data are actually helpful. So, great care need to be taken when combining these different modalities for knowledge tracing. The features considered in this study include the following:

- *Response*. This is a binary variable indicating either one correct or incorrect answer for the current problem. For models like DKT and DKVMN, this modality is the only one used.
- *Time spent*. The total time spent on the current problem. This is a continuous variable.
- *Attempts*. This feature indicates how many opportunities the current student has had for applying the associated skill.
- *Hints*. How many possible hints are there for this problem. This one is problem specific.

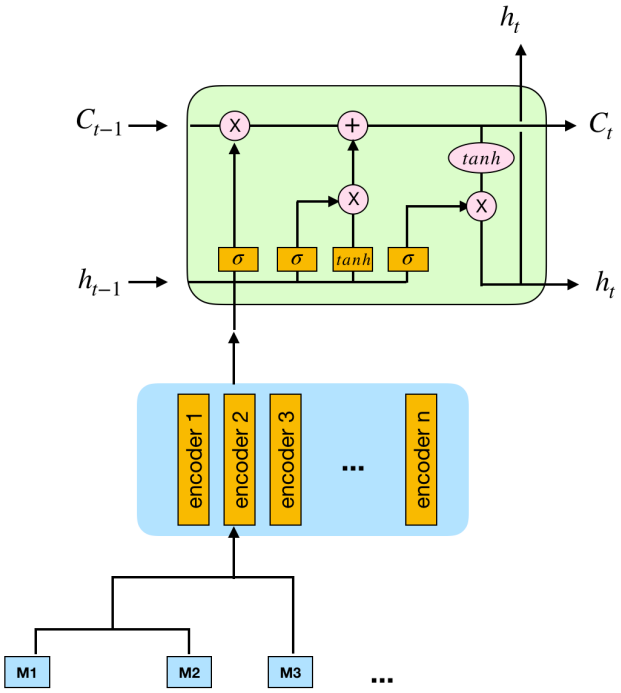

Figure 5.12. NAS for multimodality fusion with each skill has its own separate encoder.

- *First Action*. This is a binary variable indicating whether one student tries to attempt or ask for hint. If this student asked for a hint, the response is set to incorrect.

In this section, I consider the best way to fuse different modalities while using a simple recurrent cell like LSTM. The architecture is shown in Figure 5.12. Two significant differences from the DKT is that I use multimodality and we have a separate encoder for each skill. Piech *et al.* [107] also discussed to encode the skill id and correctness separately. However, they did not get as good results as using one-hot encoding. Also, they did not give the details about how they conducted separate encoding. In my proposed architecture. Each skill has a separate encoder, which is a fully connected layer. Using a separate encoder has at least two benefits here. Firstly, I argue that each skill is different from others at some degree, thus using a separate encoder allows the catch of uniqueness of that skill. Secondly, it is more convenient to combine more modalities. The final fused representation could just go through the corresponding encoder and there is no need to worry about one-hot encoding.

To make sure the performance improvement is due to the actual fusion structure, not because of the using of separate encoders, I did an experiment only using the response and separate encoders. I find the results are similar to the ones from DKT model.

Each modality will be first embedded using an embedding layer. Thus, there is no need to worry about if the input is discrete (responses) or continuous (time spent). I use embedding size 100, this is the same for all modalities. I consider the problem of which modalities to fuse at which order and what activation function to use. The architecture could be encoded as a list of sequence of numbers. For example, the fusion architecture [[m1, m2, f0], [m4, m3, f1]] could be interpreted as: Fuse modality 1 and modality 2 first using activation function 0 (say tanh). Then fuse in another modality 3 (here 4 means the fused representation of modality 1 and modality 2) using the activation function 1. This fusion architecture search process is also a features selection process. Thus, harmful or noisy features will be automatically filtered out. In the experiment set up, modalities *response*, *time spent*, *attempts*, *first action* and *hints* have corresponding indices, 0, 1, 2, 3 and 4. I used three activation functions relu, tanh and sigmoid, with corresponding indices 0, 1 and 2. Even though, the number of modalities and types of activation functions used in this study is small. The proposed methodology could be easily extended to any number of modalities and activation functions. Adding more modalities could be considered as gradually increase the complexity of the sampled space. Thus, I use SMBO instead of reinforcement learning in this study.

5.3.4. Extending sub-graph Sampling to Multimodality

In the previous section. I applied NAS for multimodal fusion search while keeping the cell architecture fixed (LSTM). Now I consider the possibility of combining multimodal fusion search with architecture search. Most of the existing research works involving NAS focus on architecture search for one modality [93, 105, 108]. There are few works that combine multimodal fusion search and architecture search within one methodology. One exception is the work from Pérez-Rúa *et al.* [104]. However, they only used two modalities. They used two pre-trained networks for two different modalities. Each modality is represented

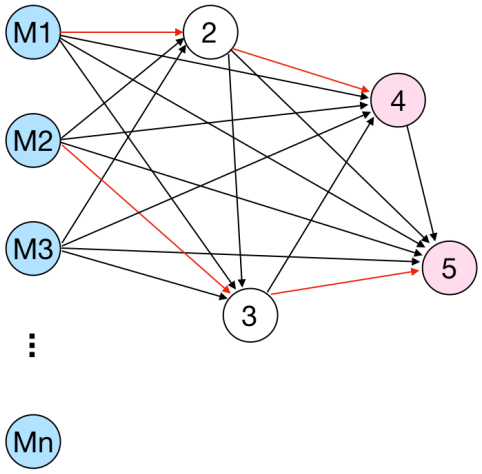

Figure 5.13. Extend sub-graph sampling to multimodality.

by a pre-trained multi-layer neural network. Their goal is to extract representations from different layers from these two modalities and fuse them in a specific order to achieve good prediction performance. Strictly speaking, their work is representation search from two different modalities, and not modality search.

NAS is very computational demanding for one modality alone, needless to say adding more modalities. In this and the following section, I discuss how could we combine multi-modal fusion search and architecture search within one methodology. Right now, let's focus on how can we extend the sub-graph sampling process discussed in section 5.3.2 to multi-modality. Our proposed architecture is shown in Figure 5.13. Similarly, each node (layer) is fully connected with its previous nodes and it could choose the input from any of these previous nodes. The difference is, instead of one input node, we have multiple input nodes with one for each modality. Again, I first apply an embedding layer for each modality to get a representation easier for further processing. We can see this search space is a superset of the one we discussed in section 5.3.2. Within this methodology, any amount of modalities could be fused at any layer. The sampled architecture could also be represented by a list of sequence of numbers [[p1, a1], [p2, a2],...]. p_i refers to the previous node and a_i refers to the activation function. The red arrows in Figure 5.13 indicates a sampled architecture.

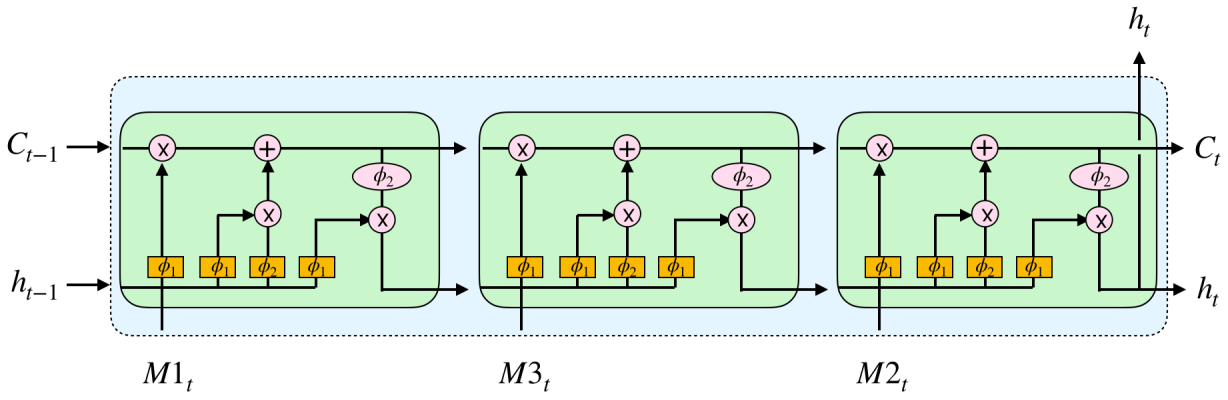

Figure 5.14. Extend LSTM cell for Multimodality.

Modality 1 is used as input to node 2 and then further node 4. Modality 2 is used as input by node 3, then further by node 5. The outputs from node 4 and 5 will be combined as the final output. This sampled architecture could be encoded as [[1, 0], [2, 0], [2, 1], [3, 1]] (We can not see the activation functions used in Figure 5.13). However, one limitation of this architecture is that each node only takes one input from its previous layers. That means Modality 1 and modality 2 could not be used at the same time by node 2.

5.3.5. Extend the LSTM Cell for Multimodality

The third attempt I tried is to extend the LSTM cell for multimodality. My proposed architecture is shown in Figure 5.14. I use the architecture similar to LSTM. However, instead of using the sigmoid and tanh activation functions as in LSTM, I make them tunable. I use one such LSTM variant unit for one modality. All the modalities share the same memory cell c and hidden state h. The search process is to find in which order to fuse what modalities and what activation functions to use. Similarly, one sampled architecture could be represented as a list of sequence of numbers. For example, [[1, 2, 0], [2, 2, 2]] represents the architecture that first incorporates modality 1 and use activation functions 2 and 0 for ϕ_1 and ϕ_2. Then incorporate modality 2 and use activation functions 2 and 2 for ϕ_1 and ϕ_2. However, I did not see significant improvements using this methodology. More research might be needed to

understand if this methodology is useful; however my initial investigations did not support this methodology as promising. Thus, I will not further discuss this model in the results section. But I hope this could still inspire researchers from other communities.

5.3.6. A New Metric for The Measurement of Performance With Time

Area under the curve (AUC) and coefficient of determination (r^2) are two standard metrics used for knowledge tracing. However, neither method is designed to be used to measure how one model performs with time. That is, does the model perform better after it sees more responses from one student? Thus, I propose to use weighted AUC (wAUC) for this purpose [82]. For a series of items to be predicted $i \in \{1, ..., n\}$. There is a series of corresponding weights $\{w_1, w_2, ...w_n\}$. Let $S_1 = \{i : y_i = 1\}$ be the set of positive examples. $S_0 = \{i : y_i = 1\}$ be the set of negative examples. Weighted false positive rate and weighted true positive rate could be defined as follows:

$$FPR = \frac{1}{W_0} \sum_{i \in S_0} I[y_i = 1] w_i$$

$$TPR = \frac{1}{W_1} \sum_{i \in S_1} I[y_i = 1] w_i$$

where, $W_0 = \sum_{i \in S_0} w_i$ is total negative weights and $W_1 = \sum_{i \in S_1} w_i$ is the total positive weights. Thus, weighted ROC curve could be plotted for all thresholds. Weighted AUC could be used in cases in which we want to emphasize low false positive rate, etc. In our case, to measure how the knowledge tracing model performs with time. We could assign more weights to those most recent responses. One simple strategy is to increase the weight by one each time the student sees the same skill. For example. If one student sees the following sequence of skills

$$22, 22, 3, 3, 3, 22, 99$$

Table 5.5. Dataset Statistics for Multimodality

	Records	Students	Skills
Assistment 09-10	337,236	3,884	123
Oli Engineering Statics 2012 Fall	344,403	566	1251

The corresponding weights will be

$$1, 2, 1, 2, 3, 3, 1$$

This strategy of weight assigning is simple, but is also sensitive to measures that occur later in the sequence. I believe more mature weights assigning strategies using domain knowledge could be investigated. However, as an initial analysis, this simple metric and weight assigning strategy allows us to measure how one model performs with time. I hope this work could inspire more knowledge tracing model measurement research in the future.

5.3.7. Results

It is difficult to perform a fair comparison between two different deep neural network models for a number of reasons. Firstly, we only have a limited understanding of how neural networks function or make exact use of training data. Secondly, techniques like batch normalization [61], weights initialization, learning rate, regularization, selection of optimizer, and many others can have a significant impact on the performance of models. Can one model with a novel weight initialization technique be considered a different model? What exacerbated this problem is that one set of hyperparameters that works well on one dataset might not work well on another dataset. To explore all possible hyperparameters combination is also impractical. A recent paper [13] states that a simple model with careful picked settings could achieve better results than many deep neural network models. Thus, the success of complicated models have might due to the fine tuning of hyperparameters, not because they have better decision strategies. Machine learning model evaluation is a complicated research

Table 5.6. Comparision of different models for knowledge tracing. SM stands for Single Modality. MM stands for multimodality. SC stands for simple concatenation. FS stands for fusion search.

		Assistment 09-10			Oli Statics 2012		
		r2	AUC	wAUC	r2	AUC	wAUC
SM	DKT(Baseline 1)	0.1628	0.7326	0.7367	0.4106	0.8819	0.8622
	DKVMN (Baseline 2)	0.1507	0.7299	0.7354	0.3557	0.8793	0.8614
	NAS cell	0.1678	0.7364	0.7408	0.4169	0.8844	0.8661
MM	DKT + SC	0.1743	0.7371	0.7441	0.4316	0.8884	0.8734
	DKT + FS	**0.1844**	0.7454	0.7493	0.4239	0.8863	0.8651
	NAS Extend	0.1829	**0.7458**	**0.7545**	**0.4348**	**0.8902**	**0.8779**

topic, until a better standard is proposed in the machine learning community. In my work, to make a fair comparison, I follow the conventions from other researchers. I did a reasonable amount of hyperparameters tuning (limited only by time and available computation) for the baseline models used in this study (DKT and DKVMN) and I limit the tuning parameters to learning rate, weight decay, the epilson value of adam optimizer. However, I would not claim that I am using the best hyperparameters combination and I believe it is always possible to improve these numbers shown in Table 5.6 by tweaking a little bit more on these hyperparameters. Thus, I hope readers will focus more on the architecture innovations and other contributions. Despite the fact that there is no significant test on most deep neural network papers, I did McNemar's test on the predictions. Although these significance testing tools have their own limitations [32]. And some of the underlying assumptions might be violated. For example, the dataset I used is not i.i.d data (independently and identically distributed data). They are more like time series data.

Since the process of neural architecture search is highly time consuming and we have two different datasets. The methodology I took is I use the assistment 09-10 dataset for the finding of best architectures, then I apply these architecture to the Oli Engineering Statics dataset. In other words, I did not perform architecture search for the Oli Engineering Statics dataset. The statistics of the datasets used are shown in Table 5.5. The found architecture

using architecture search is evaluated on the dataset only. I believe the performance might be further improved if I conduct NAS on this specific dataset. Another reason of taking this approach is I want to see if the architectures found in one dataset could also generalize to another. In other words, I want to see if some modality fusions are helpful when making predictions in unseen datasets.

The results are shown in Table 5.6. As we can see, incorporating more modalities does help improve the performance. I performed Mcnemar's test (with $p < 0.01$) on the predictions from the NAS Extend model and the DKT model. The results show significant difference. However, we also notice that the improvement is not very big. One possible explanation is that both datasets are noisy, which makes it hard for the model to learn meaningful relations among modalities. For example, for the *time spent* feature, there are some transactions with *time spent* values less than 1 second. Thus, I argue better procedures for collecting these data for future research. Besides, it seems the improvement is more obvious on the assistment 09-10 dataset than on the Oli Statistic 2012 Dataset. This could be explained by the fact that I used assistment 09-10 for NAS. In short, from the results, the found architectures do generalize to the Oli statics dataset. Besides, we could always run the same methodology separately on a new dataset to see if that could further improve the performance (but we need to consider the tradeoff here, since this process is computational demanding). Because it is possible, due to settings, one modality may have more signal than another in different datasets. In this case, NAS may return different architectures for two different datasets. Readers should be aware of the difference between the generalizability of the models discovered by the methodology and the generalizability of the methodology itself.

My found best model for DKT + FS is [[1, 2, 2], [3, 4, 2], [0, 5, 1]]). Intuitively, this result can be interpreted as combine *time spent* and *attempts* first using activation function *tanh*, then merge in *first action* using activation function *tanh*. At last, merge in the *response* using activation function *sigmoid*. The best model for NAS extend is [[3, 1], [0, 0]]. This model means using the *first action* as the input for node 1 and use *sigmoid* activation

function. Then, use the *response* as the input for the node 2 and apply activation fucntion *relu*. The output of the node 1 and node 2 will be combined to generate the final output. From the discovered best architectures, both the *response* and the *first action* are selected. I also noticed the performances of these architectures decrease significantly if I exclude the *response* modality. This makes intuitive sense, since a correct response indicates that there is a high probability that this student has mastered the corresponding skill. And the *response* is the only modality used for models like DKT and DKVMN (I do not consider skill id as a separate modality). The modality *first action* is also selected for two models. This modality indicates if one student chooses to answer the problem or takes other actions, for instance, ask for a hint. If one student asked for a hint, the response will be automatically marked as incorrect. Thus, there are two different situations that may result in an incorrect response and we should be aware the difference. In one situation, student A tried to answer the problem but got an incorrect response. On the other hand, student B asked for a hint, thus resulted in an incorrect response. This *first action* modality might reflect the different confidence levels of these two students for some skill. Our model may have learnt assigning different mastery level probabilities for these two situations. Another observation is that using all modalities available might worsen the performance. This is especially true if the dataset itself is noisy. I evaluated the proposed models on two different datasets from the domains of mathematics and engineering separately. There are two reasons why I chose these datasets. Firstly, it is easier to define a skill in these domains. For example, we can consider calculating the area of a circle as one skill. It is much harder to define such skills in other domains like literature, where there usually have lots of open ended questions. The second reason is these two public datasets are most often used, so it is easier to compare my proposed models with others. However, I believe my proposed models are generalizable to domains as long as skills could be precisely defined.

Chapter 6

ATTENTION MECHANISMS AND KNOWLEDGE TRACING

To motivate the methods used in this chapter, I make parallels to the human visual system. There are two modes of our visual processing [81]: Focal vision and Ambient vision. Focal vision allows us to see an object clearly, but is limited to a small central area (usually two degrees). Ambient vision is usually 'blurry', but it helps us be aware of our surroundings and is used for navigation. Take reading as an example, there are only several words that are clear to us one at a time (Focal vision), but we are also aware of the whole paragraph and know where to look for next words (Ambient vision). Similarly, when we look at an image of dog, we automatically focus our eyes on the dog with 'high resolution' while ignoring the background with 'low resolution'. Both focal vision and ambient vision play important roles in our life. Attention mechanisms are loosely inspired by how our vision system works. The idea of attention mechanisms is not new, but we have seen its renaissance in recent years with the popularity of deep learning [10]. The attention mechanism allows our model to focus on the most important information for inference by assigning different weights to different representation dynamically. For an object recognition task, it allows a model to focus on the objects rather than on the background. For a translation task, it allows a model to focus on the most relevant words when decoding the next word. From previous discussions in chapter 3, I have shown some limitations of the DKT model and discussed the attention mechanism as a potential way to mitigate these issues. Therefore, in this chapter, I am seeking to answer the following question:

- Does adding attention mechanisms improve the performance of deep neural network based knowledge tracing models?

In section 6.1, I will first introduce the motivations, the problems of current deep neural network models, and different kinds of attention mechanisms. In section 6.2, I investigate two

possible ways of applying attention mechanisms for deep neural network based knowledge tracing; followed by discussion of the results and limitations.

6.1. Attention Mechanisms

Why do we need attention mechanisms? What problems does our current model have? Figure 6.1 shows a sequence to sequence machine translation model. For simplicity, I am not showing the embedding layer (which is often required for a translation task) and other details. This model consists of an encoder and a decoder. The encoder transforms the input sentence from one language into some high dimensional representation. The decoder then transforms the high dimensional representation into an output sentence in another language. The assumption is two different language sentences with the same meaning share the same representation in some high dimensional space. In Figure 6.1, both the encoder and the decoder are recurrent neural networks, as introduced in chapter 2. Thus, the input and output sentences could have different length. At each time step, one word is encoded into the hidden state and passed to the next step. The last hidden state of the encoder, which is also called the context vector, is used as one of the input for the decoder. We assume this hidden state vector will be a good summary of the input sentence. However, it usually does not work very well when the input sentence becomes very long, especially for a vanilla recurrent network because the information about the initial words could be overwritten (or forgotten). Variants of recurrent neural networks have been proposed including the Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) [22], trying to solve this long dependency issue. The cell architectures of these models do allow them to store more information, thus improve the overall performance. However, they did not solve the main problem, so methods of using the entire encoder outputs were proposed, with attention [59]. In other words, for the decoder to work well, it should have access to the whole input sentence (but with different attention/weights), not just the last hidden state of the encoder. Attention mechanisms are also widely used in image processing tasks [141]. In such cases, convolutional neural networks (CNNs) are usually used instead of recurrent neural networks. These models used attention

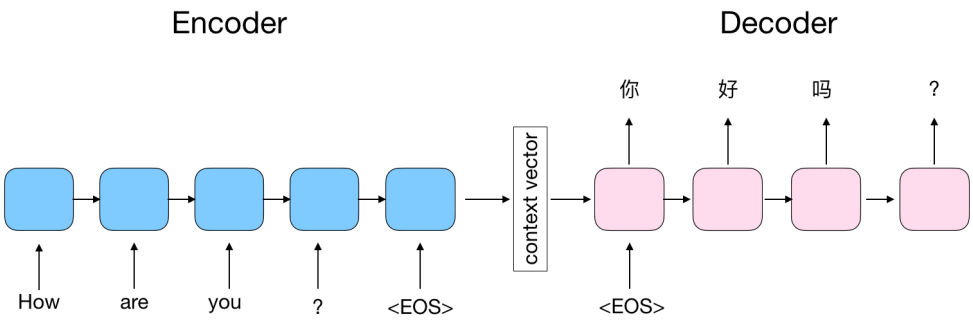

Figure 6.1. Sequence to sequence modeling with encoder and decoder. Here <EOS> marks the end of a sentence.

in the CNN model to focus on the most informative regions of an image by assigning more weight. In this study, I mainly use recurrent architectures for knowledge tracing, thus I will not further expand upon the details of attention mechanisms for CNNs.

6.1.1. Attention for Recurrent Neural Networks

I now propose how can we use attention mechanisms to solve the problem proposed in the previous section. The proposed attention mechanism is not specific to machine translation task, but generalizes to any model that uses a recurrent architecture. When building a translation model like the one shown in Figure 6.1, the next prediction y_t is usually a function of the hidden state of the decoder s_{t-1} and the previous output y_{t-1}, $y_t = f(s_{t-1}, y_{t-1})$. s_0 is the context vector from the encoder. The problem here is all the information from the input sentence is stored in s_t. A natural improvement that we could think of is to make the function $f(s_{t-1}, y_{t-1})$ have access to the whole sentence when making a prediction. We could add another parameter to this function $y_t = f(s_{t-1}, y_{t-1}, c_t)$. Here c_t is a vector that will be dynamically calculated for each time step when making a prediction. This is actually the approach taken in [10]. Their model is shown in Figure 6.2. A bidirectional LSTM [49] is used as the encoder in their paper. However, using any single one directional recurrent neural network will also work. A bidirectional LSTM is just putting two single

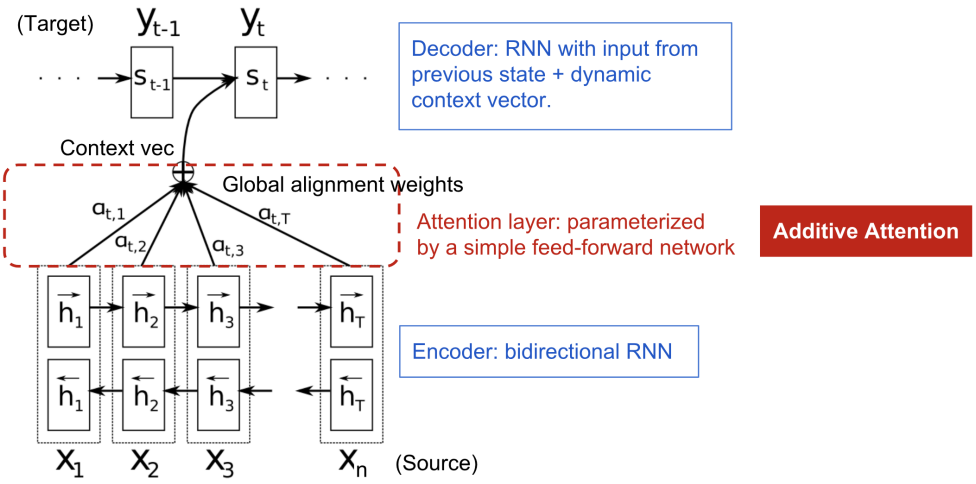

Figure 6.2. The encoder-decoder model with additive attention mechanism [133].

LSTM together, thus the model will have access to both forward and backward information. It is commonly used for language modeling. For bidirectional LSTM, a new hidden state is formed by concatenating the forward and backward hidden states.

$$h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}] \tag{6.1}$$

The extra c_t in the function $y_t = f(s_{t-1}, y_{t-1}, c_t)$ is dynamically calculated as a weighted sum of all h_t. Here, we use h_t to denote the hidden state of the encoder, use s_t to denote the hidden state of the decoder. The key point of the attention mechanism is to calculate a vector of weights. Thus, $c_t = \alpha_{t,i} h_i$. The equation used in [10] is shown below:

$$\alpha_{t,i} = \frac{exp(score(s_{t-1}, h_i))}{\sum_{j=1}^{n} exp(score(s_{t-1}, h_j))} \tag{6.2}$$

The score function is used to measure how 'related' the most recent hidden state of the decoder s_{t-1} is with all the hidden states of the encoder. Different attention mechanisms

usually differ in the choice of the score function. In [10], they used a feedforward neural network.

$$score(h, s) = v_a^T tanh(W_a[h, s]) \tag{6.3}$$

where v_a and W_a are learnable parameters. It is interesting that meaningful weights can be learnt using simple iterative optimization process and optimizers like Stochastic Gradient Descent (SGD) [112]. Different kinds of attention mechanisms are proposed in recent years [21, 86, 129, 141]. They usually differ in the choice of the score function and its parameters. I will describe some of the most recognized ones in following sections.

6.1.2. Self Attention

The first thing we need to consider when using attention mechanisms is, in a machine learning language, what two vectors are related and should be used for the calculation of weights. In other words, what are the parameters of the score function. In a machine translation task, we have a sequence of words as our inputs. Our target is another sequence of words in a different language. Thus it makes sense to use the hidden state of source input h_t and hidden state of the decoder s_{t-1} to calculate the score. We assume different words in the input sentence contribute differently when generating the current output word. However, if we only want to learn the correlation between the current word of a sentence and the rest, we could replace s_{t-1} with $h_\tau, \tau \neq t$. We call this self attention and it has proved quite successful in tasks like machine reading and image captioning. Figure 6.3 shows the correlation between the current word and the words in memories using self attention in a machine reading task [21]. Color red represents the current word, color blue indicates the degree of activation. As we can see, words like 'The', 'is', 'a' usually have lower activations. This means, these words have low correlation with the current word. On the other hand, verbs like 'chasing' or descriptive words like 'criminal' are highly correlated with the current word.

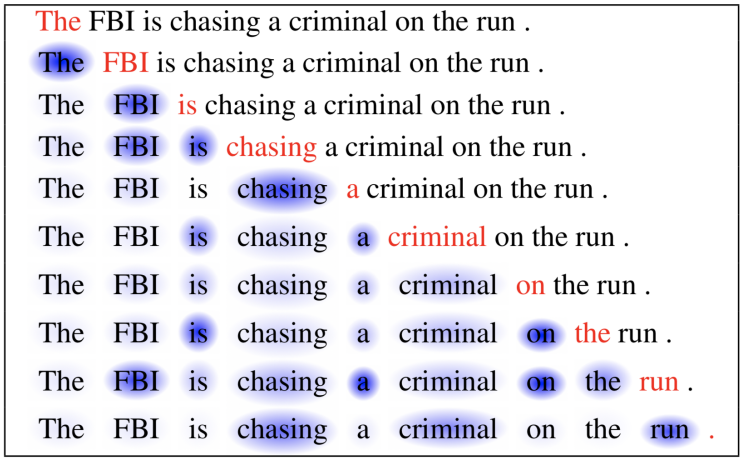

Figure 6.3. Color red represents the current word, Color blue indicates the degree of memory activation. [21]

6.1.3. Soft/Hard Attention

One advantage of most deep learning models is that they are differentiable, which means they could be trained end to end using algorithms like backpropogation. In the case of machine translation, to achieve differentiability when applying attention mechanisms, we usually calculate the weights for all relevant states. In other words, it attends to all the words (hidden states) of the input sentence for a translation task. For example, in Figure 6.2, all the hidden states h_t have been assigned weights. For image related tasks, weights are calculated for all pixels. This kind of attention mechanism is called soft attention. Soft attention is mathematically convenient and can be easily represented as matrix multiplications. However, one disadvantage is that it is very computational demanding if the input sequence is very long or the image is high resolution. A more serious problem is weights might be forced to be assigned to noisy hidden states. In contrast, hard attention only calculates the weights for a subset of the input, thus saving computational power. Moreover, it allows us to focus on the portion that is important to us. One limitation of hard attention is the whole model usually can not be trained end to end. They are usually trained using algorithms like REINFORCE [137]. Xu *et al.* used a stochastic method to select which image patch to focus

on when generating the next word [141]. Another similar concept to soft/hard attention is called global/local attention [86]. Global attention attends to all pixels or words. On the other hand, local attention attends to a fixed size window. For neural machine translation tasks, the window could be the latest k words used for translation. Both global and local attention are differentiable, thus could be trained end to end. For a more comprehensive discussion of attention mechanisms, readers are referred to this survey, which provides an excellent discussion of the recent trends and methods [133].

6.1.4. The Transformer

Attention is designed to solve the long range dependency problem; in its simplest form, it is a vector of weights. In this work, I make use of the attention mechanism applied to recurrent network—however recent work has shown that the recurrent structure, for language understanding, is not necessarily required for good performance. While I do not employ this methodology in the current work, its discussion is warranted in the context of attention models. Most existing works use attention mechanisms together with recurrent neural networks or convolutional neural networks. A recent work from Vaswani *et al.* [129] proposed the Transformer model and demonstrated that only using attention mechanisms, dispensing any recurrent and convolutional architectures, can achieve superior results. At the same time, this approach takes less time to train because it naturally can be computed with parallelism. The Transformer model consists of an encoder and decoder, just like other sequence to sequence model. However, there is no recurrent or convolutional component in this architecture. Both the encoder and decoder are a stack of mixed attention layers and feed-forward layers. The Transformer architecture uses multi-headed attention mechanisms in which three matrices are used to map the input into 'Value', 'Key', and 'Query' vectors. These matrices are jointly trained with the entire model.

The Transformer model has no recurrence structure. Instead, it uses positional encoding to make use of the order of sequence. They incorporate sin and cos functions into dimensions

of the embedding as shown below:

$$PE_{pos,2i} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{pos,2i+1} = cos(pos/10000^{2i/d_{model}})$$

where *pos* is the position, i is the dimension. No recurrence structure makes parallel training possible, thus dramatically reducing the training time. The state of art language model BERT [31] is also built upon the Transformer model. Because this methodology is not used in my current research, I do not expand on additional details for Transformer networks.

At this point, we have a general understanding of what problems recurrent neural networks faced and how attention mechanisms have helped in solving the long time dependency issue. I also introduced different types of attention mechanisms. In the next section, I will discuss possible ways of applying attention mechanisms for knowledge tracing. Even though it is possible to dispense the recurrence structure like the Transformer model when applying attention, most existing deep neural network based models for knowledge tracing still use recurrence structure. My goal in this study is to see if adding attention mechanisms could further improve the performance of these models. Thus, I will not consider using the Transformer in this work, which would be proposing a totally new model rather than answering our attention-based research question. I leave investigation of the Transformer architecture in knowledge tracing to future work.

6.2. Attention Mechanisms and Knowledge Tracing

There are two strategies to apply attention mechanisms when involving both recurrent neural networks and multimodality. One strategy is that attention mechanisms might help the model learn which modalities are more important than others. Combining attention mechanisms and deep multimodal learning has been explored in other fields like video classification, video description [59, 85]. Hori *et al.* [59] use attention for multimodal fusion in the context of video description. Previous works on video description used simple concatenation.

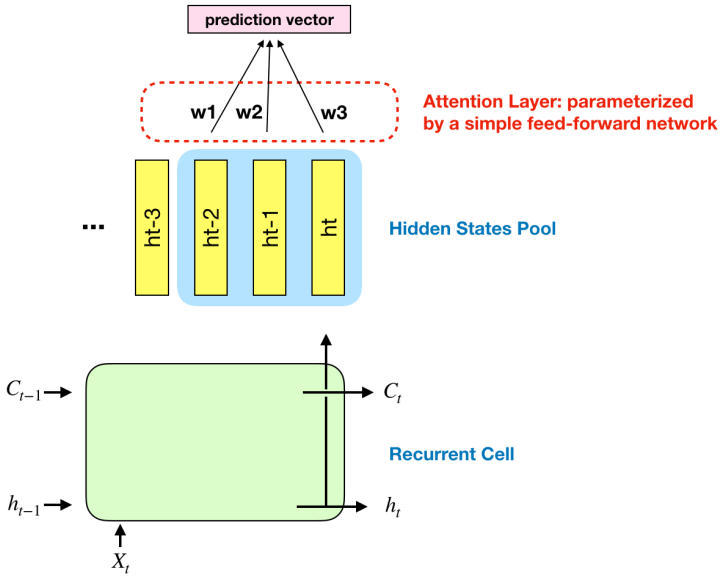

Figure 6.4. NAS Extend Model with Attention Mechanism. The recent hidden states are cached in the pool.

Hori *et al.* argued that different modalities may carry different task relevant information at different times. For instance, two musical instruments may look similar, but will produce very different sounds. In this case, we probably should give more weights to the voice modality. On the other hand, when a person is brushing their teeth, the watering sound may be just noise and should be assigned less weights. Another strategy is to use attention mechanisms to overcome the long time dependency issue. In Chapter 3, I explained why deep knowledge tracing might have less depth than anticipated. Instead of tracking each skill, the DKT model is more likely learning an 'ability' state. Once in an oracle state, the model will predict all skills correct. This behavior is partly due to the fact that RNN can not adequately track information across long sequences. Thus, I hypothesize that allowing the DKT model to access the history performance might help improve the performance.

In the previous chapter, I used NAS to automatically look for which modalities are most useful when making predictions. Thus, I assumed unimportant modalities would be automatically 'filtered' out in this process. However, another thinking is that modalities used

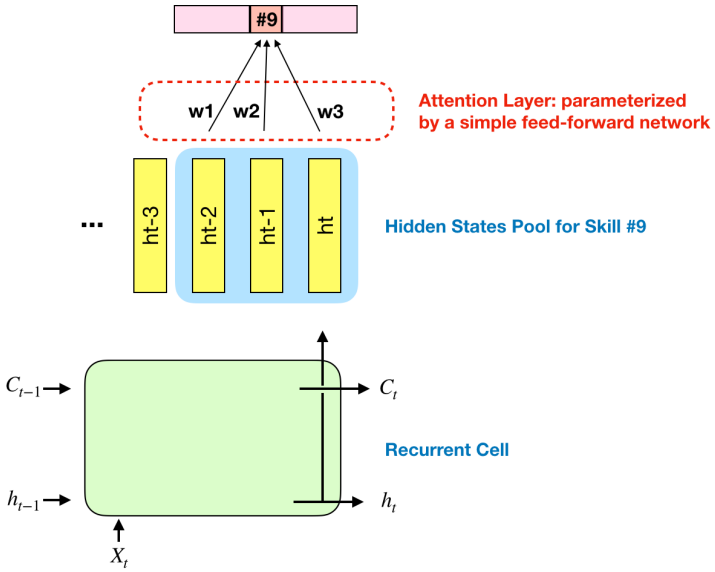

Figure 6.5. NAS Extend Model with Attention Mechanism. The recent hidden states for skill #9 are cached in the pool.

in the knowledge tracing context are 'thin' compared with those used in video classification. For example, an image as one modality could include very rich information. On the other hand, modalities like *time spent* may only narrowly reflect a 'confidence level'. Thus, in this study I take the second strategy, which is to use attention mechanisms to overcome the long term dependency issue.

Before discussing my proposed architectures, let's first take another look at the DKVMN model. There are two memories involved in this model. The key memory M^k is an embedding of all skills and is immutable. The value memory M_t^v stores the mastery level information for different skills and is updated for each iteration. When there is a query, q_k, it will first be multiplied by an embedding matrix A to get an embedding vector, k_t. Then, a weight vector is calculated $w_t(i) = Softmax(k_t^T M^k(i))$. This is also called location based attention [86]. The learnt attention vector w_t represents the correlation between current skill and other skills. Thus such attention mechanisms could not solve the long time dependency issue. It might be effective when these skills are highly correlated and making one prediction

requires the knowledge of other skills. As we already saw in Table 5.6, this kind of attention mechanism did not improve results.

Thus, a natural step to improve performance is to allow one model to access previous hidden states. My first proposed architecture is shown in Figure 6.4. At the bottom of this architecture could be any recurrent cell, including the DKVMN model. In this work, I use the best architecture found in the previous studies, the *NAS extend* model. The hidden states are cached in a fixed size pool. All these cached hidden states are used for the decoding of the prediction vector. This is similar to the idea of local attention [86], thus our model is differentiable and could be trained end to end. The weights are automatically calculated using an attention layer (a feedforward neural network).

$$w_t = tanh(W_a[h_t]) \tag{6.4}$$

where W_a is a trainable matrix. The context vector for decoding the prediction vector is then a weighted some of these cached states.

$$c_t = \sum_t w_t h_t \tag{6.5}$$

Here we rely on the attention layer solely to generate reasonable weights. The design consideration behind this model is that adding the history performance of one student might improve the predictions. Different pool size $m \in [3, 5, 10]$ are investigated. However, none of these investigations significantly improved the model performance. My first assumption is that perhaps I need to use hidden states from the beginning, since the latest hidden state might already include all the information about the recent performance. Thus hidden states with distances like $h_t, h_{t-d}, h_{t-d*2}...$ are used in which $d \in [10, 20, 50]$ is another hyperparameter. Unfortunately, this change does not improve the performance.

My second attempt of applying attention mechanism is shown in Figure 6.5. The difference from the previous model is that, in this model, instead of using the general cached hidden states for decoding, I use cached hidden states for specific skills we want to predict.

Table 6.1. Comparision of different models for knowledge tracing. SM stands for Single Modality. MM stands for multimodality. SC stands for simple concatenation. FS stands for fusion search. A stands for attention.

		Assistment 09-10			Oli Statics 2012		
		r2	AUC	wAUC	r2	AUC	wAUC
SM	DKT (Baseline 1)	0.1628	0.7326	0.7367	0.4106	0.8819	0.8622
	DKVMN (Baseline 2)	0.1507	0.7299	0.7354	0.3557	0.8793	0.8614
	NAS cell	0.1678	0.7364	0.7408	0.4169	0.8844	0.8661
MM	DKT + SC	0.1743	0.7371	0.7441	0.4316	0.8884	0.8734
	DKT + FS	**0.1844**	0.7454	0.7493	0.4239	0.8863	0.8651
	NAS Extend	0.1829	0.7458	0.7545	**0.4348**	**0.8902**	**0.8779**
	NAS Extend + A	0.1842	**0.7465**	**0.7548**	0.4315	0.8894	0.8767

For example, if we want to predict skill 9, the cached 'latest hidden states for skill 9' will be used. This means we have one pool for each skill. Different pool size $m \in [3, 5, 10]$ are tried. However, the improvement is still limited. In Table 6.1, we can see it only improve the results by a small average margin on the assistment 09-10 dataset. And it gives slightly worse results on the Oli Engineering Statics dataset. This inconsistency in performance leads me to conclude that attention mechanisms are not well suited for the application of knowledge tracing (at least using the proposed methodologies).

One explanation why attention mechanisms help in neural machine translation task is that the words in a sentence have strong correlations. In other words, they follow the rule of grammar. However, the sequence of question answering attempts from one student does not follow any such rule. The attempts might follow some curriculum, but the choice of 'which problems to solve' seems quite flexible. Moreover, the sequences of attempts from different students are highly heterogeneous. In such cases, attention mechanisms might not be able to learn meaningful weights. Another explanation is related to DKT model architecture and its variants. These models might not be able to effectively utilize history for predictions. Deep neural network based models can easily get stuck in local optimal, thus it is possible these architectures are in a location that could only use the latest hidden state for decoding. We

already seen in chapter 3 that DKT model and its variants might not be able to distinguish different skills, which also might explain why using a separate cache pool for different skills does not help. Thus, further research is needed to see if attention mechanisms could help in the case of deep neural networks based knowledge tracing. Even so, in the absence of new research, we conclude that the gains from using attention in knowledge tracing are limited.

Chapter 7

CONCLUSION

7.1. Conclusion

In this thesis, I investigated deep neural network based models for student response modeling. There are two goals in this thesis: 1) To have a better understanding of deep neural network based student response modeling through visualization and through incorporating uncertainties. 2) To investigate the performance of student response modeling with multimodality and attention mechanisms. In this chapter, I summarize the main studies that have been conducted and intuit their conclusions.

Many researchers, especially those not entrenched in the machine learning community, use deep neural networks as black boxes. However, a clear understanding of how these models make decisions is highly important. Thus, I proposed interpretable deep neural network models for student response modeling. First, I proposed using a post-hoc approach to interpret the DKT model through visualization. I concatenated the intermediate outputs from the DKT model, forming the activation vector. Both synthetic and real data were used as input to analyze the behaviors of DKT in high dimensional space. Through visualization of the activation vector, we observed that the DKT model might not be tracking each different skill. Instead it is more likely learning an 'oracle state' and, once in this oracle state, one student has very high probabilities of getting all skills correct. One student could use any practice skill to get into this oracle state, which is not a desirable behavior of the model. Also, the learnt high dimensional representation might not be meaningful. Some possible ways of improving the DKT model are also discussed.

The second approach to interpretable deep neural networks is to make the model learn an uncertainty score for each prediction. Thus, decisions could be left to human beings

when the uncertainty is high. I achieved this goal through extending Kendall's work [67] by regularizing the loss function. Two types of uncertainties: aleatoric uncertainty and epistemic uncertainties are discussed. My proposed model was able to achieve comparable performance as the DKT model, while at the same time providing meaningful uncertainty scores. To quantitatively analyze the uncertainties, I also proposed using synthetic data in which we have access to the actual probabilities that generated these responses. I also argued that if two models have similar overall performance, the one whose performance gets better with time is preferred. Because of this, a new metric for weighted AUC, which could measure how one model performs with time, was also proposed.

I also investigated using multimodality to improve the performance of student response modeling. As an initial study, I presented the EduAware system which is a tablet based adult learning module. Interactive features like swipe speed, time spent, click duration, etc. were used to predict the student performance on a final comprehensive test. I found that adding more features significantly improved the performance. An analysis of which features are important for making predictions was also provided. However, one limitation is that the dataset used in EduAware is small, thus only simple machine learning models like logistic regression and linear support vector machines are evaluated. To investigate multimodality in the context of deep learning, two public datasets with millions of records were employed. Firstly, I discussed the using of neural architecture search when there is only one modality. Using reinforcement learning and parameters sharing technique, a recurrent cell is discovered. The discovered recurrent cell achieved superior performance compared to the traditional LSTM cell. Statistical test shows the results are significantly different. Furthermore, I proposed three methods for using neural architecture search in the context of multimodality. First, I used NAS to automatically look for the best fusion structure while keeping the recurrent cell fixed (LSTM). Second, I extended neural architecture search to the case of multimodality, automatically looking for the best recurrent cell and modalities combination for knowledge tracing within one methodology. Third, an attempt to extend the LSTM cell for multimodal fusion is also discussed. The discovered architectures perform

better than both the DKT and DKVMN models on the two public datasets. Limitations of proposed methods are also discussed.

Finally, I investigated the attention mechanisms for deep neural network based knowledge tracing models. Various types of attentions mechanisms were discussed. Two possible ways of applying the attention mechanisms to knowledge tracing were evaluated. However, I found the proposed attention mechanism is inconsistent, only improving the performance in one dataset by a small margin. I hypothesized how this might be due to the heterogeneous fact of student response data. Thus, more research is needed for applying attention mechanisms for knowledge tracing.

7.2. Contributions to Other Research Fields

I described the contributions of this dissertation in section 1.5. Even though the proposed studies are conducted in the domain of student response modeling, some of which might be inspiring for researchers from other fields.

- Use visualization to analyze the behaviors of deep neural network models [33]. The use of synthetic data allows the analysis of behaviors of deep neural networks in the extreme conditions. These behaviors might not be accessible using real world data and might help us understand how decisions are made.

- I proposed a regularizer that could guide the training process to result in sensible uncertainties [35]. In this study, I discussed the issues of existing methodologies. These theoretical discussions are domain independent. And I believe the proposed approach of regularizing the loss function is generalizable to domains other than student response modeling.

- I proposed a way of combining multimodal fusion search and cell architecture search within one framework. To my best knowledge, this is the first architecture that combines these two. This framework is also domain independent. Besides, the sub-graph sampling process might be an effective way of preventing overfitting [34].

- I proposed using a new metric, weighted AUC (wAUC) and a simple weight assigning strategy. wAUC could be used to measure how one model performs with time. I hope this work will inspire more research in this direction.

- I discussed two ways of applying attention mechanisms on recurrent models. The results show attention mechanisms will not be helpful in all cases, especially when the data is very heterogeneous.

7.3. Future Research

Compared with other domains, the problem of noisy data is more serious in the domain of student response modeling—and it is difficult to detect outliers. Outliers could manifest in different ways considering student responses. For instance, one student might switch applications on the computer, resulting in a long *time spent* feature. To build robust models and do fair comparisons, we must make sure our data is solid and clean. I propose to tackle this problem in two ways: The first one is to develop a more robust outliers detector for students data. Unsupervised clustering algorithms might be used for this purpose. The second approach is to use synthetic data. There are at least two benefits of using synthetic data. The first benefit is scalability. Deep Learning based models require a lot of training data, using synthetic data could save huge amount of data collection time. The second benefit is synthetic data provide a way to do fair comparisons between different models, since we also have access to the backend generative models. Generative adversarial networks might be useful in this case. Even so, the value of actual data, even with noise sources, cannot be overstated.

BIBLIOGRAPHY

[1] Cs231n convolutional neural networks for visual recognition. https://cs231n.github.io/understanding-cnn/. (Accessed on 07/29/2020).

[2] Datashop home. https://pslcdatashop.web.cmu.edu/. (Accessed on 01/10/2020).

[3] Google ai blog: Inceptionism: Going deeper into neural networks. https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html. (Accessed on 07/29/2020).

[4] Karl rosaen — scikit-learn pipeline gotchas, k-fold cross-validation, hyperparameter tuning and improving my score on kaggle's forest cover type competition — ml learning log. http://karlrosaen.com/ml/learning-log/2016-06-20/. (Accessed on 07/27/2020).

[5] lecture_slides_lec6.pdf. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. (Accessed on 07/31/2020).

[6] Understanding lstm networks – colah's blog. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. (Accessed on 12/31/2019).

[7] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[8] Ai, F., Chen, Y., Guo, Y., Zhao, Y., Wang, Z., and Fu, G. Concept-aware deep knowledge tracing and exercise recommendation in an online learning system.

[9] Back, T. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[10] Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[11] Baltrušaitis, T., Ahuja, C., and Morency, L.-P. Multimodal machine learning: A survey and taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence 41*, 2 (2018), 423–443.

[12] Bouckaert, R. R., and Frank, E. Evaluating the replicability of significance tests for comparing learning algorithms. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2004), Springer, pp. 3–12.

[13] Brendel, W., and Bethge, M. Approximating cnns with bag-of-local-features models works surprisingly well on imagenet. *arXiv preprint arXiv:1904.00760* (2019).

[14] Brizan, D. G., Goodkind, A., Koch, P., Balagani, K., Phoha, V. V., and Rosenberg, A. Utilizing linguistically enhanced keystroke dynamics to predict typist cognition and demographics. *International Journal of Human-Computer Studies 82* (2015), 57–68.

[15] Carbonell, J. R. Ai in cai: An artificial-intelligence approach to computer-assisted instruction. *IEEE transactions on man-machine systems 11*, 4 (1970), 190–202.

[16] Carbonell, J. R. Mixed-initiative man-computer instructional dialogues. Tech. rep., BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS, 1970.

[17] Cen, H., Koedinger, K., and Junker, B. Learning factors analysis–a general method for cognitive model evaluation and improvement. In *International Conference on Intelligent Tutoring Systems* (2006), Springer, pp. 164–175.

[18] Chakraborty, S., Tomsett, R., Raghavendra, R., Harborne, D., Alzantot, M., Cerutti, F., Srivastava, M., Preece, A., Julier, S., Rao, R. M., et al. Interpretability of deep learning models: a survey of results. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)* (2017), IEEE, pp. 1–6.

[19] Chandrashekar, G., and Sahin, F. A survey on feature selection methods. *Computers & Electrical Engineering 40*, 1 (2014), 16–28.

[20] Chen, C.-H., Hwang, G.-J., and Tsai, C.-H. A progressive prompting approach to conducting context-aware learning activities for natural science courses. *Interacting with Computers 26*, 4 (2014), 348–359.

[21] Cheng, J., Dong, L., and Lapata, M. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733* (2016).

[22] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[23] Clancey, W. J. Transfer of rule-based expertise through a tutorial dialogue. Tech. rep., STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1979.

[24] Conklin, J. A taxonomy for learning, teaching, and assessing: A revision of bloom's taxonomy of educational objectives complete edition, 2005.

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

[26] Cortes, C., and Vapnik, V. Support-vector networks. *Machine learning 20*, 3 (1995), 273–297.

[27] Csáji, B. C. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary 24* (2001), 48.

[28] Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological) 39*, 1 (1977), 1–22.

[29] Demšar, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research 7*, Jan (2006), 1–30.

[30] Der Kiureghian, A., and Ditlevsen, O. Aleatory or epistemic? does it matter? *Structural safety 31*, 2 (2009), 105–112.

[31] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[32] Dietterich, T. G. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation 10*, 7 (1998), 1895–1923.

[33] Ding, X., and Larson, E. Why deep knowledge tracing has less depth than anticipated. In *Educational Data Mining* (2019), pp. 282–287.

[34] Ding, X., and Larson, E. C. Automatic rnn cell design for knowledge tracing using reinforcement learning. In *Proceedings of the Seventh ACM Conference on Learning@ Scale* (2020), pp. 285–288.

[35] Ding, X., and Larson, E. C. Incorporating uncertainties in student response modeling by loss function regularization. *Neurocomputing* (2020).

[36] Ding, X., Larson, E. C., Doyle, A., Donahoo, K., Rajgopal, R., and Bing, E. Eduaware: using tablet-based navigation gestures to predict learning module performance. *Interactive Learning Environments* (2019), 1–13.

[37] Ding, X., Nassehi, D., and Larson, E. C. Measuring oxygen saturation with smartphone cameras using convolutional neural networks. *IEEE journal of biomedical and health informatics* (2018).

[38] Ding, X., Raziei, Z., Larson, E. C., Olinick, E. V., Krueger, P., and Hahsler, M. Swapped face detection using deep learning and subjective assessment. *arXiv preprint arXiv:1909.04217* (2019).

[39] Dong, X., and Yang, Y. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 3681–3690.

[40] Drasgow, F., and Hulin, C. L. Item response theory.

[41] Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *Journal of Machine Learning Research 20*, 55 (2019), 1–21.

[42] Feng, M., Heffernan, N. T., and Koedinger, K. R. Addressing the testing challenge with a web-based e-assessment system that tutors as it assesses. In *Proceedings of the 15th international conference on World Wide Web* (2006), ACM, pp. 307–316.

[43] Gal, Y. *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge, 2016.

[44] Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2014), pp. 580–587.

[45] Glorot, X., and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010), pp. 249–256.

[46] González-Brenes, J., Huang, Y., and Brusilovsky, P. General features in knowledge tracing to model multiple subskills, temporal item response theory, and expert knowledge. In *The 7th International Conference on Educational Data Mining* (2014), University of Pittsburgh, pp. 84–91.

[47] Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press, 2016.

[48] Graves, A. Practical variational inference for neural networks. In *Advances in neural information processing systems* (2011), pp. 2348–2356.

[49] Graves, A., and Schmidhuber, J. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks 18*, 5-6 (2005), 602–610.

[50] Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv preprint arXiv:1410.5401* (2014).

[51] Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. On calibration of modern neural networks. *arXiv preprint arXiv:1706.04599* (2017).

[52] Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. Gene selection for cancer classification using support vector machines. *Machine learning 46*, 1-3 (2002), 389–422.

[53] Hachey, A. C., Wladis, C. W., and Conway, K. M. Do prior online course outcomes provide more information than gpa alone in predicting subsequent online course grades and retention? an observational study at an urban community college. *Computers & Education 72* (2014), 59–67.

[54] He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (2015), pp. 1026–1034.

[55] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[56] Hestenes, M. R., Stiefel, E., et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards 49*, 6 (1952), 409–436.

[57] Hinton, G. E., and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *science 313*, 5786 (2006), 504–507.

[58] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[59] Hori, C., Hori, T., Lee, T.-Y., Zhang, Z., Harsham, B., Hershey, J. R., Marks, T. K., and Sumi, K. Attention-based multimodal fusion for video description. In *Proceedings of the IEEE international conference on computer vision* (2017), pp. 4193–4202.

[60] Huang, S., and Fang, N. Predicting student academic performance in an engineering dynamics course: A comparison of four types of predictive mathematical models. *Computers & Education 61* (2013), 133–145.

[61] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[62] Jaynes, E. T. *Probability theory: The logic of science*. Cambridge university press, 2003.

[63] Johns, J., Mahadevan, S., and Woolf, B. Estimating student proficiency using an item response theory model. In *International Conference on Intelligent Tutoring Systems* (2006), Springer, pp. 473–480.

[64] Johnson, M. S., et al. Marginal maximum likelihood estimation of item response models in r. *Journal of Statistical Software 20*, 10 (2007), 1–24.

[65] Joshi, P. D., Bewoor, M., and Patil, S. System for document summarization using graphs in text mining.

[66] Käser, T., Klingler, S., Schwing, A. G., and Gross, M. Beyond knowledge tracing: Modeling skill topologies with bayesian networks. In *International Conference on Intelligent Tutoring Systems* (2014), Springer, pp. 188–198.

[67] Kendall, A., and Gal, Y. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems* (2017), pp. 5574–5584.

[68] Khajah, M., Lindsey, R. V., and Mozer, M. C. How deep is knowledge tracing? *arXiv preprint arXiv:1604.02416* (2016).

[69] Khajah, M., Wing, R., Lindsey, R., and Mozer, M. Integrating latent-factor and knowledge-tracing models to predict individual differences in learning. In *Educational Data Mining 2014* (2014), Citeseer.

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

[71] Kim, B. *Interactive and interpretable machine learning models for human machine collaboration*. PhD thesis, Massachusetts Institute of Technology, 2015.

[72] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[73] Koedinger, K. R., Anderson, J. R., Hadley, W. H., and Mark, M. A. Intelligent tutoring goes to school in the big city.

[74] Koedinger, K. R., Baker, R. S., Cunningham, K., Skogsholm, A., Leber, B., and Stamper, J. A data repository for the edm community: The pslc datashop. *Handbook of educational data mining 43* (2010), 43–56.

[75] Krening, S., Harrison, B., Feigh, K. M., Isbell, C. L., Riedl, M., and Thomaz, A. Learning from explanations using sentiment and advice in rl. *IEEE Transactions on Cognitive and Developmental Systems 9*, 1 (2016), 44–55.

[76] Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images.

[77] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[78] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature 521*, 7553 (2015), 436–444.

[79] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[80] LeCun, Y., et al. Generalization and network design strategies. In *Connectionism in perspective*, vol. 19. Citeseer, 1989.

[81] Leibowitz, H. W., Shupert, C., and Post, R. B. The two modes of visual processing: Implications for spatial orientation.

[82] Li, J., and Fine, J. P. Weighted area under the receiver operating characteristic curve and its application to gene selection. *Journal of the Royal Statistical Society: Series C (Applied Statistics) 59*, 4 (2010), 673–692.

[83] Lipton, Z. C. The mythos of model interpretability. *Queue 16*, 3 (2018), 31–57.

[84] Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 19–34.

[85] Long, X., Gan, C., De Melo, G., Liu, X., Li, Y., Li, F., and Wen, S. Multimodal keyless attention fusion for video classification. In *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

[86] Luong, M.-T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).

[87] Maaten, L. v. d., and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research 9*, Nov (2008), 2579–2605.

[88] MacKay, D. J. A practical bayesian framework for backpropagation networks. *Neural computation 4*, 3 (1992), 448–472.

[89] Margaryan, A., Bianco, M., and Littlejohn, A. Instructional quality of massive open online courses (moocs). *Computers & Education 80* (2015), 77–83.

[90] McCulloch, W. S., and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics 5*, 4 (1943), 115–133.

[91] McNemar, Q. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika 12*, 2 (June 1947), 153–157.

[92] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[93] Morgado, P., and Vasconcelos, N. Nettailor: Tuning the architecture, not just the weights. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 3044–3054.

[94] Neal, R. Bayesian learning for neural networks [phd thesis]. *Toronto, Ontario, Canada: Department of Computer Science, University of Toronto* (1995).

[95] Nghe, N. T., Janecek, P., and Haddawy, P. A comparative analysis of techniques for predicting academic performance. In *Frontiers in Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 2007. FIE'07. 37th Annual* (2007), IEEE, pp. T2G–7.

[96] Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., and Ng, A. Y. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (2011), pp. 689–696.

[97] Nguyen, A., Yosinski, J., and Clune, J. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 427–436.

[98] Pandey, S., and Karypis, G. A self-attentive model for knowledge tracing. In *Educational Data Mining* (2019), pp. 384–389.

[99] Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International conference on machine learning* (2013), pp. 1310–1318.

[100] Pasunuru, R., and Bansal, M. Continual and multi-task architecture search. *arXiv preprint arXiv:1906.05226* (2019).

[101] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch.

[102] Pavlik Jr, P. I., Cen, H., and Koedinger, K. R. Performance factors analysis–a new alternative to knowledge tracing. *Online Submission* (2009).

[103] Peng, H., Chuang, P.-Y., Hwang, G.-J., Chu, H.-C., Wu, T.-T., Huang, S.-X., et al. Ubiquitous performance-support system as mindtool: A case study of instructional decision making and learning assistant. *Educational Technology & Society 12*, 1 (2009), 107–120.

[104] Pérez-Rúa, J.-M., Vielzeuf, V., Pateux, S., Baccouche, M., and Jurie, F. Mfas: Multimodal fusion architecture search. In *Proceedings of the IEEE Conference on computer vision and pattern recognition* (2019), pp. 6966–6975.

[105] Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268* (2018).

[106] Phan, T., McNeil, S. G., and Robin, B. R. Students' patterns of engagement and course performance in a massive open online course. *Computers & Education 95* (2016), 36–44.

[107] Piech, C., Bassen, J., Huang, J., Ganguli, S., Sahami, M., Guibas, L. J., and Sohl-Dickstein, J. Deep knowledge tracing. In *Advances in neural information processing systems* (2015), pp. 505–513.

[108] Piergiovanni, A., Angelova, A., Toshev, A., and Ryoo, M. S. Evolving space-time neural architectures for videos. In *Proceedings of the IEEE international conference on computer vision* (2019), pp. 1793–1802.

[109] Radu, V., Tong, C., Bhattacharya, S., Lane, N. D., Mascolo, C., Marina, M. K., and Kawsar, F. Multimodal deep learning for activity and context recognition. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 1*, 4 (2018), 157.

[110] Rae, J., Hunt, J. J., Danihelka, I., Harley, T., Senior, A. W., Wayne, G., Graves, A., and Lillicrap, T. Scaling memory-augmented neural networks with sparse reads and writes. In *Advances in Neural Information Processing Systems* (2016), pp. 3621–3629.

[111] Ramachandram, D., and Taylor, G. W. Deep multimodal learning: A survey on recent advances and trends. *IEEE Signal Processing Magazine 34*, 6 (2017), 96–108.

[112] Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[113] Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. Learning representations by back-propagating errors. *Cognitive modeling 5*, 3 (1988), 1.

[114] Russell, S., and Norvig, P. Artificial intelligence: a modern approach.

[115] Saikia, T., Marrakchi, Y., Zela, A., Hutter, F., and Brox, T. Autodispnet: Improving disparity estimation with automl. In *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 1812–1823.

[116] Self, J. A. Student models in computer-aided instruction. *International Journal of Man-machine studies 6*, 2 (1974), 261–276.

[117] Self, J. A. Concept teaching. *Artificial intelligence 9*, 2 (1977), 197–221.

[118] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 618–626.

[119] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature 529*, 7587 (2016), 484–489.

[120] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research 15*, 1 (2014), 1929–1958.

[121] Srivastava, N., and Salakhutdinov, R. R. Multimodal learning with deep boltzmann machines. In *Advances in neural information processing systems* (2012), pp. 2222–2230.

[122] Stamper, J., Niculescu-Mizil, A., Ritter, S., Gordon, G., and Koedinger, K. R. Algebra i 2005-2006. challenge data set from kdd cup 2010 educational data mining challenge. find it at http://pslcdatashop.web.cmu.edu/kddcup/downloads.jsp, 2010.

[123] Sukhbaatar, S., Weston, J., Fergus, R., et al. End-to-end memory networks. In *Advances in neural information processing systems* (2015), pp. 2440–2448.

[124] Sutton, R. S., and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

[125] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems* (2000), pp. 1057–1063.

[126] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[127] Torres Irribarra, D., and Freund, R. Wright map: Irt item-person map with conquest integration. *R package. Retrieved January 10* (2014), 2015.

[128] van der Linden, W. J., and Hambleton, R. K. *Handbook of modern item response theory*. Springer Science & Business Media, 2013.

[129] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems* (2017), pp. 5998–6008.

[130] Vizer, L. M., Zhou, L., and Sears, A. Automated stress detection using keystroke and linguistic features: An exploratory study. *International Journal of Human-Computer Studies 67*, 10 (2009), 870–886.

[131] Wang, T., Wang, K., Cai, H., Lin, J., Liu, Z., Wang, H., Lin, Y., and Han, S. Apq: Joint search for network architecture, pruning and quantization policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 2078–2087.

[132] Wangwiwattana, C., Ding, X., and Larson, E. C. Pupilnet, measuring task evoked pupillary response using commodity rgb tablet cameras: Comparison to mobile, infrared gaze trackers for inferring cognitive load. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 1*, 4 (2018), 171.

[133] Weng, L. Attention? attention! *lilianweng.github.io/lil-log* (2018).

[134] Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE 78*, 10 (1990), 1550–1560.

[135] Weston, J., Chopra, S., and Bordes, A. Memory networks. *arXiv preprint arXiv:1410.3916* (2014).

[136] Wieting, J., and Kiela, D. No training required: Exploring random encoders for sentence classification. *arXiv preprint arXiv:1901.10444* (2019).

[137] Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning 8*, 3-4 (1992), 229–256.

[138] Wilson, M., and Draney, K. A technique for setting standards and maintaining them over time. *Measurement and multivariate analysis* (2002), 12–14.

[139] Woolf, B. P. *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning*. Morgan Kaufmann, 2010.

[140] Xiong, X., Zhao, S., Van Inwegen, E. G., and Beck, J. E. Going deeper with deep knowledge tracing. *International Educational Data Mining Society* (2016).

[141] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning* (2015), pp. 2048–2057.

[142] Yang, D., Sinha, T., Adamson, D., and Rosé, C. P. Turn on, tune in, drop out: Anticipating student dropouts in massive open online courses. In *Proceedings of the 2013 NIPS Data-driven education workshop* (2013), vol. 11, p. 14.

[143] Yang, H., and Cheung, L. P. Implicit heterogeneous features embedding in deep knowledge tracing. *Cognitive Computation 10*, 1 (2018), 3–14.

[144] Yeung, C.-K. Deep-irt: Make deep learning based knowledge tracing explainable using item response theory. *arXiv preprint arXiv:1904.11738* (2019).

[145] Yeung, C.-K., and Yeung, D.-Y. Addressing two problems in deep knowledge tracing via prediction-consistent regularization. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale* (2018), ACM, p. 5.

[146] Yudelson, M. V., Koedinger, K. R., and Gordon, G. J. Individualized bayesian knowledge tracing models. In *International conference on artificial intelligence in education* (2013), Springer, pp. 171–180.

[147] Zaremba, W., and Sutskever, I. Reinforcement learning neural turing machines-revised. *arXiv preprint arXiv:1505.00521* (2015).

[148] Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).

[149] Zhang, J., Shi, X., King, I., and Yeung, D.-Y. Dynamic key-value memory networks for knowledge tracing. In *Proceedings of the 26th international conference on World Wide Web* (2017), International World Wide Web Conferences Steering Committee, pp. 765–774.

[150] Zhang, L., Xiong, X., Zhao, S., Botelho, A., and Heffernan, N. T. Incorporating rich features into deep knowledge tracing. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale* (2017), ACM, pp. 169–172.

[151] Zilly, J. G., Srivastava, R. K., Koutnik, J., and Schmidhuber, J. Recurrent highway networks. In *International Conference on Machine Learning* (2017), pp. 4189–4198.

[152] Zoph, B., and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).