Master's Thesis

Applying Probabilistic Graphical Models and Deep Reinforcement Learning in a Learning-Aware Application

Anna Saranti

Institute of Interactive Systems and Data Science,
Graz, University of Technology

Supervisor: Assoc. Prof. Dr. Andreas Holzinger

Graz, April 1, 2019

This page intentionally left blank

Masterarbeit

(Diese Arbeit ist in englischer Sprache verfasst)

Entwicklung von Probabilistischen Graphischen Modellen und tiefen Neuronalen Netzen für Bestärkendes Lernen in einer Lernadaptiven Applikation

Anna Saranti

Institute of Interactive Systems and Data Science,
Technische Universität Graz

Betreuer: Univ.-Doz. Mag.phil. Mag.rer.nat. Dr.phil. Ing. Andreas Holzinger

Graz, 1. April 2019

This page intentionally left blank

Abstract

Learning Analytics applications can be enhanced by the use of Machine Learning methods, particularly by Probabilistic Graphical Models and Deep Reinforcement Learning. This thesis addresses the research question of finding a possibility for a qualitative change from informative tools, providing analytical reports to human supervisors, to a learning-aware system with active intervention to the learning process. The possibility as well as the degree of improvement of the learning process through the combination of different artificial intelligence techniques are explored, both basing on real data and programmatic simulations.

This thesis starts with the overview of existing student modelling tools using Bayesian Networks. It continues with the description of the learning application that not only provided the research data but also influenced the design decisions for the Probabilistic Graphical Model suited to the problem purpose. The mathematical equations that were derived for the efficient computation of the model's parameters as well as the insights about the students learning competence that emerged with the application of inference on the model, complete the first thematic part of the thesis.

In the second part, information contained in the Probabilistic Graphical Model was used for a simulated decision-making process that improves the learning efficiency. A Deep Neural Network trained asynchronously learns the descriptive features of the model and proposes the next learning application's action; thereby a personalized suitable sequence is discovered. Explainable artificial intelligence methods are used to uncover and evaluate the criteria on which the learning strategy is decided. The thesis concludes with the basic description of the programmed software components and the planned future research.

Keywords

Probabilistic Graphical Models, Bayesian Networks, Deep Reinforcement Learning, Learning Analytics, Learning-Aware Application

ÖSTAT classification

102019 Machine Learning

ACM classification

Machine Learning

Kurzfassung

Lernadaptive Applikationen können durch die Verwendung von Methoden des Maschinellen Lernens erweitert werden, insbesondere durch Probabilistische Graphische Modelle und tiefe Neuronale Netze für Bestärkendes Lernen. Die Forschungsfrage dieser Masterarbeit befasst sich mit der Suche nach einem Ansatz einer qualitativen Veränderung von informativen Mitteln, die Analyseberichten an menschliche Betreuer bereitstellen, zu einem lernfähigen System mit aktivem Eingriff in den Lernprozess. Es werden sowohl die Möglichkeit als auch der Grad der Verbesserung des Lernprozesses durch Kombination verschiedener Techniken der künstlichen Intelligenz untersucht. Dies erfolgt auf Basis realer Daten als auch anhand programmatischer Simulationen.

Diese Masterarbeit beginnt mit der Übersicht über vorhandene Studenten-Modellierungstools, welche Bayes'sche Netze verwenden. Darauffolgend wird die Lernanwendung beschrieben, welche nicht nur die Forschungsdaten lieferte, sondern auch die Entwurfsentscheidungen für das auf diesen Problemzweck abgestimmte Probabilistische Graphische Modell beeinflusste. Die mathematischen Gleichungen, die zur effizienten Berechnung der Parameter des Modells abgeleitet wurden, sowie die Erkenntnisse über die Lernkompetenz der Schüler, die sich aus der Anwendung von Inferenz auf das Modell ergaben, vervollständigen den ersten thematischen Teil der Arbeit.

In einem zweiten Teil wurden die im Probabilistischen Graphischen Modell enthaltenen Informationen für einen simulierten Entscheidungsprozess verwendet, welcher die Lerneffizienz verbessert. Ein asynchron trainiertes tiefes Neuronales Netz lernt die deskriptiven Merkmale des Modells und schlägt die nächste Aktion der Lernanwendung vor. Auf diese Weise wird eine passende personalisierte Sequenz entdeckt. Es werden Methoden, welche künstliche Intelligenz erklärbar machen, verwendet, um Kriterien nach denen die Lernstrategie festgelegt wird aufzudecken und zu evaluieren. Die Arbeit schließt mit einer kompakten Beschreibung der programmierten Softwarekomponenten sowie der geplanten zukünftigen Forschung ab.

Schlüsselwörter

Probabilistische Graphische Modelle, Bayes'sche Netze, Bestärkendes Lernen mit

tiefen Neuronalen Netzen, Analyse des Lernens, lernadaptives Applikation

ÖSTAT Klassifikation

102019 Machine Learning

ACM Klassifikation

Machine Learning

This page intentionally left blank

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, April 1, 2019

Anna Saranti

This page intentionally left blank

Acknowledgements

Usually, this section is all about thanking people that helped the making of the diploma thesis and the completion of the studies, but I would first and foremost like to thank every person that posed difficulties in my life. They were very time- and energy-consuming, but set off the charge for this studies and over the years helped to maintain my determination and persistence.

The supervisor of this diploma thesis, Univ.-Doz. Mag.phil. Mag.rer.nat. Dr.phil. Ing. Andreas Holzinger, has helped me enormously with his competence, excellent technical support, and patience. It is he, as well as the Institute of Interactive Systems and Data Science as a whole, that I can't thank enough for everything they've taught me and the new job opportunities that emerged as a result of their teaching activities.

A special mention needs also to be made for Priv.-Doz. Dipl.-Ing. Dr.techn. Martin Ebner, head of the Educational Technology, whom I was privileged to work for. His ideas, unique vision, decisiveness and support to this challenging work was always there.

I dedicate this diploma thesis euqally to Dipl.-Ing. Behnam Taraghi BSc. and Mag.rer.nat. Walther Nagler.

Anna Saranti

Graz, April 1, 2019

This page intentionally left blank

Table of Contents

1. Introduction

1.1 Previous Work

In general, Learning Analytics is said to be a key factor for the future of learning (Siemens and d Baker (2012); Ebner et al. (2013a, 2015)). On the base of data analyses, Learning Analytics strives to assist the learning process by giving educators a deeper insight into learning processes and results. It must be stated that teachers play an essential role in Learning Analytics. Including results of data analysis indicating incidents, they are responsible for intervening in a pedagogical manner. A related field of Learning Analytics is educational data mining. According to Romero and Ventura (2010), educational data mining is a field that makes use of statistical, machine-learning based, and data-mining algorithms applying to the different types of educational data (Romero and Ventura (2010)). While educational data mining targets on automatized learning activities, Learning Analytics helps and supports educators practising their daily work.

The so called "1x1 trainer" [1] is a Learning Analytics application developed by the department Educational Technology of Graz University of Technology. It uses the benefits of both fields, Learning Analytics as well as educational data mining (Ebner and Schön (2013); Ebner et al. (2013b); Schön et al. (2012)). The application poses exercises to students from the multiplication table with one decimal digit operands. The algorithm of the "1x1 trainer" adapts the sequence of questions given subject to the students answers individually, in order to improve ones learning progress. Furthermore, the algorithm itself needs to react adaptively to the changes of the learning progress of a student. In that way, supports each pupil according to the distinct learning progress over the whole period of learning with the application. This underlying personalized adaptive learning algorithm points out weak mathematical knowledge of single students and alerts teachers to intervene.

This thesis bases on previous work that used data gathered by the "1x1 trainer". Firstly, different mathematical questions were roughly classified according to the learners' answers. Questions were considered to be more difficult than others when students needed more attempts to answer them (Taraghi et al. (2014a,c)). Some specific questions could be identified difficult for the major part of the users. The next step was to analyse the reasons of errors made. Therefore, wrongly answered

[1] `https://schule.learninglab.tugraz.at/einmaleins/`, Last accessed 17 March 2019

questions were divided into error types corresponding to the innate cognitive and conceptual learning shortcomings of the users (Taraghi et al. (2015)). The "relative difficulty" of those questions - 2×3 seems to be simpler than 7×8 - played no role for the error type classification.

1.2 Bayesian Student Models

There is a plethora of learning applications that use probabilistic graphical models (also called bayesian models/networks) to model student's knowledge; most of them belong to the category of intelligent tutoring systems (ITS) or adaptive educational systems (AES). The main goal of an ITS is to provide personalized recommendations according to the different learning styles whereas AES adapts the learning content as well as its sequence according to the student's profile (Schiaffino et al. (2008)). As explained in the literature review by Chrysafiadi and Virvou (2013), there are two classes of intelligent tutoring systems: Systems that make diagnosis with the student's knowledge, misconceptions, learning style or cognitive state, and systems that plan a personalized strategy using diagnosis for each learner individually. Student modelling is considered a subproblem of user modelling, which is of central importance to ITS since otherwise each student is treated the same (Nouh et al. (2006)).

Primarily, Bayesian networks are chosen because of their ability to model uncertainty (Millán et al. (2000)) and, at the same time, to support a decision making process. The user modelling goals of a bayesian network for knowledge modelling is mainly to have an adaptive estimation of the knowledge itself, since it may increase or decrease during the learning process (Brusilovsky and Millán (2007)). Since scalar models and fuzzy logic approaches (Danaparamita and Gaol (2014)) have lower precision, structural models are built with the assumption that the knowledge is composed mainly by independent parts. On the other hand, bug/perturbation models (Chrysafiadi and Virvou (2013)) represent errors and misconceptions of the student. In this case, the Bayesian network is used to find the error that most probably caused the observable behavior (also called evidence) (Millán et al. (2000)), which is called credit/blame assignment problem (Pardos et al. (2010)). Bayesian networks can model the assumption that a wrongly answered question having two

potential causes is most probably caused by the one that is more prevalent, according to the data provided so far. Sometimes, random slips or typos are included in the model and do not rely on assumptions as for example: A wrongly answered question does not necessarily mean that the student does not know a concept completely, or a correctly answered one wasn't a guess. The structure in both cases constitutes the qualitative model; its definition uses domain knowledge and (optionally) data. The parametrization is learned from the data during a training phase and constitutes the quantitative model.

The reason for the creation of the model is in some cases to assist the teachers of large classes that suffer from a high dropout rate (Xenos (2004)). A model recognizes the student's knowledge faster and more accurate (Millán et al. (2000)), which is primarily beneficial when the class has a large number of students. In other cases that are summed up in Brusilovsky and Peylo (2003), the goal is to provide a personalized optimal sequence of the learning material or even to sequence the curriculum according to the student's individual needs. And yet, further cases (Stacey et al. (2003); Stacey and Flynn (2003)) show that the learning application that bases on the model provides long-term learning effects as opposed to traditional methods. This was pointed out by a post-test that was made several weeks after the learning sessions.

The issue of defining the prior beliefs, which consist the starting parameterization, is often coupled with user clustering; demographics, longitudal data (Stacey et al. (2003)), pre-tests (Goguadze et al. (2011b); Conati et al. (2002)), defining the prior beliefs as well as the starting groupings (Brusilovsky and Millán (2007)) with respect to the learners. In other cases, the teacher sets the prior beliefs from his/her experience (Nouh et al. (2006)) or a uniform prior is used (Conati et al. (2002); Pardos et al. (2010)). Another common characteristic is the definition of hidden structural elements that represent unobservable entities, which must be estimated from the observed ones. The design of the structure must take correct assumptions into account, basing a solid theoretical background, or else the model will not behave in a sound way (Millán et al. (2000)).

In the work of Millán et al. (2001), the researchers draw a parallel between medical diagnosis systems and student's knowledge diagnosis (Millán et al. (2010)). The student answers a set of questions that can only be answered correctly, when

several concepts are known. In this case the knowledge of the concepts is the cause of the answer. The noise in the process, for instance when a student knows the concept but answers wrongly and vice versa a correct guess, is also modelled. The initialization of the model parameters is made by teachers; afterwards the parameters are learned from the data. The model is used to efficiently determine those concepts the student knows less and the deductive proposal of the next question.

The "eTeacher" is a web-based education system (García et al. (2007); Schiaffino et al. (2008)) that recognizes the learning style of a student according to the performance in exams as well as the email, chat and messaging usage. The number of different learning ways and their characteristics is the domain knowledge defining the structure of the Bayesian network. The initialization of the parameters uses in some cases uniform priors and in others priors defined by experts. After that initial phase, the parameters are continuously learned from the behaviour of the students. After identifying the learning style, a recommendation engine proposes different ways to learn the same material to each student according to his or her learning style.

"ANDES" is an ITS developed by Conati et al. (1997), which mainly focuses on knowledge tracing but also on recognition of the learning plan of the user. The students solve Newtonian physics problems with different possible solution paths that define the Bayesian network's structure. Since each action may belong to different solution paths and the user does not provide its reasoning explicitly, the credit assignment problem is to find and quantify the most likely solution an action belongs to. This triggers personalized help instructions and hints in two cases: when a wrong answer is given or when the model predicts that the answer might be wrong. The parameters of the network change in an online manner while the student is solving the problems. Firstly, the evaluation was made by simulating students that have different knowledge profiles and measuring the accuracy of the predictions made by the model. In a second step, a post-test was carried out to compare real students having used "ANDES" to students who have not. Regression analysis was used to recognize the correlation between the use of the program and the learning gain (Bunt and Conati (2003)).

Specifically for mathematical problems there are several approaches that specialize in dealing with decimals misconceptions. In the work of Stacey et al. (2003);

Stacey and Flynn (2003) the misconceptions that define the structure of the model are provided by two main factors: the domain knowledge and data of a Decimal Conception Test (DCT) that students had to go through. Wrongly answered questions provided by the students depend on their misconceptions. The researchers defined the distinct misconception by computing which of them has the highest probability according to the data. Although the model drives different question sequencing strategies, some of the misconceptions were not correctly recognized. Therefore, the researchers decided that the teacher and not the system should provide instructions.

Also, the research work of Goguadze concentrates on the modelling of decimals misconceptions (Goguadze et al. (2011b,a)). The "AdaptErrEx" project selected the most frequently occurring misconceptions and ordered them a taxonomy (higher and lower level misconceptions), which is reflected in the dependencies of the Bayesian network. As the previous application, a wrong answer may be caused by different misconceptions. The prior beliefs are defined by a pre-test; the researchers assert that sufficient training data diminish the role of the prior in the computation of the posterior. This prior defines the typical/average student and then each user's parameters can be updated and individualized accordingly. One aspect that has not been considered in this model yet, is the difficulty of each question: easy questions will more likely be answered correctly than difficult ones, even if there is a high probability of misconception.

Several student modelling models track the progress of knowledge through time with Dynamic Bayesian Networks (DBN). The knowledge of the learner at each time point can be considered to be dependent on the knowledge and (optionally) the observed result of the interaction at the previous time point (Millán and Pérez-De-La-Cruz (2002)). The project "LISTEN" (Chang et al. (2006)) represents the hidden knowledge state of the student at each time point. The observable entities are the tutor interventions and the student's performance which are used to infer the knowledge state. In the work of Käser et al. (2017) there is an overview and comparison of Bayesian Knowledge Tracing (BKT), which is a technique for student modelling using a Hidden Markov model (HMM) modelling and DBN for various learning topics, such as number representation, mathematical operations, physics, algebra and spelling. A HMM is a special case of a DBN, which, according to the researchers, cannot represent dependencies that would lead to hierarchies of skills;

in these case DBNs create more adequate models.

All above described applications have a Bayesian network of the students model at its architecture core. There are a number of other components that either support the teacher or the student. One of them, for example, is the visualization of the model in the "VisMod" application (Zapata-Rivera and Greer (2004)), which is displayed in (among other things) color and arrow intensity instead of number-filled tables. This increases the readability of the model and enhances the tutor's understanding. Gamification elements can also be found in "Maths Garden" (Klinkenberg et al. (2011)), an application that lets users gain and loose points and coins depending on answering correctly or wrongly. A coaching component that provides feedback and hints to refresh the memory can be found in "ANDES" 's architecture (Bunt and Conati (2003)). An overview about the design and architectural elements of intelligent tutoring systems that have a Bayesian network as user model is provided in the work of Gamboa and Fred (2002).

A detailed overview about intelligent techniques other than Bayesian networks, such as recommender systems for the computation of the learning path as well as clustering and classification for learner profiles that are used in e-learning systems, is provided in Markowska-Kaczmar et al. (2010). Specifically in Brusilovsky and Millán (2007), the demand for the most appropriate activity proposed - neither too easy nor too difficult - can only be fulfilled, if the used model is both accurate and adaptive.

1.3 Research Question

The main objective of this thesis is to answer the research question, whether Bayesian networks can quantify the probability of defining misconceptions of one-digit multiplication problems or not. In order to answer this question a learning application is developed that considers both, the difficulty of a question as well as the relative importance of each misconception. Thus, the application focuses on the recognition of the current learning status. Learning-aware applications maintain an adaptive learning model that represents the knowledge of the learner/user with regard to the learned topic. The application is expected to support individual learning needs and abilities as well as considering common characteristics in the learning process of

different persons. The progress of the learning model itself will be used to transform the learning application into an adaptive one; that may change the content and sequence of assessments constantly to improve the learning process and to maximize the learning efforts.

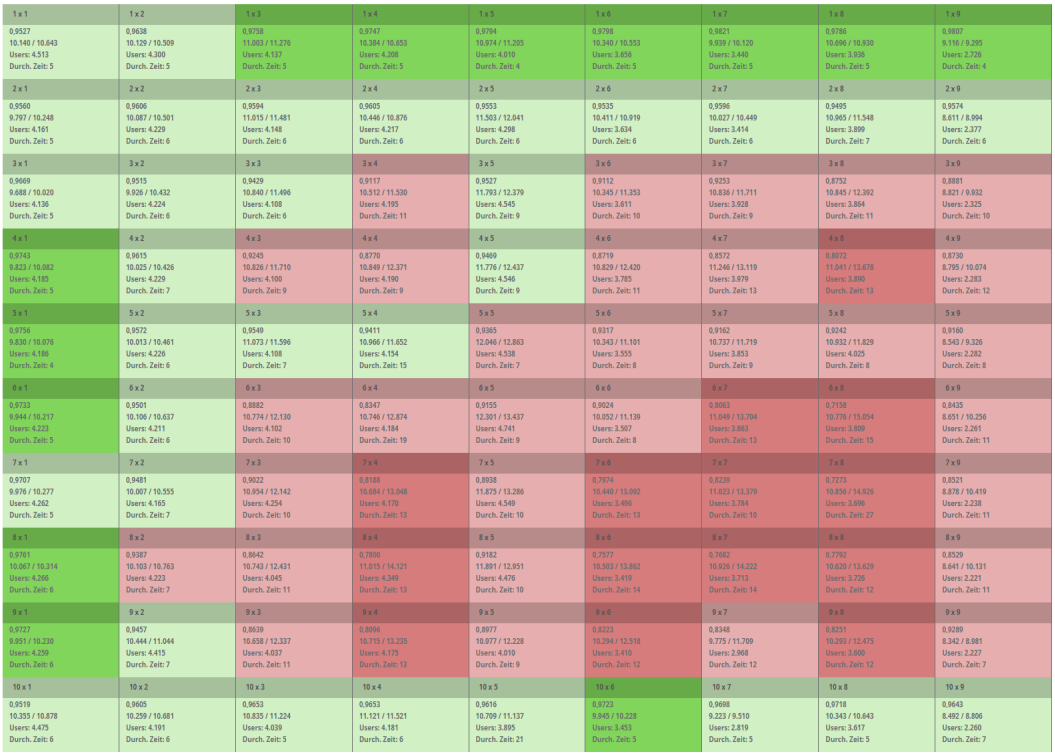


Figure 1.1: Current report of "1x1 trainer" provided to the teachers. The difficulty of the questions is color-encoded and several statistics as the percentage of users that have answered correct or false and the mean answering time is presented.

Figure 1.1 shows the current overall report of "1x1 trainer" that is accessible to teachers. It contains information about the actual number as well as the proportion of correct and wrong answers of each posed question. The color encoding helps distinguishing four sets of questions with similar proportion of correct and wrong answers. The implementation of the Bayesian model provides further insights to detailed cognitive information that enriches the information content of the current report. Furthermore, the new report can concentrate on individualized learning status and can be updated in real time after each action of the student.

1.4 Outline

This thesis proposes a Bayesian model for the learning competence of students using the "1x1 trainer" application. The first step is to specify the error types that are relevant for this research; their detailed description is made in chapter 2. Data analysis (specifically descriptive statistics) is used to guide the necessary assumptions about the modelled entities and their independences. Based on this information, the structure of the model and its parametrization is defined. The personalized model of each student and the method by which it adapts its parameters to new data is described in chapter 3. The usage of the model and the insights that are provided to the teachers in the form of an enhanced report are explained in chapter 4.

The Bayesian model can be used to predict the future behaviour of the student. Therefore, there exists a possibility to create a learning-aware application that takes that into account and adapts the posed learning content accordingly. The theoretical basis of neural networks, deep reinforcement learning, and specially the architectures applied to grid inputs are covered in chapter 5. With the use of a deep reinforcement learning method in a simulated environment, the sequence of posed questions is changed in a way that the learning goal is reached faster than with the current application's logic 6. The code organization, required libraries, installation guideline, as well as testing framework information is in chapter 7. Finally, a conclusion about future research and improvement possibilities is presented in the last chapter 8.

2. Error types of one-digit multiplication and descriptive statistics

2.1 Cognitive misconceptions

Research based on cognitive neuroscience can contribute domain knowledge about the influencing factors that cause errors in learning mathematics (Sousa (2008)). Technologies and methods like brain scanning were used to find out, which brain areas are active when performing specific mathematical problems. The first issue that needs to be defined in order to build a learning model for basic mathematical problems is, to indicate by what the causing factors are influenced. To figure out the potential dependencies and independences between these factors is the next thing to deal with before building their representative structure. One possible indication of independent causes can be given by their different processing areas in the brain.

It is suggested that humans are born with numerosity (concept of quantity) for numbers from 1 to 3; they developed number sense for their survival. Subitizing (quantifying a group of ≤ 4 objects) is also something that does not require abstract thinking. But as numbers get higher (more abstract), counting and putting numbers into sequence becomes necessary for accomplishing basic mathematical operations. Many errors occur from false estimations as well as comparison misunderstandings, which in turn mainly arise from the (compressed) mental number line representation. Arguably, one does not have intuition or a mental model about negative numbers, fractions, and irrationals. Language, in particular number words, symbols and their composition have an effect on number learning and processing abilities.

The most analysed basic mathematical operation (in regard to errors) is the multiplication. It mainly requires memorization of the multiplication table, which refers to long-term memory (rote learning), especially for big operands. In the following answers 45, 54, 56, 58 of the question 6×9, associative memory rhymes and patterns occur. Sometimes, interference with addition is apparent, as in the example of: $2 \times 3 = 5$. The different step-wise strategies that learners use to manage multiplication of big operands - by dividing them into simpler computations and combining the partial results - also disclose their problems with other basic mathematical operations.

One basic assumption of this thesis is that the error types of the wrongly answered mathematical questions are in direct connection to all possible misunder-

standings and shortcomings in these areas. To help learners and teachers to deal with these problems, a learning-aware application needs more than a statistical evaluation of the error types per question and per user as shown in 1.1. A probabilistic graphical model can capture different types of dependencies with a complex structure and is adaptive by its construction. An accurate quantification of the (mentally formed) reasons of the detected errors would provide a more informative policy invention for their correction.

2.2 Error types of one-digit multiplication problems

The bug library (Chrysafiadi and Virvou (2013)) of the proposed learning competence model contains six error types: operand, intrusion, consistency, off-by, add/-sub/div, and pattern. Any false answer that does not belong to one of those six categories is assigned to the unclassified category. The description of the error types is explained in detail in Taraghi et al. (2014b); a brief description follows here:

1. Operand error: It occurs, when the student mistakes at least one operand for one of its neighbours (Campbell (1995)). In the implementation only a neighbourhood of overall absolute distance of 2 from the correct operands was considered. One example is the answer 48 to the question 7×8 since the user may mistakenly multiplied 6×8. Research shows that this is the most frequently occurred error, but it occurs with a different proportion in each posed question (Campbell (1997)).

2. Operand intrusion (abbreviated intrusion) error: It happens, when the decades digit and/or the unit digit of the result equals one of the two operands of the posed question, for example $7 \times 8 = 78$. It is argued by Campbell (1995) that the two operands of the multiplication question are perceived as one number by the student (the first operand corresponding to the decades digit and the second to the unit digit).

3. Consistency: The student's answer has either the unit digit or the decade digit of the correct answer (Seidenberg and McClelland (1989); Domahs et al.

(2006)). For example, the answer 46 to the question 7×8 indicates that the unit digit is correct, but the decades digit is false.

4. Off-by-± 1, Off-by-± 2 : It occurs, when the answer of the student deviates from the correct one by ± 1 or ± 2, for example, when the answer of the question $5 * 8$ is one of the following: $\{38, 39, 41, 42\}$.

5. Add/Sub/Div: The student confuses the operation itself and performs for example an addition instead of a multiplication; in that case the answer to 7×8 is 15.

6. Pattern: The student mistakes the order of the digits of the result, for example, question 7×8 provides the answer 65 (the decades digit and the unit digit are permuted).

7. Unclassified: Any answer that can not be matched to one of the above error types.

All questions that have a correct answer with value smaller than 10 do not have consistency error. These are: $1 \times 1, 1 \times 2, 2 \times 1, 1 \times 3, 3 \times 1, 1 \times 4, 4 \times 1,$
$1 \times 5, 5 \times 1, 1 \times 6, 6 \times 1, 1 \times 7, 7 \times 1, 1 \times 8, 8 \times 1, 1 \times 9, 9 \times 1, 2 \times 2, 2 \times 3,$
$2 \times 4, 3 \times 2, 3 \times 3, 4 \times 2.$

One of the main reasons to use a probabilistic graphical model, is the fact that a specific false answer can be classified to multiple error types. The identification of the most probable error type causing a wrong answer is called credit assignment. The table 2.1 shows the possible false answers classification for the question 7×8. One can see that for example the answer 72 could occur because of an operand or an intrusion error.

Error Type	Answers
operand	40, 42, 48, 49, 54, 63, 64, 72
intrusion	18, 28, 38, 48, 58, 68, 71, 72, 73, 74, 75, 76, 77, 78, 79, 88, 98
consistency	16, 26, 36, 46, 51, 52, 53, 54, 55, 57, 58, 59, 66, 76, 86, 96
off-by-±1/off-by-±2	54, 55, 57, 58
add/sub/div	1, 15
pattern	65
unclassified	4, 5, 6, 7, 8, 9, 10, 11, 12, 19, 20, 21, 22, 23, 24, 25, 27, 29, 30, 31, 32, 33, 34, 35, 37, 39, 41, 43, 44, 45, 47, 50, 60, 61, 62, 67, 69, 80, 81, 82, 83, 84, 85, 87, 89, 90, 91, 92, 93, 94, 95, 97, 99
correct	56

Table 2.1: Answers for question 7×8 listed by error types

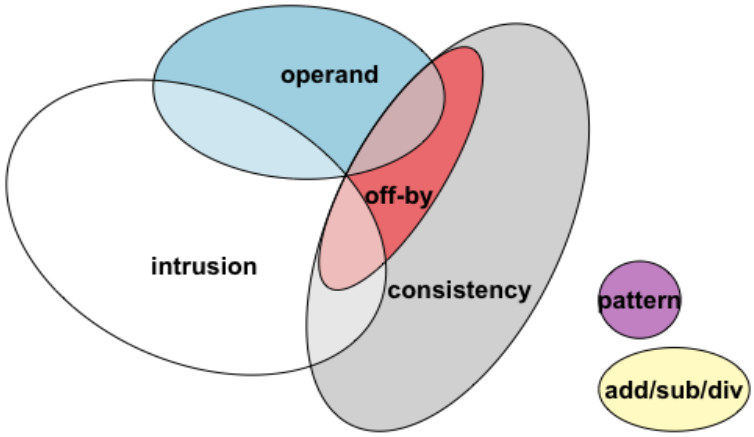


Figure 2.1: Euler diagram of the error type's answers of the question 7×8. The area of each ellipse is proportional to the number of elements in the corresponding error type's answers set.

The difference in the number of answers caused by each error type is depicted with an Euler Diagram in figure 2.1. All error types of the question 7×8 are presented, except the unclassified error, which per definition does not contain any common elements with other error types. The Euler diagram was created with the

library `eulerr` [1] of the language `R` [2]; the area of each ellipse is proportional to the number of elements in the corresponding set.

2.3 Data of the "1x1 trainer"

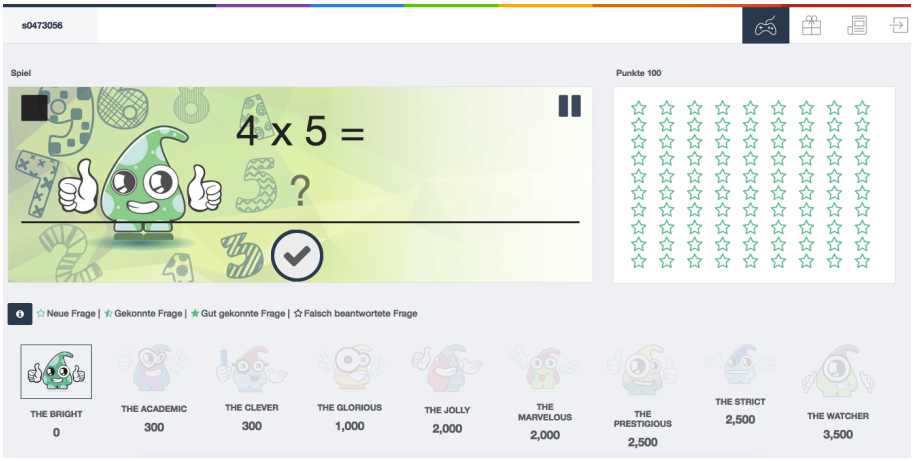
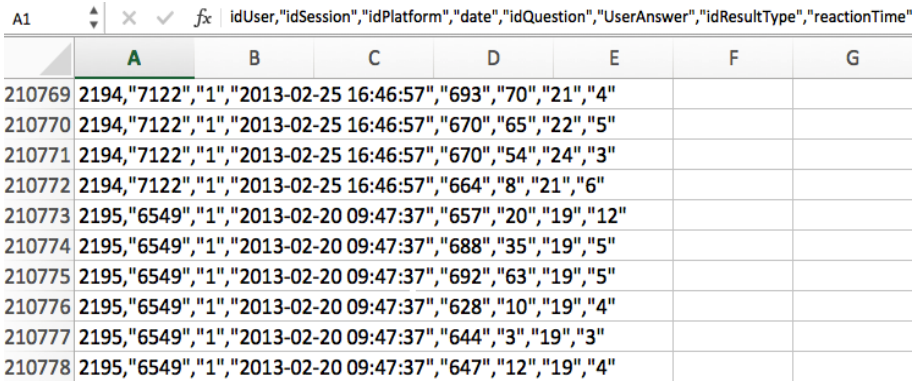


Figure 2.2: "1x1 trainer" application interface

The data that were used for building the model were provided from the "1x1 trainer" application. The application is for both students and teachers. For this work it was also used for a preliminary categorization of learners. Users of this application are confronted with multiplication questions with both multiplicands being one-digit integer numbers. The possible questions range from 1×1 up to 10×9 (a total of 90 questions) and are posed in a pre-specified order, which is briefly described in 2.5. The application does not provide any means of help or hints to the students so far; the only feedback users get, is whether their answer is correct or not. It is expected that by repeated use of the application the students will learn and get better through exercise. But there is no individualisation that takes care of the personal needs of the learning style and knowledge level of the users. Furthermore, personal information such as age, gender, demographics, and educational level were not collected.

[1] `https://cran.r-project.org/web/packages/eulerr/index.html`, Last accessed 17 March 2019

[2] `https://www.r-project.org/`, Last accessed 17 March 2019



A1	idUser,"idSession","idPlatform","date","idQuestion","UserAnswer","idResultType","reactionTime"						
	A	B	C	D	E	F	G
210769	2194,"7122","1","2013-02-25 16:46:57","693","70","21","4"						
210770	2194,"7122","1","2013-02-25 16:46:57","670","65","22","5"						
210771	2194,"7122","1","2013-02-25 16:46:57","670","54","24","3"						
210772	2194,"7122","1","2013-02-25 16:46:57","664","8","21","6"						
210773	2195,"6549","1","2013-02-20 09:47:37","657","20","19","12"						
210774	2195,"6549","1","2013-02-20 09:47:37","688","35","19","5"						
210775	2195,"6549","1","2013-02-20 09:47:37","692","63","19","5"						
210776	2195,"6549","1","2013-02-20 09:47:37","628","10","19","4"						
210777	2195,"6549","1","2013-02-20 09:47:37","644","3","19","3"						
210778	2195,"6549","1","2013-02-20 09:47:37","647","12","19","4"						

Figure 2.3: Example of the contents of the answers in a comma-separated file

The data were provided in a comma-separated file (.csv) as shown in 2.3 and needed preprocessing. The answers that did not lie in the interval $[0 - 100)$ were considered invalid and were removed. Overall there were 1179720 question-answer pairs with 1164786 valid. The number of unique users that gave at least one valid answer is 9058. The file covers eight columns providing the user ID, session ID, platform ID, date and time of the answer, as well as the reaction time of the student. Along with the posed question and the provided answer, the ID of the result type as one of {R, WR, W, WWR, WW, WWWR, WWW, WWWW}, whereas W means "wrong" and R "right" is stored. The detailed description of the result type is shown in Taraghi et al. (2014b) and is basically a way to quantify the relative difficulty of each question by keeping the recent history of the user's answering behaviour for each question. This information and the reaction time was not used in the model.

2.4 Data Analysis and Descriptive Statistics

To help designing the probabilistic graphical model, some analysis steps were necessary to be carried out with the data. The analysis and descriptive statistics (Freedman et al. (2007)) provides insights about the overall similarities and differences between the students (Godsey (2017)).

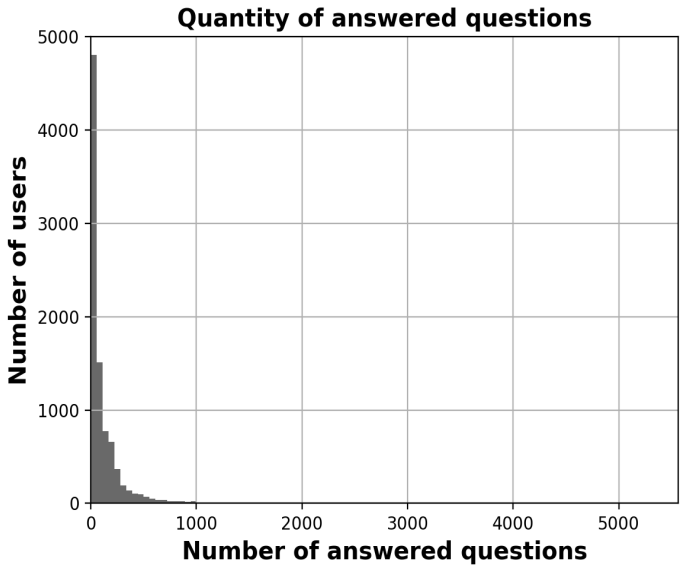


Figure 2.4: Histogram of user activity. $98,6\%$ of the users have ≤ 1000 valid answered questions.

Firstly, not every user has answered the same number of questions as shown in figure 2.4. The vast majority of the users $(98,6\%)$ have ≤ 1000 valid answered questions. One user has answered 13228 times and therefore was removed from the histogram to improve the readability of figure 2.4. For the training of the model, the prior must have an equal amount of answers for each question. This does not take the sequence of posed questions into consideration.

Secondly, for each question the proportion of wrong answers was computed and depicted with a heatmap, whereas the x-axis is the first operand and the y-axis the second (see figure 2.5). As it turned out, the most difficult question is 6×8 with 26.8% of wrong answers given. It must be advised to remember that not all questions are posed the same number of times, because of the algorithm that chooses the question sequence. Therefore the belief about the relative difficulty of the questions has not an equal confidence for all the questions.

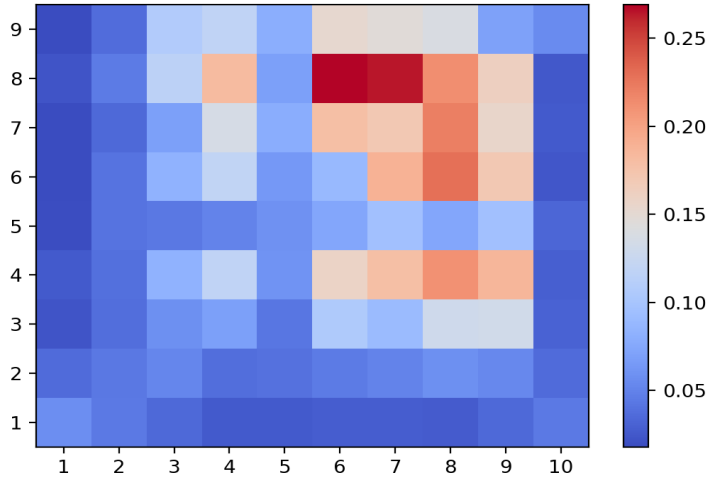


Figure 2.5: Relative difficulty of the questions measured by the proportion of wrong answers.

2.5 Sequence of posed questions

The algorithm for the sequence of the posed questions is described in detail in Schön et al. (2012), Kraja et al. (2017). The main goal is the individualization of the learning path for each student in a way that the posed question is "appropriate" to the current learning status; the question must be in a set of solvable but not too easy questions. To achieve this, the application has to measure the difficulty of the question in a non-personalized way and compare it to an estimation of the learning competence of each user. The learning competence was not modelled by a probabilistic model; it is described by a number that represents the degree of learning competence, which is updated in real-time.

There is a list of requirements specifying the next posed question. For example, a question that was posed at a specific point of time should not be repeated directly after with high probability. The estimation of the initial learning competence of each student is made with a pre-test containing two representative questions out of the whole set of questions. The rules for the definition and adaptation of the learning competence estimation are expressed by mathematical equations. The authors characterize some adaptation formulas that they used at the beginning of their

research “volatile” and discovered new ones that provided a more stable estimation of the learning competence.

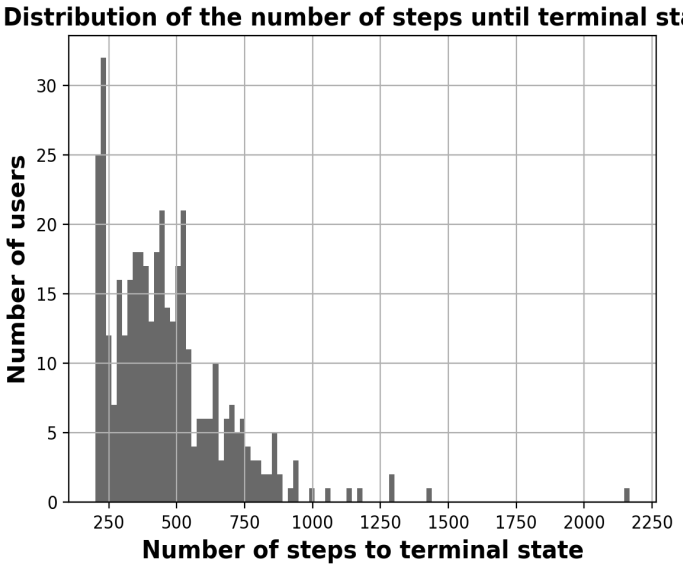


Figure 2.6: Number of users that reach the learning goal in a specific number of steps.

The learning goal of the application is reached when the student has answered all questions correctly the last two times they were posed. The number of steps that are needed to reach that state (terminal state) with the current algorithm is depicted in the histogram of figure 2.6. The number of users that eventually reached this state is 393.

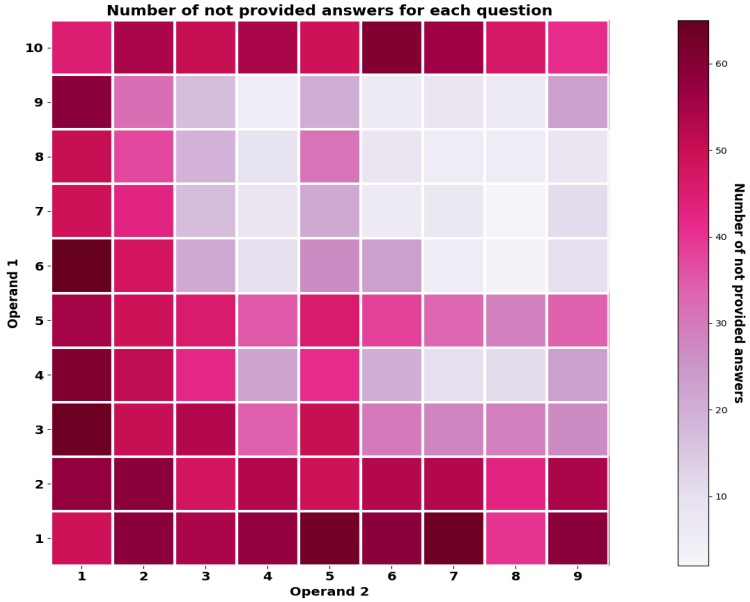


Figure 2.7: Number of not provided valid answers per question.

Even in a dataset of this size, there were answers that were never provided by the students. This influences the student model as it will be shown in chapter 3. Figure 2.7 indicates that the users provided almost all possible valid answers to questions that are relatively more difficult (as depicted in figure 2.5). In the contrary students explore less in relatively easier questions; this phenomenon stands out even more because of the fact that those questions are posed more frequently.

3. Probabilistic Graphical Model for Learning Competence

3.1 Probabilistic Graphical Model for Learning Competence

The use of a learning-aware data-driven application cannot assume that the user's learning competence remains unchanged. In fact, the motivation behind the application itself is at first to monitor it and then to influence it. Simple statistical descriptions as the ones presented in the previous chapter 2.4 are not practical in representing a continuous change and do not effectively capture the differences between the learning process of the users. Furthermore, the purpose is to choose intelligent actions (also called "actionable information" (Barga et al. (2015))) based on the data and this is not possible simply by one rigid and non-adaptive analysis of the data.

The choice of a probabilistic graphical model has several benefits. Firstly, it allows the representation of conditional dependencies (and independencies correspondingly) in the graphical representation of the model of the data. Those are assumed to be the same for all users and stay stable over the course of application usage. Secondly, its parameters (that can be thought of as a configuration or instance) are adaptive and change with each new data sample that is observed. They may be a temporary snapshot description that characterizes the learning competence but unlike the statistics there is an effective way to adapt those and not recompute them from scratch each time the model confronts new data. Thirdly, they've already been extensively used for decision problems (Barga et al. (2015); Kochenderfer (2015)), which are the forefronts of reinforcement learning algorithms.

This chapter is structured as follows: the next section is a brief introduction to probabilistic graphical models. All theoretical aspects that are necessary for understanding the model of student's learning competence are presented; a thoroughly analysis of the topic can be found in Koller and Friedman (2009), Barber (2012). It is also assumed that basic knowledge about probability and in particular random variables is also present; books on the topic are Bertsekas and Tsitsiklis (2008) as well as Papoulis and Pillai (2002). Some preliminary knowledge on graphs is also a prerequisite (Koller and Friedman (2009); Barber (2012)). The following subsections describe the model's structure as well as the reasons and the decision process for

choosing that structure is presented. After this is established, the algorithm that learns the parameters for this particular model structure is analytically solved and used for efficient learning of the models parameter values. When the learning of the parameters is accomplished, the next step is the application of inference in probabilistic queries which is the application of a particular function over the distribution that extracts new insights from the learned model, such as for example the most probable error type of a wrongly answered question for a particular user, which will be described in the next chapter 4.

3.2 Introduction to Probabilistical Graphical Models

Probabilistic Graphical Models are representations of joint probability distributions over random variables that have probabilistic relationships expressed through a graph. The random variables involved can be discrete which have categorical values or continuous with real values. One simple example of a discrete random variable is the one that models the toss of a coin which has two possible outcomes and its represented by the Bernoulli distribution. A continuous random variable on the other hand can be described for example by a Gaussian distribution. Some hybrid models can contain both types of random variables (Kochenderfer (2015)). The set of possible values that a random variable can take - sometimes also referred as the possible outcomes of the experiment described by the random variable - is its domain.

The random variables can be either visible or hidden. The visible ones have outcomes that can be directly observed and their values are contained in the dataset. The hidden variables are defined by human experts using the domain knowledge of the problem, but their outcomes are not directly accessible. They usually represent latent causes of visible random variables and can improve the accuracy and the interpretability of the model (Koller and Friedman (2009)).

To specify the dependencies of the variables in general, one needs to specify their direction, type and intensity. This is made with the use of graphs, which provide the terminology and theory for understanding and reasoning about Prob-

abilistic Graphical Models. The nodes (also called vertices) of the graph represent the random variables and the edges their dependencies, which can be directed or undirected. Undirected models - also called Markov networks - on the other hand represent symmetric probabilistic interactions where there is no dependency with direction, only factors that represent the degree of the strength of the connection. In both cases the graph must be a directed acyclic graph (DAG), otherwise circular reasoning would be possible. These two categories are used in different applications (see section 3.3). This thesis uses a directed model, therefore further information about indirected models is out of scope and can be found in Koller and Friedman (2009) as well as Barber (2012).

The joint distribution represented by the model is mathematically expressed by the chain rule:

$$P(X_1, X_2, \cdots X_N) = \prod_{n=1}^{N} P(X_n | Parents_G(X_n)) \tag{3.1}$$

where $Parents_G(X_n)$ denotes the parents of the X_n random variable in the graph. The $P(X_n|Parents_G(X_n))$ conditional distributions represent local probability models that have their own local likelihood and the estimation of their parameters can be made individually. The equation 3.1 also expresses the factorization of the joint distribution which is also apparent from its graphical representation. The value that a random variable will take is in general dependent on the values of its parent(s) random variable(s). The conditional probability distribution $P(X_n|Parents_G(X_n))$ of each value of the child variable is defined for each value of the parent variable. In the case of discrete random variables, this can be represented by a table which is called conditional probability table (abbreviated CPT); sometimes the two terms are used interchangeably. The number of rows of each CPT equals the number of combinations of all possible values of all parent variable(s) and the number of columns equals the number of values of the child variable. If a random variable has no parents in the graph, then the conditioning of 3.1 does not apply. Each row of a CPT has values that sum up to 1.0.

The dependence and independence of the random variables involved is expressed by the structure of the graph. Independence denotes a situation where knowing about the value of one random variable does not add any new information about the

value of another. Two random variables X and Y are independent iff the equation $P(X,Y) = P(X)P(Y)$ holds. The related concept of conditional independence describes the situation where knowledge about a particular random variable C turns previously dependent variables A and B to independent:

$$P(A,B|C) = P(A|C)P(B|C) \tag{3.2}$$

The equation 3.2 can be also expressed with the following notation: $(A \perp B \mid C)$. The graph structure of the model can be used to find which variables are conditionally independent, even in big complex graphs. In those cases the independence of two random variables A and B is not conditioned on the knowledge of just one variable, but a set of variables which can be denoted with $\mathcal{C}$. The directional or d-separation of A and B with regard to $\mathcal{C}$ is expressed analogously as $(A \perp B \mid \mathcal{C})$. The smallest set of nodes that d-separates a node from all others is called the Markov blanket and can be computed from specific rules that apply to the structure of the model's graph. The detailed description of the rules that define direct and indirect dependencies (ones that persist over intermediate nodes) are out of the scope of this thesis; only the necessary conditional independences of the variables of the model will be presented in 3.4.

The appropriate expression of conditional independences based on the assumptions greatly impacts the performance of the model. Since probabilistic graphical models encode beliefs about phenomena, this can be also expressed by stating that in case of independence the belief remains unchanged.

A large category of Bayesian networks that contain random variables that change over time (for example a Markov chain) in a way that can be described by a stationary transition distribution are Dynamical Bayesian Networks (abbreviated DBN) (Koller and Friedman (2009); Barber (2012)). The proposed Learning Competence model's parameters change over time as further questions are posed. The learning competence of the student at a specific time point is dependent from the one in the previous time point, but the transition is not stationary (unless one considers a random variable as the process of questions selection included in the model). This thesis takes another approach which will be thoroughly described in the following sections.

3.3 Applications of probabilistic graphical models

Probabilistic graphical models are used as probabilistic reasoning systems that can infer and quantify hidden factors, as diseases in medical diagnosis applications from observed information such as illness symptoms (Shwe and Cooper (1991); Jaakkola and Jordan (1999)). A bibliographical review on the use of Bayesian Networks used in fault diagnosis is provided in (Godsey (2017)). In the broader field of undirected probabilistic graphical models, conditional random fields (CRF) are used in image segmentation, taking into account neighbouring properties of the pixels (He et al. (2004)).

Baysian networks are also widely used for human cognition (Goodman et al. (2016); Danks (2014); Lee and Wagenmakers (2013)) and psychometric modelling (Levy and Mislevy (2016)). Recommender systems such as the Matchbox Recommender of the Azure Machine Learning Studio (Stern et al. (2009)) uses a probabilistic graphical model to find latent traits that are common to users and recommended items.

3.4 Model Structure

Domain knowledge about the already described error types that are encountered in one-digit multiplication, as described in the previous chapter 2, was used to define the model. This is in accordance with the data-driven approach of model construction (Koller and Friedman (2009)) where the structure of the model is specified by the designer and the parameters are learned from the data.

A question is either answered correctly or faulty. The student can make one of the following errors: Operand, intrusion, consistency, off-by-$\pm$1 and off-by-$\pm$2, pattern, confusion with addition, subtraction, division or an unclassified error (meaning none of the above). Therefore a multinomial random variable called $\mathbf{Learning\ State_q}$ - individual for each question q was chosen to represent the proportion of each of these misconceptions of the user, when he or she is answering a one-digit multiplication question. The variable follows the categorical distribution:

$$p(x = k) = \theta_k, \sum_{k=1}^{K} \theta_k = 1 \tag{3.3}$$

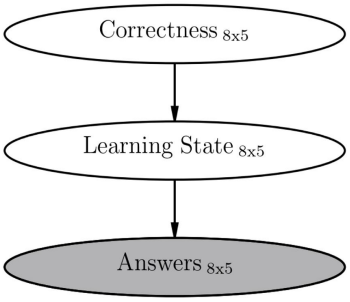
where the possible values are encoded as $1, ..., k$. In this case the $\mathbf{Learning\ State_q}$ has eight possible outcomes and the domain of this random variable is Val($\mathbf{Learning\ State_q}$) = {operand, intrusion, consistency, pattern, confusion, unclassified, correct} (meaning that 1 is the operand error, 2 the intrusion error and so on). The $\mathbf{Learning\ State_q}$ of a specific user can be described for example 5% operand error, 4% consistency error and 91% correct answering (the rest possible outcomes have 0%). This parametrization must be learned from the data.

In the previous chapter it is shown that a specific faulty answer may be classified to more than one error types. Although in reality the model does not assume that more than one error type created a particular answer, the model cannot know a priori which error type was more prevalent and played the decisive role in choosing the wrong answer. The $\mathbf{Learning\ State_q}$ is hidden and the percent of each error type is expected to be learned by the provided answers. Thereby, a dominant error type (for a specific user) can be still discovered and weaken the belief that multiple error types played a role for a specific faulty answer. In chapter 4 the inference of the most probable error type (credit assignment problem) of a specific wrong answer will be made after the learning of the parameters is completed.

As described in section 2.4, the proportion of correct and false answers is different for each question. Even though each question is not posed the same number of times and the belief about the possibility of correctly answering each question is different, this was also taken into account. That means that although the error types distribution is the same for every question, the probability of answering correctly is not. Therefore, there are 90 random variables called $\mathbf{CorrectnessofQuestion_{1\times1}}$ to $\mathbf{CorrectnessofQuestion_{10\times9}}$ (abbreviated by $\mathbf{Correctness_q}$) that have each two possible outcomes. Therefore the Bernoulli distribution was chosen, which is equivalent to a categorical distribution with a domain of two values:

$$P(x|\theta) = \begin{cases} \theta, & \text{if } x = 0 \\ 1 - \theta, & \text{if } x = 1 \end{cases} \tag{3.4}$$

Each question has a distinct random variable, named accordingly as $\mathbf{AnsweringStateofQuestions_{1\times1}}$ to $\mathbf{AnsweringStateofQuestions_{10\times9}}$ (abbreviated as $\mathbf{Answers_q}$), which is a child of the $\mathbf{Learning\ State_q}$ random variable. The arrows from the $\mathbf{Learning\ State_q}$ to its children reflect the dependency of the answer to a question from the misconception or correct understanding of the user. The possible outcomes are all possible answers, as seen in the conditional probability tables of figure 3.1.



wrong	correct
0.5	0.5

	operand	···	unclassified	correct
wrong	1/7	···	1/7	0.0
correct	0.0	···	0.0	1.0

	···	36	37	38	39	40	41	42	43	···
operand	···	0.083	0.0	0.0	0.0	0.0	0.0	0.083	0.0	···
intrusion	···	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	···
consistency	···	0.0	0.0	0.0	0.0	0.0	0.058	0.058	0.058	···
off-by	···	0.0	0.0	0.25	0.25	0.0	0.25	0.25	0.0	···
add/sub/div	···	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	···
pattern	···	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	···
unclassified	···	0.0	0.018	0.0	0.0	0.0	0.0	0.0	0.0	···
correct	···	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	···

Figure 3.1: Conditional Probability Tables (CPT) of the learning competence model. The parameters are uniformly initialized (uninformed prior).

The conditional independence property of each Learning Competence model is expressed by the following equation:

$$\mathbf{Answes_q} \perp \mathbf{Correctness_q} | \mathbf{Learning\ State_q} \tag{3.5}$$

As seen in figure 3.2, the $\mathbf{Correctness_q}$ random variables influences the $\mathbf{Learning\ State_q}$ and in turn $\mathbf{Learning\ State_q}$ is a cause to the particular $\mathbf{Answers_q}$ random variable. The $\mathbf{Learning\ State_q}$ and the $\mathbf{Correctness_q}$ are inferred by the answers of the students in question q.

The joint probability distribution for each question $\mathbf{q}$ has the following factorization:

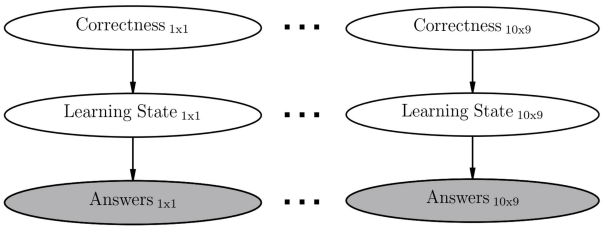


Figure 3.2: The structure of all Probabilistic Graphical models for Learning Competence. The shaded $\mathbf{Answers_q}$ nodes are the ones that are observed, whereas the $\mathbf{Correctness_q}$, $\mathbf{Learning\ State_q}$ random variables remain unobserved.

$$\begin{aligned} P(\mathbf{Correctness_q}, \mathbf{Learning\ State_q}, \mathbf{Answers_q}) = \\ P(\mathbf{Correctness_q})\ P(\mathbf{Learning\ State_q}|\mathbf{Correctness_q}) \\ P(\mathbf{Answers_q}|\mathbf{Learning\ State_q}) \end{aligned} \tag{3.6}$$

As discussed in the previous chapter, each error type can only produce a specific subset of answers, so the others will have zero probability of occurring given this particular error type. Every row of the conditional probability table has values that sum to one and the last row has only one entry with probability 1.0 at the column with the correct answer and 0.0 everywhere else. Figure 3.2 depicts the described structure of learning competence models.

So the model needs to express the following procedure: First knowing if the question is answered correctly; this is provided by the $\mathbf{Correctness_q}$ random variable. If this is true then are no more steps to follow. In the case where the answer is false, there must have been an error, which belongs to the hidden $\mathbf{Learning\ State_q}$. One of the possible answers of this error, as seen and quantified by $\mathbf{Answers_q}$, will be the actual answer of the user. This is a simplified description of the forward sampling procedure that will be described in subsection 4.3.

The model reflects our belief about the overall learning competence of the user. Its structure is considered to be the same for all users, but the conditional probability values (entries in the conditional probability tables) will differ for each individual user. Nevertheless, the model can also reveal similarities between the users, meaning at this stage models that have similar parameter values.

3.5 Learning the Model's Parameters

The students' answers already gathered comprise the data set denoted by $\mathcal{D} = \{d[1], \cdots, d[N]\}$ where $d[n]$ is one data sample (one question-answer pair). The goal of parameter learning is the estimation of the densities of all random variables in the model. It is assumed that the data samples $\mathcal{D}$ are independent and identically distributed (i.i.d.). The joint probability distribution $P_{\mathcal{M}}$ defined by the model $\mathcal{M}$ with parameters Θ is expressed by equation 3.6. The parameter learning's goal is to increase the likelihood of the data given the model: $P(\mathcal{D}|\mathcal{M})$ or equivalently the log-likelihood: $log\, P(\mathcal{D}|\mathcal{M})$ with respect to the set of the parameters Θ of the model. The likelihood expresses the probability of the data given a particular model; a model that assigns a higher likelihood to the data $\mathcal{D}$ approximates the true distribution (the one that has generated the data) better.

To initialize the parameters Θ one needs first to define their prior distribution, which expresses our beliefs about them before seeing any data samples. For example: our beliefs about the outcome of the toss of a coin can be that we will encounter "heads" with probability around 50% and "tails" also with probability 50%. But if we encounter a coin where 9 of 10 tosses the outcome is "heads", we revise our belief of a uniform distribution of the outcomes - we no longer believe that the coin is fair. This revised belief is called posterior distribution and can be made after there was at least one observation in one or more variables of the model. This observation is called evidence. In case of new evidence the old posterior becomes the new prior.

In the case where there are two possible outcomes (as in the coin toss and the $\mathbf{Correctness_q}$ random variables), the phenomenon is described by the Bernoulli distribution 3.4. The prior of the Bernoulli distribution is the Beta distribution:

$$P(\theta|a,b) = \frac{1}{B(a,b)}\theta^{a-1}(1-\theta)^{b-1}, a, b > 0, 0 \leq \theta \leq 1 \tag{3.7}$$

where a and b are the hyperparameters that represent the number of pseudo-counts, meaning the number of times we've encountered each of the two outcomes in previous experiments (or our belief about those outcomes in general). This is the prior of each of the $\mathbf{Correctness_q}$ random variables. The normalization factor $B(a,b)$ 3.8 uses the Gamma function 3.9.

$$B(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \tag{3.8}$$

$$\Gamma(x) = \int_0^\infty u^{x-1}e^{-u}du \tag{3.9}$$

The Dirichlet distribution is the prior of the categorical distribution 3.3. Its density is defined in equation 3.10.

$$Dirichlet(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\Pi_{k=1}^K \Gamma(\alpha_k)} \Pi_{k=1}^K \theta_k^{\alpha_k - 1} \tag{3.10}$$

where the Gamma distribution is defined by equation 3.9, $\boldsymbol{\theta} = \{\theta_1, \cdots, \theta_k\}$ and $\boldsymbol{\alpha} = \{\alpha_1, \cdots, \alpha_k\}$. As in the Beta prior, the α_k are the hyperparameters that represent the number of pseudocounts of each possible outcome of the categorical distribution. This is the prior of the $\mathbf{Learning\ State_q}$ and of each row in the $\mathbf{Answers_q}$ Conditional Probability Table. The number of pseudocounts does not need to be an integer.

In Bayesian parameter estimation the computation of the posterior distribution with regard to the prior is computed with the Bayes rule 3.11:

$$\overbrace{P(\Theta|\mathcal{D})}^{posterior} = \frac{\overbrace{P(\mathcal{D}|\Theta)}^{likelihood}\overbrace{P(\Theta)}^{prior}}{\underbrace{P(\mathcal{D})}_{marginal\ likelihood}} \tag{3.11}$$

$P(\mathcal{D}|\Theta)$ is the likelihood of the data given the parameters Θ and expresses the probability that the current model (seen as a generative model) has created the data $\mathcal{D}$. The marginal likelihood is an integral over all possible values of the parameters Θ and it is a normalising factor in equation 3.11:

$$P(\mathcal{D}) = \int_\Theta P(\mathcal{D}|\theta)P(\theta)d\theta \tag{3.12}$$

As seen from the equation 3.11, the strength of our belief about the experiment, before seeing data, influences the posterior distribution. Since there is no information gathered other than the actual answers of the questions, there was no other way to set the prior for each user than to take some data from the users and learn

the parameters from them. The initialization of the prior parameters of this first training set is made uniformly (non-informative uniform prior). The dataset contains users that answered a different number of questions because of the existing sequencing algorithm. Once the parameters are learned from the dataset, the prior of the individualized model of each user is initialized to these values (the posterior of the first learning phase becomes the prior of the next).

Since the prior reflects the beliefs, a typical starting point for the prior is a symmetric Dirichlet where all α_k are equal to 1, for example for a six-sided dice: $Dirichlet(1, 1, 1, 1, 1, 1)$. The parameters in this case have the uniform distribution $(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6})$. Another symmetric Dirichlet resulting in the same distribution is the $Dirichlet(100, 100, 100, 100, 100, 100)$ (in general every symmetric Dirichlet will result in this distribution). Bayesian learning differentiates these two cases, because in the second one more data will be necessary to make the parameters shift from the uniform distribution. This characteristic applies to the Beta prior as well.

It can be shown (Koller and Friedman (2009)) that the posterior of the Dirichlet distribution $P(\theta) = Dirichlet(a_1, \cdots, a_k)$, after seeing $M[k]$ outcomes of each possible outcome k, is expressed by:

$$P(\theta|\mathcal{D}) = Dirichlet(a_1 + M[1], \cdots, a_k + M[k]) \tag{3.13}$$

In this case the posterior has the same form as the prior and it is called conjugate prior. This property holds for the Beta and the Bernoulli distribution as well; the Beta distribution is the conjugate prior of the Bernoulli distribution. By applying the 3.13 equation, a big enough number of samples of each class can diminish the influence of the prior and make our belief converge to the true distribution (Koller and Friedman (2009)).

The uniform prior is a weak prior. That means that the data that will be used in for the first informed prior will change the model's parameters (that represent our beliefs) a lot. After that, the model will have a stronger prior, which will need much more data for its values to change substantially. The whole model (containing both the prior and the posterior as random variables) is called meta-model. Learning the parameters of this model is an inference task since the model has hidden random variables (Kochenderfer (2015)).

Sparse data (zero-count problem) may create overfitting as well as the impression that a specific outcome does not occur. The model of Learning Competence exhibits this issue too; as seen in the previous chapter, the proportion of correctly answered questions dominates with regard to the wrongly answered ones 2.4 and some answers were never provided when specific questions were posed 2.5. The prior helps tempering this problem (Murphy (2012)).

3.6 Learning the Model's Parameters with batch Expectation-Maximization (EM)

Bayesian parameter learning is applicable when all variables are visible; in that case all components of the equation 3.11 are computable. The model of the learning competence contains hidden variables, therefore the computation of the posterior of each random variable cannot be made directly. In this case the method that is used is expectation-maximization (abbreviated by EM). The concepts of prior, likelihood and posterior that were described in the previous section are used in the description of this method.

3.6.1 Notation

The entities that are necessary for the analytical solution for the computation of the posterior distributions of all model's variables are the following:

N : Number of all samples in dataset

n : One sample of N

M : Number of $\mathbf{Correctness_q}$ possible outcomes ($M = 2$)

μ : Index of $\mathbf{Correctness_q}$ outcome

w_μ : Parameters of $\mathbf{Correctness_q}$

K : Number of $\mathbf{Learning\ State_q}$ possible error types and correct outcome ($K = 8$)

k : Index of $\mathbf{Learning\ State_q}$ outcome (one error type out of $K - 1$ or correct)

$\pi_{k|\mu}$: Parameters of the $\mathbf{Learning\ State_q}$ variable

Q : Number of $\mathbf{Answers_q}$ random variables (90 in total)

q : Index of $\mathbf{Answers_q}$ (one question of Q)

X : Number of all possible answers of each question (columns of conditional probability tables of $\mathbf{Answers_{1\times1}}$ to
$\mathbf{Answers_{10\times9}}$)

x : one answer out of X

x_n : the answer of the n-th sample

$\theta_{x|k}$: The parameters of the $\mathbf{Correctness_q}$ random variable

Θ : All current parameters of the model : (set of all w_μ, $\pi_{k|\mu}$, $\theta_{x|k}$).

Θ_{old} : All parameters of the previous EM iteration.

$\mathbf{X}$: The set of all visible variables. In this model, they are all
$\mathbf{Answers_q}$ variables.

$\mathbf{Z}$: The set of all latent variables or hidden causes. In this model, they are all $\mathbf{Correctness_q}$ and $\mathbf{Learning\ State_q}$ variable.

3.6.2 Expectation-Maximization (EM) algorithm

The goal of the EM-Algorithm is to find appropriate values for all parameters Θ. In general a better model will fit the data better, although it must not overfit. The latent variables $\mathbf{Correctness_q}$ and $\mathbf{Learning\ State_q}$ are not observed, so the direct maximization of the likelihood $P(\mathbf{X};\Theta)$ of the data according to this model is not possible; the observed data $\mathbf{X}$ (not to be confused with the number of possible answers of each question X) is incomplete. Each iteration of the EM-algorithm computes a different instantiation of the table CPDs.

By using marginalization:

$$P(\boldsymbol{X};\Theta) = \sum_{\boldsymbol{Z}} P(\boldsymbol{X},\boldsymbol{Z};\Theta) \tag{3.14}$$

$$\ln P(\boldsymbol{X};\Theta) = \ln\left\{\sum_{\boldsymbol{Z}} P(\boldsymbol{X},\boldsymbol{Z};\Theta)\right\} \tag{3.15}$$

If the complete data set $\{\boldsymbol{X},\boldsymbol{Z}\}$ were known, then it would be straightforward to try to maximize the complete data log-likelihood. To avoid multiplication of very

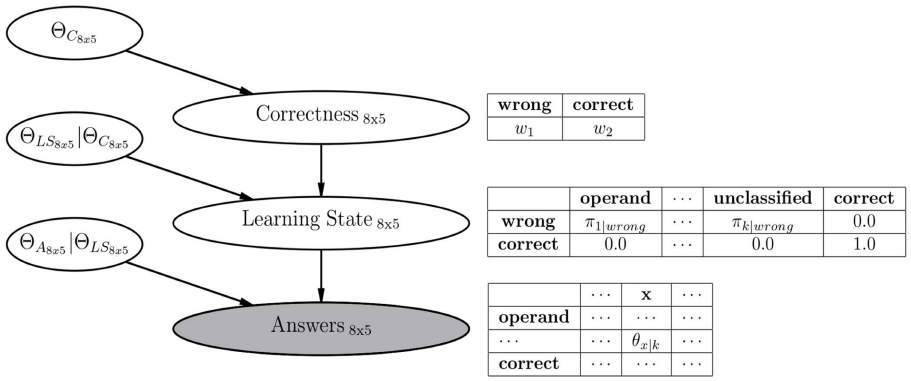


Figure 3.3: Parameters of the Learning Competence Probabilistic Graphical Model

small floating point numbers that can lead to zero, one can equivalently maximize the log-likelihood function $\ln P(\boldsymbol{X};\Theta)$.

The EM-Algorithm works iteratively and consists of four steps:

1. Initialization of all parameters to Θ_0 of the complete dataset $\{\boldsymbol{X},\boldsymbol{Z}\}$ and set $\Theta_0 = \Theta_{old}$.

2. E-Step: Computation of the posterior distribution $P(\boldsymbol{Z}|\boldsymbol{X};\Theta_{old})$ of $\mathbf{Z}$ given the visible variables and the previous parameters.

3. M-Step: Compute new Θ parameters by trying to maximize 3.17 the expected value of the posterior distribution 3.16 over the latent variables $\boldsymbol{Z}$:

$$\mathcal{Q}(\Theta,\Theta_{old}) = \sum_{\boldsymbol{Z}} P(\boldsymbol{Z}|\boldsymbol{X};\Theta_{old})\ \ln P(\boldsymbol{X},\boldsymbol{Z};\Theta) \tag{3.16}$$

$$\Theta = \underset{\Theta}{\operatorname{argmax}}\ \mathcal{Q}(\Theta,\Theta_{old}) \tag{3.17}$$

4. Compute the incomplete data likelihood $P(\boldsymbol{X};\Theta)$ or equivalently the log-likelihood $\ln P(\boldsymbol{X};\Theta)$ 3.15. If the log-likelihood's increase or the Θ parameters' change is not significant compared to the previous iteration, then stop. Else, set current Θ with the values computed in M-Step and return to E-Step. The EM-algorithm is a "meta-algorithm" since it contains an inference in the E-Step (Pfeffer (2016)). The iterative process is depicted in figure 3.4.

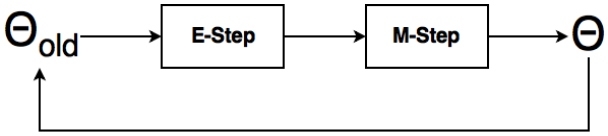


Figure 3.4: The iteration loop of the EM-algorithm

3.6.3 Analytical Solution of Expectation-Maximization (EM) for the model of Learning Competence

The steps of the EM-algorithm are applied to the model of the Learning Competence for the derivation of the analytical solution for the update of the parameters. We apply those steps to the model of Learning Competence of each question q separately. The equations in this subsection omit the subscript q; they apply to the model of each question independently.

The equation 3.18 expresses the joint probability distribution equation 3.19 is derived from:

$$P(\boldsymbol{X}, \boldsymbol{Z}; \Theta) = \prod_{n} \prod_{\mu} \prod_{k} w_{\mu} \pi_{k|\mu} \theta_{x_n|k} \tag{3.18}$$

$$\ln P(\boldsymbol{X}, \boldsymbol{Z}; \Theta) = \sum_{n=1}^{N} \ln \left(\sum_{\mu=1}^{M} \sum_{k=1}^{K} w_{\mu} \pi_{k|\mu} \theta_{x_n|k} \right) \tag{3.19}$$

The expected value of the complete log-likelihood $\mathcal{Q}(\Theta, \Theta_{old})$ is:

$$\begin{aligned} \mathcal{Q}(\Theta, \Theta_{old}) = \mathbb{E}_{P(\boldsymbol{Z}|\boldsymbol{X};\Theta_{old})} \left[\ln P(\boldsymbol{X}, \boldsymbol{Z}; \Theta) \right] = \\ \sum_{\boldsymbol{Z}} P(\boldsymbol{Z}|\boldsymbol{X}; \Theta_{old}) \ln P(\boldsymbol{X}, \boldsymbol{Z}; \Theta) = \\ \sum_{n} \sum_{\mu} \sum_{k} \gamma(z_{\mu k}^{(n)}) \left(\ln w_{\mu} + \ln \pi_{k|\mu} + \ln \theta_{x_n|k} \right) \end{aligned} \tag{3.20}$$

The responsibility $\gamma(z_{\mu k}^{(n)})$ of the hidden error cause or correct k for $n-$th sample coupled with the probability of the answer being answered correctly, can be computed by using the Bayes rule and the factorization of the joint probability distribution from equation 3.6:

$$\gamma(z_{\mu k}^{(n)}) = P(z_{\mu k}^{(n)}|x^{(n)};\Theta_{old}) \propto P(x^{(n)}|z_{\mu k}^{(n)};\Theta_{old})P(z_{\mu k}^{(n)};\Theta_{old}) \tag{3.21}$$

The values of all $\gamma(z_{\mu k}^{(n)})$ values in equation 3.21 are provided up to a normalization factor. Since $\gamma(z_{\mu k}^{(n)})$ depends only on Θ_{old}, it can be considered a constant in the process of maximization of $\mathcal{Q}$. At the same time following constraints that reflect the conditional probability rules must be fulfilled:

$$\sum_{\mu=1}^{M} w_\mu = 1 \tag{3.22}$$

$$\sum_{k=1}^{K} \pi_{k|\mu} = 1 \tag{3.23}$$

$$\sum_{x=1}^{X} \theta_{x|k} = 1 \tag{3.24}$$

The maximization of the complete log-likelihood $\mathcal{Q}(\Theta, \Theta_{old})$ leads to the parameters of the model. The maximization process must also fulfil the constraints in equations 3.22, 3.23, 3.24, which can be made with the use of Lagrange Multipliers. The maximum of the following expression must be found:

$$\mathcal{Q}(\Theta, \Theta_{old}) + \lambda\left(\sum_{\mu} w_\mu - 1\right) + \sum_{\mu}\lambda_\mu\left(\sum_{k}\pi_{k|\mu} - 1\right) + \sum_{k}\lambda_k\left(\sum_{x}\theta_{x|k} - 1\right) \tag{3.25}$$

First, the update of the parameters of a particular $\mathbf{Correctness_q}$ value $w_{m|q}$, is made from the data samples n' that have as question $q = q_{n'} \in [q_1 \cdots q_Q]$, and answer m being either correct or wrong:

$$\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')})\frac{1}{w_{m|q}} + \lambda \stackrel{!}{=} 0 \quad \| \cdot w_{m|q} \tag{3.26}$$

$$\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) + \lambda w_{m|q} \stackrel{!}{=} 0 \quad \| \sum_{m=1}^{M} \tag{3.27}$$

$$\sum_{m=1}^{M}\sum_{n'=1}^{N'}\gamma(z_{\mu k}^{(n')}) + \lambda\sum_{m=1}^{M} w_{m|q} \stackrel{!}{=} 0 \tag{3.28}$$

$$\lambda = -\sum_{m=1}^{M}\sum_{n'=1}^{N'}\gamma(z_{\mu k}^{(n')}) = -N' \tag{3.29}$$

because :

$$\sum_{m=1}^{M}\gamma(z_{\mu k}^{(n')}) = 1 \tag{3.30}$$

$$w_{m|q} = \frac{\sum_{n'=1}^{N'}\gamma(z_{\mu k}^{(n')})}{N} \tag{3.31}$$

Secondly, the maximization with respect to a particular $\pi_l \in [\pi_1 \cdots \pi_K]$ is computed. The derivative must be set to 0 and all parameters of the expression 3.25 not related to π_l can be eliminated as constants. If the number of samples that are answered wrongly is N', the following steps provide the analytical solution for the update rule for any π_k:

$$\sum_{n'=1}^{N'}\gamma(z_{\mu k}^{(n')})\frac{1}{\pi_l} + \lambda_\mu \stackrel{!}{=} 0 \quad \| \cdot \pi_l \tag{3.32}$$

$$\sum_{n'=1}^{N'}\gamma(z_{\mu k}^{(n')}) + \lambda_\mu \pi_l \stackrel{!}{=} 0 \quad \| \sum_{k=1}^{K} \tag{3.33}$$

$$\sum_{n'=1}^{N'}\gamma(z_{\mu k}^{(n')}) + \lambda_\mu \sum_{k=1}^{K}\pi_l \stackrel{!}{=} 0 \tag{3.34}$$

$$\lambda_\mu = -\sum_{k=1}^{K}\sum_{n'=1}^{N'}\gamma(z_{\mu k}^{(n')}) \tag{3.35}$$

$$\pi_k = \frac{\sum_{n'=1}^{N'}\gamma(z_{\mu k}^{(n')})}{\sum_{k=1}^{K}\sum_{n'=1}^{N'}\gamma(z_{\mu k}^{(n')})} \tag{3.36}$$

Thirdly, the maximization with respect to $\theta_{x|q,k,}$ is performed in a similar manner. The update of the parameters of a particular

Answers$_\mathbf{q}$ value $\theta_{x|q,k}$, is made from the data samples n' that have as question $q = q_{n'} \in [q_1 \cdots q_Q]$, and answer $x = x_{n'} \in [x_1 \cdots x_X]$:

$$\sum_{n'=1}^{N'} \frac{\gamma(z_{\mu k}^{(n')})}{\theta_{x|k}} + \lambda_k \stackrel{!}{=} 0 \quad \| \cdot \theta_{x|k} \tag{3.37}$$

$$\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) + \lambda_k \theta_{x|k} \stackrel{!}{=} 0 \quad \| \sum_{x=1}^{X} \tag{3.38}$$

$$\sum_{x=1}^{X} \sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) + \lambda_k \sum_{x=1}^{X} \theta_{x|k} \stackrel{!}{=} 0 \tag{3.39}$$

$$\lambda_k = -\sum_{x=1}^{X} \sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) \tag{3.40}$$

$$\theta_{x|k} = \frac{\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')})}{\sum_{x=1}^{X} \sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')})} \tag{3.41}$$

The steps of the EM-Algorithm for updating the parameters of this Bayesian Model are as follows:

1. Initialization of all parameters Θ_0. In this case that is the uniform prior as discussed in section 3.5.

2. E-Step: Computation of $\gamma(z_{\mu k}^{(n)})$ using equation 3.21

3. M-Step: Compute new Θ parameters $w_{\mu|q}$, π_k and $\theta_{x|k}$ using equations 3.31, 3.36 and 3.41

4. Compute the likelihood $P(\boldsymbol{X}; \Theta)$ or log-likelihood $\ln P(\boldsymbol{X}; \Theta)$:

$$\begin{aligned} P(\mathbf{Correctness_q}, \mathbf{Answers_q}; \Theta) = \\ \frac{P(\mathbf{Correctness_q}, \mathbf{LearningState_q}, \mathbf{Answers_q}; \Theta)}{P(\mathbf{LearningState_q} | \mathbf{Correctness_q}, \mathbf{Answers_q}; \Theta)} \\ P(\mathbf{Correctness_q}; \Theta) P(\mathbf{Correctness_q} | \mathbf{LearningState_q}; \Theta) \end{aligned} \tag{3.42}$$

If the likelihood or the parameters values do not converge, then set current Θ with the values computed in M-Step and goto E-Step.

Figure 3.5 depicts the steps of the EM-algorithm updating procedure. The dataset used for training is called training set. Its characteristics will be explained in the next section 3.7.

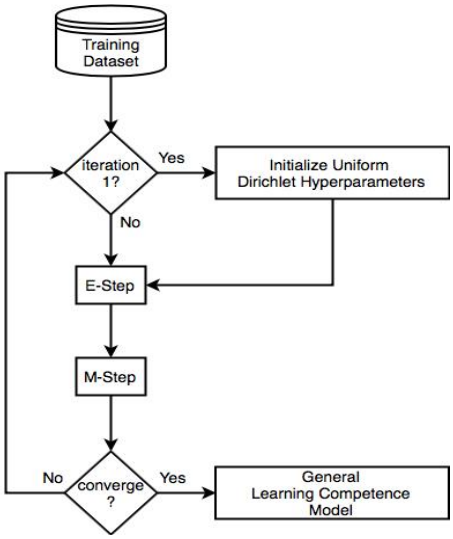


Figure 3.5: Expectation-maximization training algorithm applied on the training set. In the first step the uniform Dirichlet hyperparameters are used. Then alternating E- and M-steps bring the parameters/log-likelihood convergence.

It is proven that the EM-algorithm increases the log-likelihood of the observed data X at each iteration (Bishop (2006)):

$$\ln P(\boldsymbol{X};\Theta) \geq \ln P(\boldsymbol{X};\Theta_{old}) \tag{3.43}$$

The procedure of updating the log-likelihood in this manner is shown to guarantee convergence to a stationary point, which can be a local minimum, local maximum or saddle point. Fortunately, by initializing the iterations from different starting Θ_0 and injecting small changes to the parameters, the local minima and saddle points can be avoided (Koller and Friedman (2009)).

3.7 Datasets

The available data were divided into a training and test set, with a dataset containing data from users that have answered all the questions at least one time (The number of users that have answered all questions exactly once is 2218). The diagram in figure

3.6 describes the main computational blocks of the dataset splitting procedure. It is important to emphasize that the same preprocessing steps described in chapters 2 and 3 are applied in both datasets.

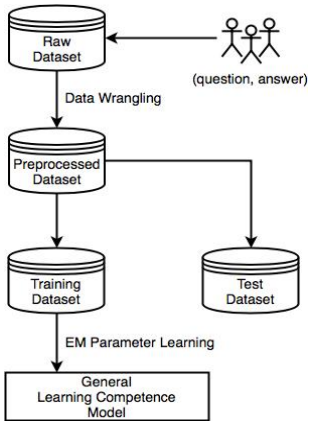


Figure 3.6: Dataset splitting procedure: After the same data preprocessing functionalities are applied to the whole dataset, the parameters are defined by the training set only.

When a new student starts to use the application, it is assumed that he or she will have some commonalities with the other users; these commonalities are already captured in the learning competence model. The application does not have to start from an agnostic point, nor to have questionnaires or time-consuming demographic registration forms. The learning competence model is learned and can be updated from that point on individually for each student depending on the provided answers. The procedure is depicted in figure 3.7:

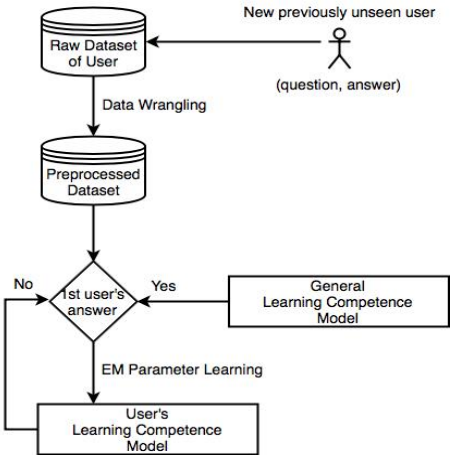


Figure 3.7: Handling a previously unseen user: the new data pass through the standardized preprocessing module. The learning competence model defined by the training set is used for modelling. After the first question is answered, this model is updated individually according to the answer(s) of this particular user.

The models' parameters are computed by the EM-algorithm on the training data. After 4 iterations the likelihood of the training set increases, but the likelihood of the test set decreases, which consists an indication of overfitting, as described in the section 3.9.

3.8 Fractional Updating

Since the learning application proposes questions continuously, it is important to update the beliefs about the learning competence of the student as soon as an answer is present. As new evidence is observed - in the form of answered questions - the model shifts the value of the parameters to reflect the fact that the belief about the learning competence of the user is changed.

With fractional updating (Jensen and Nielsen (2007)), the initialization and updating of the parameters is made by means of the Dirichlet pseudocounts, which are explained in section 3.5. The starting pseudocount number is set to 1.0 to express a weak belief about the learning competence of the student. In this application, for each question-answer pair, only one probabilistical graphical model needs to be updated. The data sample only contains the value of the corresponding observed variable $\mathbf{Answers_q}$. The update of the pseudocounts α is provided by equation:

$$\alpha_{ijk}^{l+1} = \alpha_{ijk}^{l} + P(\boldsymbol{X_i} = k, Parents_G(\boldsymbol{X_i}) = j|\mathcal{D}) \tag{3.44}$$

where the current joint probability of the updated variable and the value of its parents are used to update the value of the pseudocounts, which may no longer be an integer. $\mathcal{D}$ denotes the dataset of samples.

The fractional updating procedure can be explained by an example where a student provides the wrong answer 42 to the question 8×5. The probabilities start with the following values:

wrong	**correct**
$\frac{1}{2}$	$\frac{1}{2}$

operand	**intrusion**	**consistency**	**off-by**	**add/sub/div**	**pattern**	**unclassified**
$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$

	$\cdots$	**42**	$\cdots$
operand	$\cdots$	$\frac{1}{12}$	$\cdots$
intrusion	$\cdots$	0	$\cdots$
consistency	$\cdots$	$\frac{1}{17}$	$\cdots$
off-by	$\cdots$	$\frac{1}{4}$	$\cdots$
add/sub/div	$\cdots$	0	$\cdots$
pattern	$\cdots$	0	$\cdots$
unclassified	$\cdots$	0	$\cdots$

Table 3.1: Probabilistic graphical model of the question 8×5 where all conditional probabilities (all rows of the conditional probability tables) are set uniformly.

The corresponding pseudocounts start with the following values:

wrong	**correct**
1	1

operand	**intrusion**	**consistency**	**off-by**	**add/sub/div**	**pattern**	**unclassified**
1	1	1	1	1	1	1

	···	**26**	···
operand	···	1	···
intrusion	···	0	···
consistency	···	1	···
off-by	···	1	···
add/sub/div	···	0	···
pattern	···	0	···
unclassified	···	0	···

Table 3.2: Probabilistic graphical model of the question 8×5 where all pseudocounts are set to value 1.

There are three error types that can cause the answer 42. The weights for each case, corresponding to the entity $P(\boldsymbol{X_i} = k, Parents_G(\boldsymbol{X_i}) = j|\mathcal{D})$ of the equation 3.44, are computed as follows:

$\frac{1}{2}\frac{1}{7}\frac{1}{12}/(\frac{1}{2}\frac{1}{7}\frac{1}{12} + \frac{1}{2}\frac{1}{7}\frac{1}{17} + \frac{1}{2}\frac{1}{7}\frac{1}{4}) = 0.21259$

$\frac{1}{2}\frac{1}{7}\frac{1}{17}/(\frac{1}{2}\frac{1}{7}\frac{1}{12} + \frac{1}{2}\frac{1}{7}\frac{1}{17} + \frac{1}{2}\frac{1}{7}\frac{1}{4}) = 0.15006$

$\frac{1}{2}\frac{1}{7}\frac{1}{4}//(\frac{1}{2}\frac{1}{7}\frac{1}{12} + \frac{1}{2}\frac{1}{7}\frac{1}{17} + \frac{1}{2}\frac{1}{7}\frac{1}{4}) = 0.63776$

The value of the updated pseudocounts of the **operand**, **intrusion**, and **consistency** error types, are presented in the following table:

	$\mathbf{CQ_{8\times5}}$	$\mathbf{LS_{8\times5}}$	$\mathbf{AS_{8\times5}}$	pseudocounts	probability
$D_{8\times5,\ 42,\ operand}$	wrong	operand	42	$1 + 0.21259$	0.151
$D_{8\times5,\ 42,\ consistency}$	wrong	consistency	42	$1 + 0.15006$	0.144
$D_{8\times5,\ 42,\ off-by}$	wrong	off-by	42	$1 + 0.63776$	0.205

The pseudocounts of the rest of the error types will remain to the value 1, so the total sum of pseudocounts that will be used as normalization value is: $4 \times 1 + 1.21259 + 1.15006 + 1.63776 = 8.0004$. The actual probabilities of the involved error types are listed in the table above; the rest have the value 0.12499. The pseudocounts of "wrong" are increased by +1, setting the probability of "wrong" to $\frac{2}{3}$ and "correct" to $\frac{1}{3}$. The values of the $\mathbf{Answers_{8x5}}$ random variable are computed accordingly.

This comprises a full iteration of the online EM-algorithm. In the next iteration,

the newly computed pseudocounts will play the role of the prior that needs updating. Drawbacks and extensions of the fractional updating algorithm can be found in the work of Jensen and Nielsen (2007) and are out of scope of this thesis.

3.9 Evaluation of Parameter Learning

The evaluation of the parameter learning EM-algorithm is firstly made by computing the likelihood of a training set (see section 3.7) at each iteration. It is expected that the likelihood is increasing monotonically and converging with increasing number of iterations. As seen in figure 3.8, this also applies for the likelihood of the models as a whole.

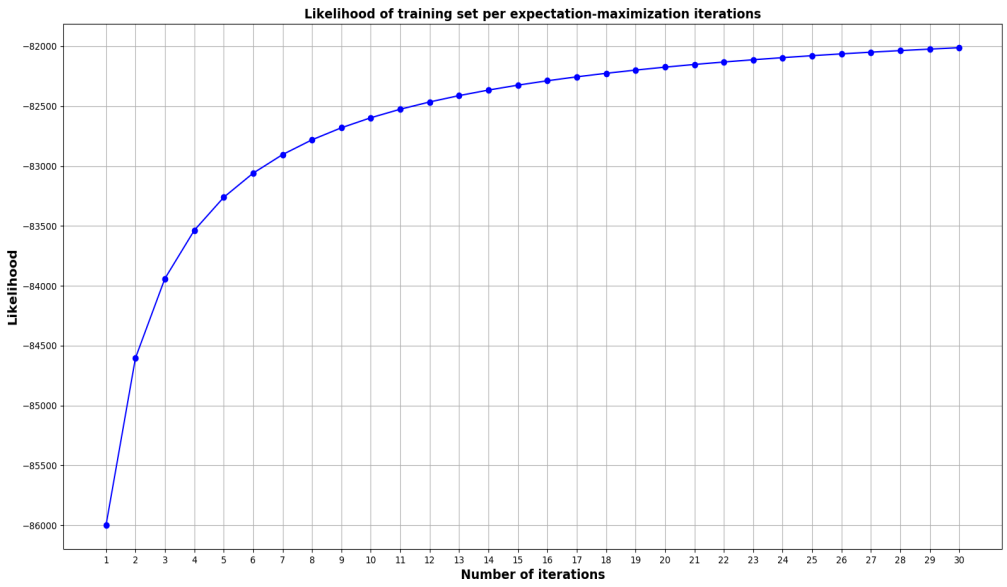


Figure 3.8: Evolution of the likelihood of the training set with respect to the number of EM-iterations in the training set.

If the EM-algorithm is repeated for many iterations, the values of the parameters will be adjusted too much to the training set, without being able to generalize to the properties of the test set; an unseen user's learning competence must be also modelled sufficiently by the model. Figure 3.9 depicts the evolution of the likelihood of the test set, with respect to the number of EM-iterations of the training set. The likelihood of the test set decreases after 4 iterations; this is an indication of overfitting

Bishop (2006).

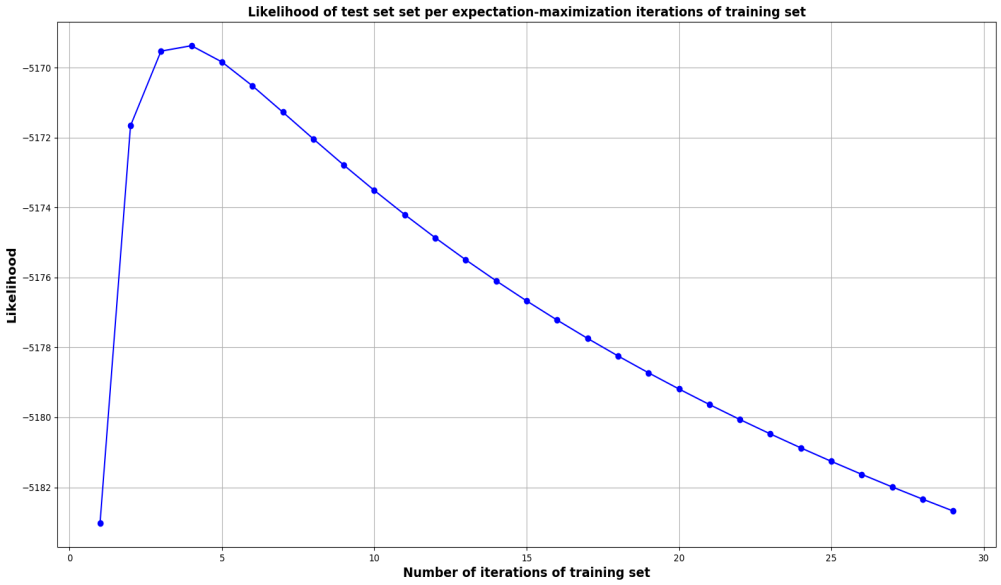


Figure 3.9: Evolution of the likelihood of the test set with respect to the number of EM-iterations in the training set. The test set contains 1000 samples.

The expectation-maximization algorithm bases on the fact that all possible outcomes of all questions present at least one sample of the dataset. This was not the case in this application; the sufficient statistics condition was not fullfiled (Koller and Friedman (2009), Murphy (2012)). Nevertheless, it has been shown that the models performance is sufficient for practical purposes and that as new data are gathered, this problem might be solved.

4. Insights

4.1 Reasoning types in Probabilistic Graphical Models

After the model of a particular student is learned - by using the informed prior as starting point and as evidence the answers he or she has given so far - a typical application would be the prediction of the answer for a particular question. The better and more accurate the model captures the learning competence of the student, the better the performance of the predictions of the answers will be. In some probabilistic modelling frameworks such as Figaro [1] the parameter learning part is made by the offline component and the probability queries by the online component.

There are three types of reasoning one can make with probabilistic graphical models: causal, evidential and explaining away. Causal reasoning (also called prediction) consists of statements that start with the knowledge of the causes as evidence and provide information about the effects. In our model this would be possible if the $\mathbf{Correctness_q}$ and $\mathbf{Learning\ State_q}$ were known: the computation of the answer to a posed question would be accurately determinable. The direction of causal reasoning in directed graphical models goes from parent to child variables ("downstream") in general and is used to predict future events.

Evidential reasoning (also called explanation) on the other hand has the opposite direction and involves situations where effects lead to the specification of causes. This is the most important reasoning in our case because the answers of the students provide the information to do evidential reasoning and learn the hidden variables $\mathbf{Correctness_q}$ and $\mathbf{Learning\ State_q}$, which in turn can be used for causal

[1] `https://www.cra.com/work/case-studies/figaro`, Last accessed 17 March 2019

reasoning to predict the future answers of each student. The difference in causal and evidential reasoning can be understood by considering the direction of time; evidential reasoning infers the past probability distribution from the current set of data whereas causal reasoning makes a prediction for the future given the data. The great benefit of graphical models over statistics is that the same model is used for both backward and forward reasoning (with respect to the perception of time).

Intercausal reasoning occurs when one random variable depends on two or more parents. In this case, the observation of the value of one parent influences the belief about the value of the other(s) (either strengthen or weaken). In this situation it is said that one reason explains away the other. The learning competence model's structure does not contain such cases; further discussion about this reasoning type can be found in Karkera (2014), Koller and Friedman (2009), Bishop (2006).

The upcoming sections proceed with an analytical implementation of probabilistic queries, which is specific to the designed learning competence models. Personalized insights computed by the latent explanations of wrong answers of each student are made possible by exact and efficient inference as described in section 4.2.1. Furthermore, the probability distributions of selected variables can be compared to validate the assumptions on which the model itself is built, as well as quantitatively verify learner's hypotheses of the general student behaviour described in section 4.5.

4.2 Probability Queries

A conditional probability query $P(\boldsymbol{Y}|\boldsymbol{E} = e)$ - also called probabilistic inference - computes the posterior of the subset of random variables represented by $\boldsymbol{Y}$ (target of query) given observations e of the subset of evidence variables denoted by $\boldsymbol{E}$ (of course there may be a subset of variables $\boldsymbol{Z}$ in the model not belonging to either of these two subsets). By using the Bayes rule, the conditional probability is written as:

$$P(\boldsymbol{Y}|\boldsymbol{E} = e) = \frac{P(\boldsymbol{Y}, e)}{P(e)} \tag{4.1}$$

The MAP query, which is also called most probable explanation (MPE) (Koller and Friedman (2009); Pfeffer (2016)), is a query that maximizes the posterior of the

joint distribution of a subset of random variables $\boldsymbol{Y}$:

$$\text{MAP}(\boldsymbol{Y}|\boldsymbol{E}=e) = \arg\max_{y} P(y,e) \tag{4.2}$$

In the case of MAP Query the whole set of random variables is $\mathcal{X} = \{\boldsymbol{Y}, \boldsymbol{E}\}$. In other words the MPE, after observing (clamping) a subset of variables, it computes the most likely values of the rest of them jointly.

A slightly different query is the marginal MAP which is written as follows:

$$\begin{aligned}\text{Marignal MAP}(\boldsymbol{Y}|\boldsymbol{E}=e) = \arg\max_{y} P(y|e) = \\ \arg\max_{\boldsymbol{Y}} \sum_{\boldsymbol{Z}} P(\boldsymbol{Y}, \boldsymbol{Z}|\boldsymbol{E}=e)\end{aligned} \tag{4.3}$$

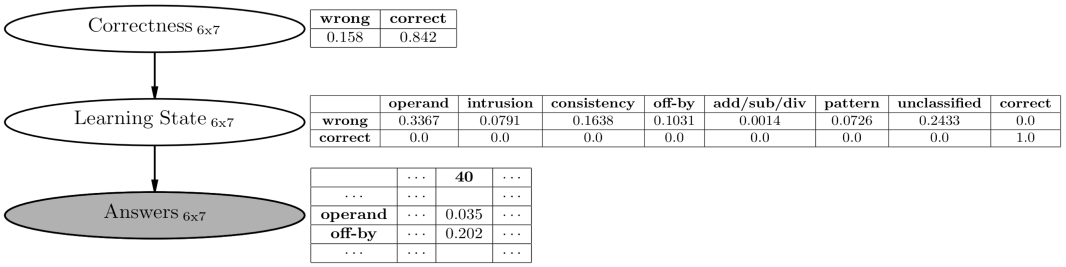
which directly follows from the fact that $\mathcal{X} = \{\boldsymbol{Y}, \boldsymbol{E}, \boldsymbol{Z}\}$.

The computation of the query result can be made with the variable elimination algorithm, which is described in the following section 4.2.1; in this case, the exact value of equation 4.1 is computed by dividing $P(y,e) = \sum_w P(y,e,w)$ and $P(e) = \sum_y P(y,e)$. Alternatively, the the normalization of a vector containing all $P(y^k, e)$ (where y^k are all possible outcomes of the variables $\boldsymbol{Y}$) so that it has sum that equals to one, provides also the desired result. For more complex bayesian networks approximate inference algorithms are applied since exact inference is NP-hard (Kochenderfer (2015)), but even those can be in the worst-case also NP-hard (Koller and Friedman (2009)). The learning competence models are simple and the variable elimination algorithm is fast enough.

4.2.1 Variable elimination in the Learning Competence Model

The probability query that is of relevance for the tutors is the probability of error types regarded as causes of a specific wrong answer. The sum of products expression 4.4 computes the distribution of the $\textbf{Learning State}_\textbf{q}$ by means of the joint distribution $P(\mathbf{C_q}, \mathbf{LS_q}, \mathbf{A_q})$:

$$P(\mathbf{LS_q}) = \sum_{\mathbf{C_q},\mathbf{A_q}} P(\mathbf{C_q}, \mathbf{LS_q}, \mathbf{A_q}) = \sum_{\mathbf{C_q}} P(\mathbf{C_q}) P(\mathbf{LS_q}|\mathbf{C_q}) \sum_{\mathbf{A_q}} P(\mathbf{A_q}|\mathbf{LS_q}) \quad (4.4)$$



wrong	correct
0.158	0.842

	operand	intrusion	consistency	off-by	add/sub/div	pattern	unclassified	correct
wrong	0.3367	0.0791	0.1638	0.1031	0.0014	0.0726	0.2433	0.0
correct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

	···	**40**	···
···	···		···
operand	···	0.035	···
off-by	···	0.202	···
···	···		···

Figure 4.1: Parameters of learning competence model of question 6×7 that are relevant to the computation of the MAP query when the answer is 40

The first step of the Variable Elimination algorithm, in case it is applied where an evidence exists, is to compute the unnormalized joint distribution $P(\mathbf{C_{6\times7}}, \mathbf{LS_{6\times7}}, \mathbf{A_{6\times7}} = 40)$. For example, the faulty answer 40 for the question 6×7 eliminates all cases for which the answer is not equal to 40; it can belong only to two potential error types: consistency and off-by. The remaining rows of the joint distribution - those containing the unnormalized proportion unequal to 0 are listed in table 4.1. The computations use the corresponding parameters of the learning competence model of question 6×7 depicted in figure 4.1.

$\mathbf{C_{6\times7}}$	$\mathbf{LS_{6\times7}}$	$\mathbf{A_{6\times7}}$	unnormalized proportions
wrong	**operand**	40	$0.158 \cdot 0.336 \cdot 0.035 = 1.85 \cdot 10^{-3}$
wrong	**off-by**	40	$0.158 \cdot 0.103 \cdot 0.202 = 3.28 \cdot 10^{-3}$

Table 4.1: Unnormalized joint distribution $P(\mathbf{C_{6\times7}}, \mathbf{LS_{6\times7}}, \mathbf{A_{6\times7}} = 40)$

The sum of the unnormalized proportions, $1.85 \cdot 10^{-3} + 3.28 \cdot 10^{-3} = 5.14 \cdot 10^{-3}$ (which is the value of $P(\mathbf{A_{6\times7}} = 40)$), can be used to compute the normalized probabilities of the causes of answer 40 as depicted in table 4.2.

$\mathbf{C_{6\times7}}$	$\mathbf{LS_{6\times7}}$	$\mathbf{A_{6\times7}}$	normalized probabilities
wrong	**operand**	40	$1.85 \cdot 10^{-3} / 5.14 \cdot 10^{-3} = 0.36$
wrong	**off-by**	40	$3.28 \cdot 10^{-3} / 5.14 \cdot 10^{-3} = 0.64$

Table 4.2: Normalized joint distribution $P(\mathbf{C_{6\times7}}, \mathbf{LS_{6\times7}}, \mathbf{A_{6\times7}} = 40)$

The process eventually performs the following computation in equation 4.5, which is in accordance to equation 4.1.

$$P(\mathbf{C_{6\times7}}, \mathbf{LS_{6\times7}} | \mathbf{A_{6\times7}} = 40) = \frac{P(\mathbf{C_{6\times7}}, \mathbf{LS_{6\times7}}, \mathbf{A_{6\times7}} = 40))}{P(\mathbf{A_{6\times7}} = 40)} \tag{4.5}$$

The distributions $\mathbf{Correctness_{6\times7}}$ and $\mathbf{Learning\ State_{6\times7}}$ in the learning competence model is as follows:

wrong	**correct**
0.158	0.842

operand	**intrusion**	**consistency**	**off-by**	**add/sub/div**	**pattern**	**unclassified**
0.336	0.079	0.163	0.103	0.0014	0.072	0.243

Table 4.3: $\mathbf{Learning\ State_{6\times7}}$ distribution of wrong answers in question 6×7 before the user answers 40

After observing 40, the Explanations probability distributions are as follows:

wrong
1.0

operand	**off-by**
0.36	0.64

Table 4.4: Explanations distribution of wrong answers in question 6×7 after the user answers 40

The result of the MAP Query (most probable explanation) is the joint assignment MAP($\mathbf{Correctness_{6\times7}}, \mathbf{Learning\ State_{6\times7}}$) = (wrong, off-by). The result of the Marginal MAP query over the $\mathbf{Learning\ State_{6\times7}}$ only, states that the most probable cause of the answer is the off-by error, as seen in figure 4.2.

This is an example of a case where the an error type has a higher probability than another one in the $P(\mathbf{Learning\ State_q} | \mathbf{Correctness_q} = \text{wrong})$ distribution,

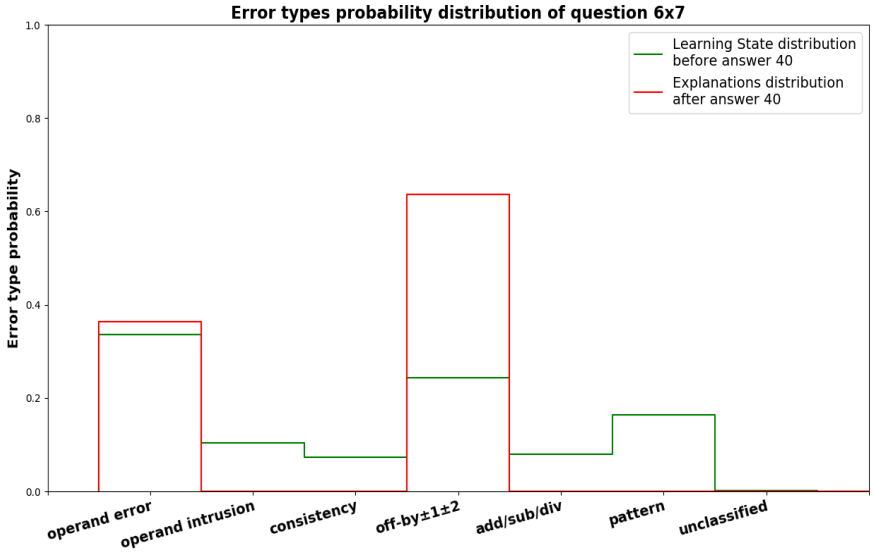


Figure 4.2: **Learning State**$_{\mathbf{6\times7}}$ and explanations distribution of question 6×7 before and after the user answers 40

but the probability query could state that the most probable cause of a particular answer is the second one.

The results of the probability queries depend on the parameters of the model, which in turn are influenced by the prior distribution and the number of EM-iterations.

4.3 Sampling of Answer

After the model parameters are learned, they can be used to predict the answer of the student to a posed question. The predicted answers can be used to validate the adequacy of the learned parameters. Two of them are implemented in the software library that was used for the implementation of this thesis: Forward sampling and Gibbs sampling (Pfeffer (2016); Koller and Friedman (2009)).

Forward sampling handles the random variables of the model in a topological order; in the learning competence model's case it is straightforward **Correctness**$_{\mathbf{q}}$ $\rightarrow$ **Learning State**$_{\mathbf{q}}$ $\rightarrow$ **Answers**$_{\mathbf{q}}$. Firstly, it samples a value from **Correctness**$_{\mathbf{q}}$ and then uses the sampled answer c to choose the next probability distribution $P(\mathbf{Learning\ State_q}|\mathbf{Correctness_q} = c)$ (which is a row in the Conditional Probability Table of **Learning State**$_{\mathbf{q}}$) to sample from, and so on. The set of the three

sampled values consist one sample. Figure 4.3 shows all possible outcomes of all random variables in the learning competence model in a tree form. The chosen sampling path is denoted with black arrows and all non-selected possibilities are coloured gray. The resulting sample is the set (wrong, off-by, 39).

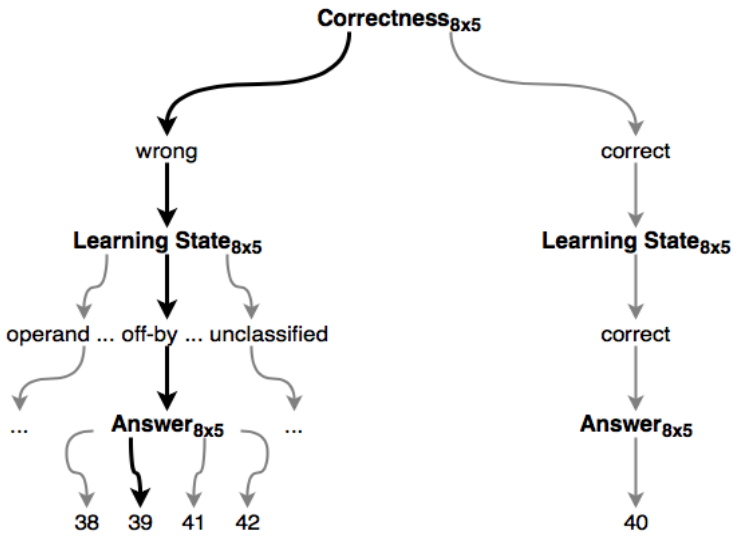


Figure 4.3: Tree of all possible paths in the forward sampling process of the learning competence model of 8×5. The sampling path for the sample (wrong, off-by, 39) is highlighted.

Gibbs sampling does not necessarily follow the topological ordering from the parent random variables to the children. It starts by generating the initial sample with the forward sampling process, but then it can proceed by sampling the random variables in any order, constrained by the values of the previous and current sample. Assuming that the initial sample is is (wrong, off-by, 39)$^{(0)}$ and the next variable that will be sampled is $\mathbf{Answers_{8\times5}}$, the value will be generated from the distribution $P(\mathbf{Answers_{8\times5}}|\mathbf{Learning\ State_{8\times5}} = \text{off-by}, \mathbf{Correctness_{8\times5}} = \text{wrong})$. The result of this sampling (for example 41), will be used in turn to constrain the sampling of the next chosen variable $\mathbf{Learning\ State_{8\times5}}$; the sample comes from the $P(\mathbf{Learning\ State_{8\times5}}|\mathbf{Answers_{8\times5}} = 41, \mathbf{Correctness_{8\times5}})$ distribution and so on.

Gibbs sampling differs from forward sampling mostly in the fact that the sampling process uses information about child random variables when sampling parent random variables. In cases where sampling is used for estimating the posterior, it is stated that with Gibbs sampling the approximated posterior will be closer to the real posterior because of this characteristic (Koller and Friedman (2009)). The initial state was randomly chosen. Unfortunately, at the time the Gibbs sampling

implementation of the used software library has a bug [2] so a comparison of the results of forward and Gibbs sampling is not possible.

To measure the prediction performance of the learning competence model, the metric that was used is similar to the Rank-N error metric (Lapin et al. (2016, 2015)). For every question in the training dataset, N samples were generated. A counter is increased, if the actually provided answer is not the list of sampled answers and the error can be defined as the following fraction 4.6:

$$\text{Error}_{\text{Top}_\text{N}} = \frac{\text{\# Number of actual answers not in sampled list of length N}}{\text{\# Number of answers in dataset}} \tag{4.6}$$

The value of the metric for the uniform dataset is 0.062. Since the dataset is unbalanced, in case where only faulty answered question are considered, setting the value of $\mathbf{Correctness_q}$ = wrong and performing forward sampling gives a Top_N error of 0.7016.

Other sampling algorithms are used in cases where evidence is present, mainly for estimating the posterior distribution after the evidence is observed. These include forward, rejection and importance sampling as well as other MCMC sampling methods apart from Gibbs sampling. Those are out of scope for the needs of this thesis; Pfeffer (2016) as well as Koller and Friedman (2009) provide detailed descriptions.

4.4 Generative Model

Another way that was used to verify the learned parameters is to use the probabilistic model as a generative model (Pfeffer (2016); MacKay (2003)). As explained in the previous section 4.3, there are processes that can generate samples from the learning competence models. These samples are used to train another model, having the same structure as the original one, with expectation-maximization as described in the previous chapter 3.6.

The values of the parameters in the corresponding CPD tables should be similar for the each two models; one being created by the training dataset and one from the

[2] https://github.com/pgmpy/pgmpy/issues/1017, Last accessed 17 March 2019

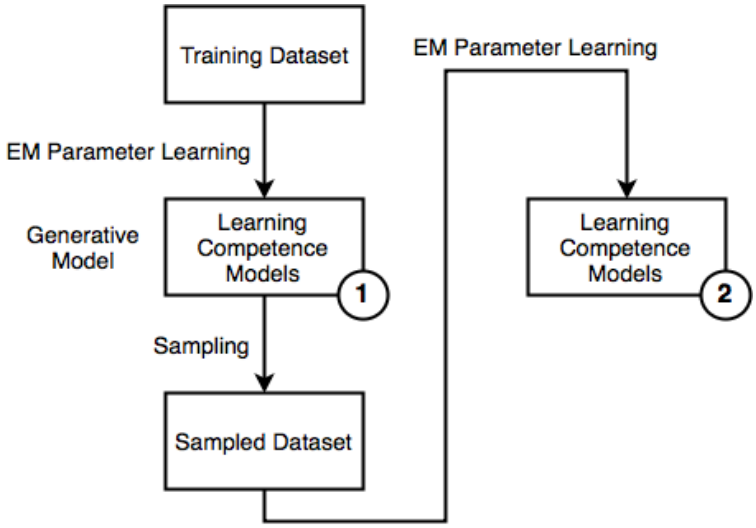


Figure 4.4: Generative Model

sampled dataset. The corresponding joint probability distributions were computed and compared with the Bhattacharyya coefficient (Bhattacharyya (1943)), which is a measure of dissimilarity between two distributions 4.7. For two discrete probability distributions p and q that have the same domain X it is defined as follows:

$$BC(p, q) = \sum_{x \in X} \sqrt{p(x)q(x)} \tag{4.7}$$

The distribution $P(\textbf{Learning State}_\textbf{q}|\textbf{Correctness}_\textbf{q} = \text{wrong})$ of all questions have the same 7 possible outcomes and can be represented as 7-dimensional vectors. The Bhattacharyya coefficient measures the angle between the $(\sqrt{p(1)}, \cdots \sqrt{p(7)})$ and $(\sqrt{q(1)}, \cdots \sqrt{q(7)})$ vectors; this geometric interpretation is in accordance to the property $0 \leq BC \leq 1$, which is derived by the Jensen's inequality (Cover and Thomas (2012)). Values close to 1 express similarity and values close to 0 express dissimilarity.

Other measures that can be used to measure the similarity or correspondingly the dissimilarity of distributions are the Hellinger discrimination (also called Matusita measure), chi-square (χ^2) measure (Derpanis (2008)). The properties, relationships between them, as well as their comparison and benefits of the Bhattacharyya coefficient are thoroughly elaborated in Aherne et al. (1998).

4.5 Similarity of Learning State among different questions

The design of the models structure assumes that the $\mathbf{Correctness_q}$, the $\mathbf{Learning\ State_q}$ as well as the $\mathbf{Answers_q}$ have parameters that are different enough for each question q to be handled by a different graphical model each. After training the set of models as described in the previous chapter 3.6.3, chapter 3.8, the $P(\mathbf{Learning\ State_q}|\mathbf{Correctness_q} = \text{wrong})$ distributions was used for comparing different questions.

The pairwise similarity over all question pairs was computed and put into sorted order. As explained in the previous section 4.4, the nearest distributions have Bhattacharyya Coefficient near 1 and the most dissimilar near 0. The minimum coefficient value is 0.452, which implies the minimum similarity or max dissimilarity within the distributions (10×6 with 3×3). In general, the most similar questions are the ones with swapped operands, as for example 6×8 with 8×6 have a similarity of 0.998.

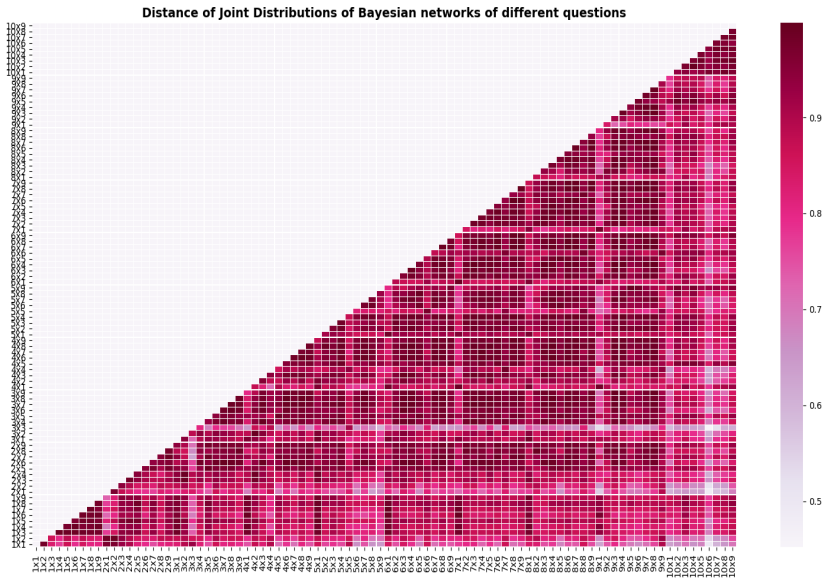


Figure 4.5: Similarity of the $P(\mathbf{Learning\ State_q}|\mathbf{Correctness_q} = \text{wrong})$ distributions of all pairs of questions as measured with the of the Bhattacharyya Coefficient 4.7.

4.6 Other uses of the Learning Competence Model

The learned probabilistic model can be used in a generative scheme where the learning application will sample the model to predict the answer of the student. There are several algorithms that compute samples from the models with different characteristics (Pfeffer (2016); Koller and Friedman (2009)). For this model, where the dataset is highly unbalanced and the number of correctly answered questions is predominant, the metric to measure prediction performance should particularly take this fact into account. Although this feature does not provide an insight per se, it can be a starting point for other informative learning aspects. One aspect is explainable artificial intelligence (explainable AI), which combines Bayesian learning approaches with classic logical approaches and ontologies, thereby making use of re-traceability and transparency (Goebel et al. (2018)).

Even though the proposed research extends the capabilities of the current learning application considerably, it cannot answer the fundamental question: Which should be the most appropriate question to be posed to the student. After the different learning competences are derived, the handling is delegated to the teacher, not the application itself. Further considerations apply to whether the models of learning competences that could be grouped together are the ones where the students will have the same learning path till they've learned to answer all questions correctly. The goal of this learning-aware application is not to group the learning competences by similarity of their parameters (expressing the current situation), but to find those that will lead to similar optimal learning paths. This learning-aware application could benefit from an answer prediction component that accurately simulates students learning paths.

5. Deep Reinforcement Learning and Convolutional Neural Networks

5.1 Deep Reinforcement Learning and Convolutional Neural Networks

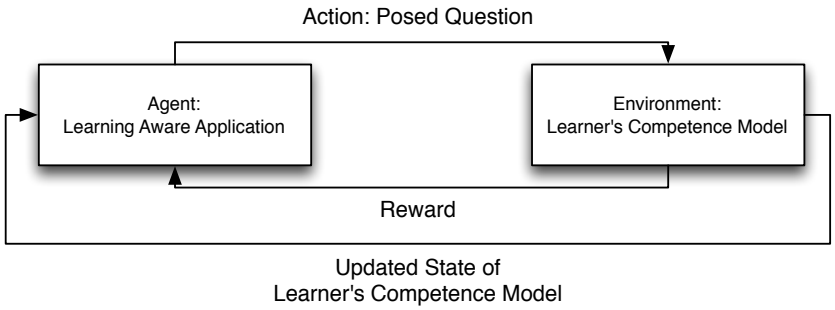


Figure 5.1: Agent, environment components and interaction of the learning-aware application

To impact the learning of each user in an efficient manner, we first formulated the problem using concepts of reinforcement learning. In a reinforcement learning setup, as seen in figure 5.1, there are two main interacting counterparts: the agent and the environment. The environment's content can be described by its state, which can be (at least partially) observed by the agent. On the other hand, the agent performs some actions resulting some influence on the state of the environment. This change of state is communicated back to the agent together with a value which is called "reward" or "reinforcement" that designates whether this state change was beneficial or not.

The usage of a learning application involves at least the user/learner and the application itself. The learning status of the user is the environment and the learning

application is the agent, which is in constant search for reasonable and effective actions that hopefully enhance the learning competence of the user. The learning application is not composed by any pre-defined programming logic that computes what to do next according to rules. In other words, it does not need to implement a methodology; the requirements only need the description of the problem. With the use of reinforcement learning and the interactions rewards, the learning application will learn itself the best behaviour to bring the greatest benefit to the learner.

The reward can be a "positive" or "negative" feedback with respect to the goal of the agent, which is to choose the actions that will return the greatest cumulative reward in the long run. The reinforcement learning system can be designed in such terms, that this occurs at the moment the environment reaches a desired state. The learning process has a "trial-and-error" character, because it tries to improve its estimation of the optimal state and action sequence by exploration, which allows the choice of local suboptimal actions. This characteristic differentiates reinforcement learning from both supervised and unsupervised learning; the process of learning is neither supervised nor consists of finding patterns in the data.

Setting a numerical reward for a specific state change (that was triggered by a particular action) needs careful considerations. Any choice of action influences the upcoming states; it excludes some of them. It is crucial to understand that any action at any given moment cannot be characterized only by its current reward, but also by the rewards that will be returned from the future states, which are reachable from the chosen one. Since reinforcement learning algorithms are not short-sighted, it is preferable to endure a temporary negative reward if it eventually brings a long-term benefit.

5.2 Applications of Reinforcement Learning

Reinforcement learning is already applied in different industries. It is used in autonomous driving (Bojarski et al. (2016); Sallab et al. (2017); Zhu et al. (2018)) and robotics (Kober et al. (2013); Amarjyoti (2017)) to carry out tasks without being given explicit instructions. Choosing the right advertisement being presented to the users in real-time is also an emerging business-case to prevent advertising fatigue (Zhao et al. (2018)). Because users are confronted daily with many ads, the presen-

tation of relevant ads at the right place and moment is an important requirement. Personalization of the content as well as the sending time of advertising e-mails and push notifications can also be improved by reinforcement learning methods. Software development companies use A/B testing, which is a simplified and more short-term form of measuring the user's acceptance of a new feature as soon as it has been deployed. The financial sector is also using similar methods to increase profits by consulting reinforcement learning techniques to reduce risks.

One of the most prominent applications of reinforcement learning that uses the techniques described in this thesis is, playing of computer games, such as Backgammon, Atari, and Go, successfully (Mnih et al. (2013); Pumperla and Ferguson (2018); Silver et al. (2017)). Scientists have shown, that by applying reinforcement learning techniques, computers can learn, how to choose good actions in order to play such games even better than humans. What is important to note is, that the program is not rule-based; the computer learns a good policy by playing several times (corresponding to episodes), with different game and score evolution, often exploring actions that may have negative outcomes. At the end of this training phase, the algorithm has learned a strategy that will be used in the following trials of the game where the learned policy is exploited. At any given state of the game, the algorithm will know, which action to choose. The given input is solely the pixel image of each game state (comparable to the image a human player would perceive) and the current score.

5.3 Formulation as a Markov Decision Problem

The state at each discrete time point t is represented by s_t. The whole probabilistic graphical model of each user represents the belief of the agent over his or her overall current learning status. The number of possible parametrizations of this model can be infinite; each state is only subject to the constraint that the sum of each line of a conditional probability table is one. The whole set is denoted by $\mathcal{S}$, where all $s_t \in \mathcal{S}$, or by $\mathcal{S}^+$ if one denotes the inclusion of the terminal state (state in which the goal is achieved) in the set. The actions are the posed questions a_t at a specific time point. There is a set of $|\mathcal{A}|$ questions at each time step for every state $\mathcal{A}(s_t)$. Each action results to an immediate numerical reward $r_{t+1} \in \mathbb{R}$ for the agent. The definition

and computation of each reward is done by the environment and has the form of a reward function that typically has the previous state as its input, the applied action, and the immediately next state. A schematic description of a small Markov decision problem with three states, two actions, and two rewards is depicted in figure 5.2.

The Markov property, expressed in equation 5.1 states that the probability of the transition from a particular state to another one through an action is not dependent on past states and actions.

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \cdots) \tag{5.1}$$

The state update can be deterministic or not; one example can be drawn from robotics: the movement of a robot might not be precise and lead on to different positions although the starting state and movement command can be the same.

The goal specification influences the definition of the reward function (also called credit assignment problem). In our application the following considerations have helped to choose the reward function: A question may enrich the knowledge of a learner and furthermore, may help understanding as well as overcoming typical errors of a particular user. The long-term goal of the application use is the correct answering of all questions, as efficient as possible; this is essential for the design of the reinforcement function. A wrongly answered question may contribute more to the mathematical understanding than a correct answered one and, in that way, can lead faster to the final state although the agent does not receive a greater reward. This difference between short- and long-term benefit emphasizes the need for the value function, which expresses the expectation an agent has about the accumulated reward, it will begin to gather beginning from a specific state on. For the agent, the value function is the one, its decisions are based on; whereas the reward function helps to compute the value function. A state with a greater value than another one is classified as being "better"; the agent must coordinate its actions towards reaching it instead of the other.

The purpose of the learning-aware application using reinforcement learning is to find/learn the optimal action/question for each learning competence of the user. This is called "optimal policy". There are many possible policies; each one denoted by π spanning the policy space. In general, there can also be many optimal policies

π^* with equal accumulated reward. The learning-aware application as an agent will know the current overall learning competence of the user. Therefore, it will be able to derive the best question to be posed. The result will be the optimal sequence of the posed actions. Optimal in our case means, that the learner will reach the goal of answering all questions correctly faster using the proposed sequence than by any other question sequence.

To achieve this, the agent must learn and choose the policy that maximizes the sum of discounted returns of the states (see equation 5.2) that the environment will find itself. Assuming the number of posed questions is finite T, γ is the discount rate where $0 \leq \gamma \leq 1$, the expected discounted return is computed as follows:

$$G_t = \sum_{k=0}^{T-1} \gamma^k r_{t+k+1} = r_{t+1} + \gamma G_{t+1} \tag{5.2}$$

The discount factor is necessary for bounding the expected return in reinforcement learning problems, where the interactions between agent and environment have no end, meaning, they don't stop even after reaching the terminal state. In the finite case the return cannot grow indefinitely, but the discounting factor is still useful to control the relative impact of future rewards with regard to immediate ones. As the time steps k grow, the discounting factor's powers get smaller, making the future rewards less significant than the present ones. When γ is almost equal to zero, immediate rewards are more important than future ones; this can prevent a really high return, since it may exclude future higher rewards. On the contrary, a γ near one makes the agent looking exactly for those greater rewards.

Our learning application falls into the category of episodic tasks, where an episode follows a finite sequence of alternating actions/questions and changes overall learning competence of user until the final state in a finite number of steps is achieved. For every state other than the final one, the reward is set to -1. The final state can have a reward ≥ 0, so that the less steps reaching the final state needed, the greater is the overall reward (so-called "Minimum Time to Goal" in Harmon and Harmon (1997)).

To find the optimal policy, the agent may go through several policies, typically by adjusting the probability of choosing each action given a particular state according to its experience. There are several ways to find the optimal policy or an approximately

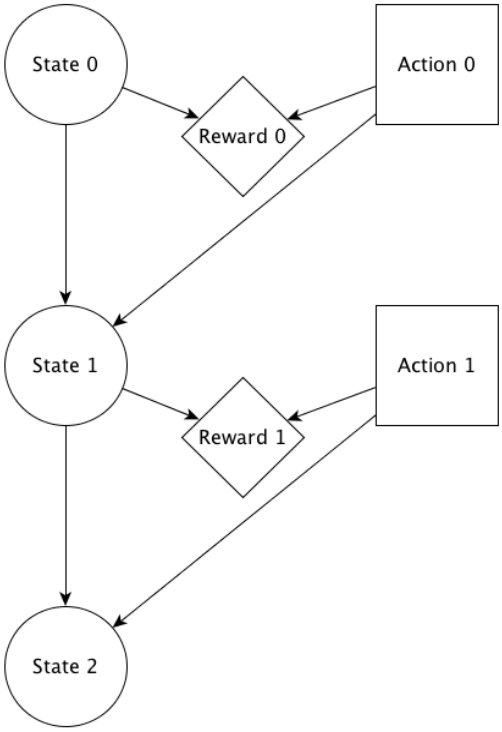


Figure 5.2: Markov Decision Process where State is the learning competence, Actions are the posed Questions and the Rewards need to be computed.

optimal policy systematically through different algorithms.

The algorithm does not need any past sequence information to compute the reward; the current state and action that will lead to the next state are enough. In fact, because of the Markov property, the agent has the opportunity to find the optimal policy only by looking at the present state S_t and choosing the best actions, ignoring every past situation. The algorithm can begin from any state and predict the probabilities of the following states, try various actions, compute their rewards, and compare them to find the optimal policy. Of course, the Markov property is satisfied when the environment is the probabilistic graphical model; if one considers the "true" mentally learning competence of the user as the "real" environment, then the state is only approximately a Markov state, because there is no possibility at the moment to have full access to the "true" state. Even so, this is a typical case and it is still appropriate for the scope of this work.

The satisfaction of the Markov property by each state classifies our reinforcement learning task as a Markov decision process (short MDP), whose dynamics are

largely specified by two components: First, the transition probability expressed by $P(s_{t+1}|s_t, a_t)$ and second, the element of an Markov decision process is the expected value of the next reward $\mathbb{E}[r_{t+1}|s_t = s, a_t = a, s_{t+1} = s']$, which is defined for every possible state and influenced by the followed policy. Its value depends on the states that will be faced after the current one and the expected rewards thereof. But as explained previously, the expected immediate reward is not of primary importance; the mathematical expression for all expected accumulated future rewards from a particular state s or an action choice a is the value function and denoted by $\upsilon_\pi(s)$ and $q_\pi(s, a)$ respectively. The state value function of a state under a particular policy π is:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{T-1} \gamma^k r_{t+k+1}|S_t = s\right] \tag{5.3}$$

The state-action value function under a particular policy π evaluated at possible each state-action pair as well as enabling the comparison of actions:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{T-1} \gamma^k r_{t+k+1}|S_t = s, A_t = a\right] \tag{5.4}$$

The constraint for the discount rate parameter is $0 < \gamma < 1$. This parameter is not just necessary for expressing the discounting of future rewards, but ensures that the expected accumulated future rewards in problems with infinite steps are finite, provided that the reward's sequence acquired by following the states defined by the policy are bounded. The expectation is used, because of the fact that the transitions are non-deterministic. A small discount factor (near 0) can make the actor have a behaviour that ignores long-term rewards for immediate ones, whereas the opposite happens in case the discount factor is close to 1.

There are several extensions of MDPs for problems that cannot be expressed by the means of MDP. The partially-observable Markov-Decision Process (POMDP) deals with cases, where dynamics of the environment are not observable as it is in this application. Although the Probabilistical Graphical Models do have hidden states making the problem expressible in a POMDP manner, it was decided to use the estimated parameters as information that represents the environment and fulfils the properties of an MDP. Others are multi-agent MDPs, where many agents interact simultaneously with the environment.

5.4 Computing the optimal policy

Defining the optimal policy is in direct connection with the computation of the value function; once the value of each combination of state and action is computed, then the trajectory is chosen on the basis of the expected value. A suboptimal policy may contain optimally decided steps, but an optimal policy contains only optimal actions on encountered states.

By substituting the value function of the next state s' in 5.3 the Bellman equation is derived:

$$V_\pi(s) = \sum_{a \in \mathcal{A}(s_t)} P(a|s) \Bigg(\sum_{s'} P(s'|s,a) \Big(r(s,a,s') + \gamma V_\pi(s') \Big) \Bigg) \tag{5.5}$$

By examining the equation from the inner to the outer parenthesis one can see, that the reward acquired from state s to state s' through the action a is added to the discounted value function of s'. This value is averaged by the probability of the transition to s' and the average over all reachable states is computed. In the last step all possible actions that are possible from state s are summed over. The value of the next state is also defined in a similar substitutional way; this provides several possibilities for exact solving of these equations for all possible states or approximations thereof.

The optimal policy provides for every state or state-action pair the optimal state-value function and optimal action-value function respectively. This is formulated by taking the maximum of the equations 5.3, 5.4 respectively and by using the Bellman equation 5.5:

$$V_*(s) = \max_\pi \upsilon_\pi(s) = \max_{a \in \mathcal{A}(s_t)} \Bigg(\sum_{s'} P(s'|s,a) \Big(r(s,a,s') + \gamma V_*(s') \Big) \Bigg) \tag{5.6}$$

$$Q_*(s,a) = \max_\pi q_\pi(s,a) = \sum_{s'} P(s'|s,a) \Big(r(s,a,s') + \gamma \max_{a'} Q_*(s',a') \Big) \tag{5.7}$$

This set of linear equations is straightforwardly solvable, because there is one lin-

ear equation with one unknown per state. The expected returns from a specific state (onwards) is a function that needs the value of the following states as input. Those are not yet computed; which means that the value function is defined recursively.

Reinforcement learning can be implemented by different algorithms (Sutton and Barto (2018)). Value iteration computes the value function for each state, whereas policy iteration goes through all policies, evaluates them according to their value and progresses to discover the optimal one. There are many approximation algorithms that can be incorporated for finding good policies, using for example Monte Carlo methods. Explaining all those methods is out of the scope of this thesis, although the selected algorithm and the reasons for its selection, are thoroughly covered.

5.5 Tabular Q-Learning

The Q-Learning algorithm is an online value based method that iteratively approximates the optimal state-action value by letting the agent making several tries from the start state to the terminal states and exploring different paths between those states. By monitoring the rewards gathered in each state of each path, the agent updates at each transition of each episode the Q-value of the state-action pair, which is currently exhibited. The rewards are fixed and stored in a rewards matrix that has row count equal to the number of states $\mathcal{S}^+$ and column count equal to the number of all possible actions. A second Q-matrix with the same dimensions is initialized with zeros (although even random initial values converge to the true Q-values) and updated for each reached state as well as chosen action from that state by means described in the following equation, which is directly derived from equation 5.7:

$$Q(s,a) = r(s,a,s') + \gamma \max_{a \in \mathcal{A}(s)} Q(s',a) \tag{5.8}$$

After a number of episodes, the Q-values matrix (short: Q-table) will converge to the true Q-values of every pair of state and action. Q-learning has two phases: The first phase where the Q-table is filled by training over many episodes and the second one where the table is fixed and decisions are made to choose the best known path from the start state to terminal state. Each state in each row can follow the best next action by choosing the one with higher Q-value.

5.6 Methods for continuous state space

The computation of 5.7 is practical for problems with a small state space and number of possible actions; it is infeasible for MDPs with a high number of states and actions, as well as unsuitable for continuous state and/or action spaces.

Since the state is represented by the Bayesian network, it is a continuous variable, meaning that the conditional probability tables contain floating point numbers. There is an infinite number of states, therefore it is not possible to take every possible state into account or revisit states seen in the past (using a lookup table for the computation of the value function is no longer possible/feasible). Although there are similarities in the learning processes of students, this fact cannot be used, because the model of their overall learning competence will never be exactly the same.

This situation is similar to the one in games; there are so many possible states, that the problem of finding the optimal policy is intractable. Nevertheless, finding a good policy is possible even when the algorithm confronts a strong opponent, because the states that occur in practice are much smaller than the whole state space. Therefore the reinforcement learning algorithm should produce good results for states that are highly likely to occur during execution, even if the performance of unlikely states becomes worse because of that.

This principle applies to our application too, since from our investigations in probabilistic graphical models for students we know, that the vast majority of students answer the posed questions correctly with a probability higher than 90%. This, in combination with the fact that the $\mathbf{Correctnes_q}$, $\mathbf{LearningState_q}$, and $\mathbf{Answers_q}$ random variables must sum to 1, reduces even more the number of states our reinforcement learning algorithm will face during training as well as the execution phase.

There are several methods that can deal with continuous state spaces such as dynamic programming for discretized state and action space (Sutton and Barto (2018); Busoniu et al. (2010); Lazaric et al. (2008)). The recent approaches use deep neural networks (called Deep Q-networks) that implement an approximation of the model-free Q-Learning algorithm. Since approximation methods try to find a non-linear mapping between the input states and the Q-Value of all possible actions,

the idea of using a deep neural network to learn this mapping is beneficial.

5.7 Artificial Neural Networks

The construction of artificial neural networks orients by the one of biological neurons. Thus, they have a simplified architecture, components, and functionality of their biological equivalent. The elementary component is the artificial neuron with its bias part that computes the sum of a weighted combination of the input $\bar{x}$ and passes it through a non-linear function 5.9:

$$y = f(w_1 x_1 + \cdots w_n x_n + b) = f(\bar{x}\,\bar{w} + b) \tag{5.9}$$

Figure 5.3 depicts an artificial neuron. It is out of scope of this thesis to describe the structure and functionalities of a biological neuron; several introductory resources to artificial intelligence (Aggarwal (2018); Patterson and Gibson (2017)) dedicate a substantial amount of material to describe the principles on which well-established artificial neural network base their design.

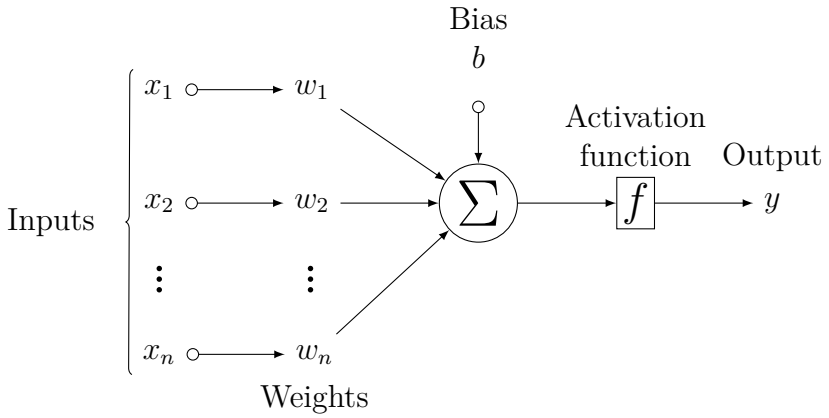


Figure 5.3: Architecture of an artificial neuron and its principal components

The non-linear function f is called activation function and is chosen according to the problem. It is necessary, because linear functions are not capable to deal with non-linear problems. The most used functions are the sigmoid, hyperbolic tangent (tanh), and the Rectifier Linear Unit (ReLU) 5.10, which counteract the vanishing gradient problem (Aggarwal (2018)): If the input to the sigmoid or hyperbolic tangent function is too small or too large, then the slope is small and the error

optimization method 5.7 is not proceeding optimally. Because of those reasons and because of the fact that it is computationally faster, the ReLU and its variations (like the Leaky ReLU) are an established choice of an activation function. Figure 5.4 depicts three of the most common activation functions.

$$f(x) = x^+ = max(0, x) \tag{5.10}$$

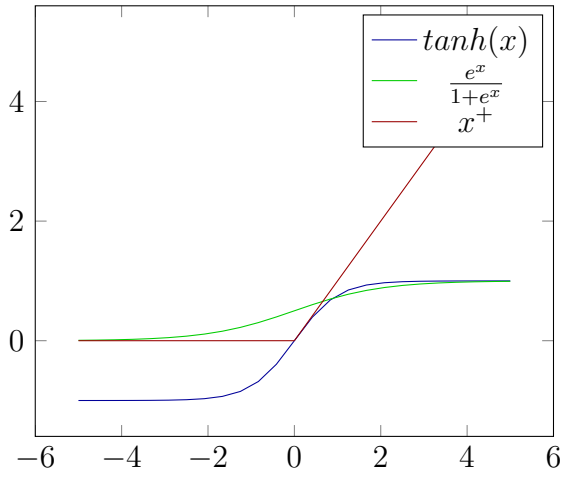


Figure 5.4: Non-linear activation functions

A neural network is composed by many neurons organized in different architectures. The simplest one groups many neurons together and organizes them in layers so that the outputs of one layer can be the inputs of the following layers as depicted in figure 5.5; which is called multi-layer perceptron architecture. The number of layers as well as the number of neurons composing each layer are configurable; the architectural decisions are influenced by the problem. The first layer, which has as input the data, is called input layer. The last layer is the output layer and all layers in-between are the hidden layers.

The weights of the neural network are learned according to the task. Primarily, neural networks are used for supervised learning where the desired output values of the network are known. Classification of the input data as well as regression problems have a pre-defined target $\hat{y}$, which the network tries to approximate during the training phase. The error of the neural network's output and the target is $E(\bar{x}) = y - \hat{y}$. This value guides the learning of the weights $\bar{w}$ over many input samples $\bar{x}$ comprising the training dataset; the minimization of the loss function $L = (y - \hat{y})^2$ is

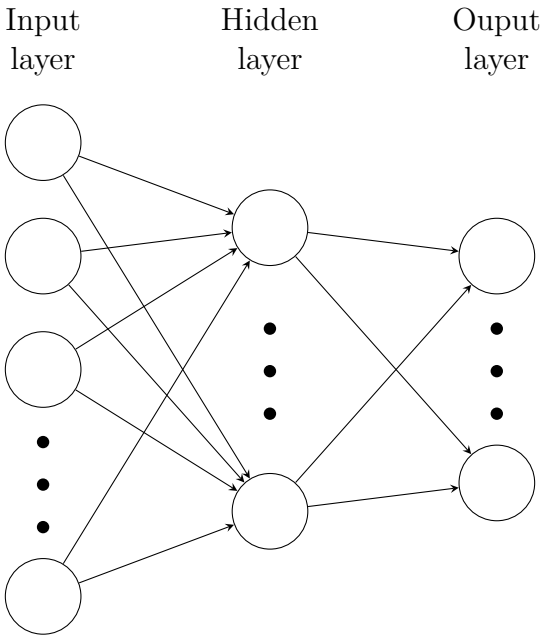


Figure 5.5: Dense neural network architecture, also called multi-layer perceptron architecture or simply feedforward dense network

made with backpropagation of the gradient of L with respect to the weights $\bar{w}$. The learning process of a neural network uses alternating feedforward phases (where the input "passes through" the network to generate y) and the backpropagation phase (where the error is used to adapt the weights). Each pair of those phases is called an epoch and can use the whole training set (batch) or a randomly selected set of it (stochastic).

As depicted in figure 5.5, all elements of each layer are used to compute the elements of the next layer. Therefore, this architecture is also referred to as dense or fully-connected. The output of each layer consists a feature that is transformed from layer to layer. The weights are found by the gradient descent optimization algorithm . Its implementation depends on both the neural network architecture as well as the error function itself. For regression problems, the typical error function is the mean squared error (MSE), whereas for classification it is the cross entropy. The general idea is that starting with random weights, the optimization method should change them in a way that the error is reduced. This happens by computing the slope for all directions at a particular weight in the weight space. After choosing the steepest one, the new weight will be on this direction, in a distance defined by the step size or learning rate α of the algorithm. The weight update equation is :

$$w_{new} = w_{old} - \alpha \frac{\partial L}{\partial \bar{w}} \tag{5.11}$$

From all the descriptions above, it is apparent that there is a set of parameters, which substantially influence the weight learning process. They need to be tuned along with the learning of the weights. These parameters are called hyperparameters. There are directives for their reasonable setting as well as necessary trial-and-error procedures. Measures against overfitting, such as a random dropout of a proportion of neurons at each epoch (Srivastava et al. (2014)), as well as regularization of the weights, and early stopping of the training phase (Bishop (2006)), complete the basic configuration spectrum of neural networks.

5.8 Deep Q-Network

A neural network can be used for function approximation of the optimal Q-function (Trask (2017)). The input is a state representation, which for a computer game is the image, and in our case, is the parameters of the probabilistic graphical model. The number of outputs equals the number of possible actions, whereas their value is the current prediction of the Q-value of that particular action. During training, the output of the neural network is compared to a Q-value target that corresponds to the Bellman equation. The mean squared error is back-propagated to adapt the weights of the neural network and correct the prediction. In the end of the training phase, the index of the network's output node with the maximum value is used to pick the proposed action.

The input of the neural network can be raw data representing the state s_t and/or features thereof (Aggarwal (2018)), which can be overall denoted by x_t. The number of outputs of the network are $|A|$ the approximated values $Q(s_t, a)$, one for each possible action a. The neural network computes the function $Q(s, a; \theta)$ where θ are all the parameters of the network (weights denoted by $\bar{w}$ in equation 5.10). Since the number of possible states is infinite, the neural network is expected to be able to recognize those states that are similar to known ones with an already computed good policy. The effectiveness relies in two assumptions: First, the number of actually encountered states is less than the number of possible states and second, similar

states will need similar good policies. The neural network enables to generalize from encountered to unseen input states.

In this case, the deep neural network represents the information of the learning competence of the student, which is necessary to decide the next best question to be posed. It is not making the usual classification (supervised) or clustering (unsupervised) of the input; instead, it learns a low-dimensional representation of it to decide the optimal policy. The function $f(\bar{x}_t, w, a)$ that is computed by the neural network w.r.t. its learned parameters $\bar{w}$ and the input $\bar{x}_t$, approximates the true value $Q(s, a; \theta)$.

$$f(\bar{x}_t, w, a) = Q^*(s, a) \approx Q(s, a; \theta) \tag{5.12}$$

Like the Q-Learning algorithm, the network does not have the value of the next state; instead of performing the value evaluation and improvement phases, the Q-value of the next state is estimated by using equation 5.8 with one-step lookahead. The equation 5.12 is transformed to the following:

$$f(x_t, w, a) = r_t + \gamma \max_{a \in \mathcal{A}(s)} f(x_{t+1}, w, a) \tag{5.13}$$

The value of $f(x_{t+1}, w, a)$ is taken from the output of the network and consists the target (equivalent to $\hat{y}$). The loss function is expressed by 5.14:

$$L_t = \Big((r_t + \gamma \max_{a \in \mathcal{A}(s)} f(x_{t+1}, w, a)) - f(x_t, w, a_t)\Big)^2 \tag{5.14}$$

The backpropagation algorithm implements the following weight update rule in the network:

$$\bar{w} \leftarrow \bar{w} + \alpha\Big((r_t + \gamma \max_{a \in \mathcal{A}(s)} f(x_{t+1}, w, a)) - f(x_t, w, a_t)\Big)\frac{\partial f(x_t, w, a_t)}{\partial \bar{w}} \tag{5.15}$$

The Q-function is approximated by the neural network and can theoretically take infinite time to converge. In contrast to the tabular Q-Learning, it does not necessarily converge to the true action value function.

Variations of deep reinforcement learning include an experience replay memory, which is a separate memory of training data randomly sampled to create a data

batch (Mnih et al. (2015); Schaul et al. (2015)), and the usage of two networks (Wang et al. (2015); Van Hasselt et al. (2016)) as it is common in the actor-critic methods (Sutton et al. (2000); Arulkumaran et al. (2017); Mnih et al. (2016)). One of these variations, the asynchronous n-step Q-Learning algorithm, was chosen for the purposes of this thesis. Enhanced performance was lately achieved by the combination of several methods by the Rainbow method (Hessel et al. (2018)).

5.9 Asynchronous n-step Q-Learning

Deep Reinforcement Learning can be made even more efficient with the use of asynchronous methods, as described in Mnih et al. (2016). The main idea is that the deep Q-Network's gradient is asynchronously updated by several agents that explore different environments in parallel threads (respectively also called workers). The computations can use the CPU power, which is beneficial in this application since the transition to the next state requires an update of one of the probabilistic graphical models.

Each of the workers uses its own copy of the current state, but they all update the same deep neural network with asynchronous gradient descent as well as using its outputs for the choice of the next action. This algorithm has several benefits compared with its non asynchronous counterpart described in the previous section. It terminates faster and the updates of the neural network are not correlated since each agent will be in a different sequence of states. That means, that there is no need for experience replay memory 5.8, thereby reducing the memory footprint.

The pseudo-code of the 1-step Q-Learning is described in Mnih et al. (2016). What is similar to the Deep Q-Network of the previous section is, that there are two neural networks, but no replay memory in this case. The parameters of the networks (θ and θ^- correspondingly) are visible and changeable from all workers. They are initialized with the same values together and the maximum number of steps is defined T_{max}.

The main loop of the algorithm is very similar to the one used in Deep Q-Learning. For every current input state a forward pass of the first network is made. The outputs of the network have the approximation of the Q-value of each possible

action $Q(s, \alpha; \theta)$. These values are used to choose the next action by the ε-greedy policy. That means that with probability ε the chosen action will be randomly picked, or else the one with the maximum Q-value will be chosen. The action is a question posed to the student; the answer to this question produces the change in our belief of the learning competence of the user. Since the probabilistic graphical model captures the current knowledge of the learner, the answer to the question can be sampled from the model. The new state s' is computed after an expectation-maximization (EM) update and is used together with the reward for the computation of the target:

$$y = \begin{cases} r \\ r + \gamma \max_{\alpha'} Q(s', \alpha'; \theta^-) \end{cases} \tag{5.16}$$

The parameters of the network are updated as follows 5.17:

$$\mathrm{d}\theta \leftarrow \mathrm{d}\theta + \frac{\partial(y - Q(s, a; \theta))}{\partial\theta} \tag{5.17}$$

The values of ε are not annealed as in the Deep Q-Network case. The end value of ε of each thread is sampled uniformly from the set $0.01, 0.1, 0.5$, thereby each thread will have a different pace of exploitation-exploration, since they all start from the value 1.0. The targets must be clipped in the $[-1, +1]$ to avoid exploding gradient.

Apart from the γ parameter, there is the ε controlling the choice of the next action in the first phase. In the first episodes the agent must be allowed to explore enough states in order to make some estimation of the Q-value for them. The choice of the actions may be set to chance. Towards the end of the phase - in the last episodes - it may be more interesting to exploit stronger and refine the estimations of those states found out to be good ones. The proportion of random actions is controlled by the parameter $0.0 < \varepsilon < 1.0$; the exploitation actions are chosen with probability $1 - \varepsilon$. The whole strategy is called ε-greedy, where the ε's value is not fix but is annealed over the course of the episodes.

The learning rate or step size λ is a number between 0.0 and 1.0 that regulates the weight of the newly acquired information compared with the already stored one. This parameter was chosen according to by observing the performance of the

network. As the training proceeds, the random actions became less and less and particularly those threads with an ending ε value of 0.01 or 0.1 had a total number of steps near to two times the number of questions. Since the vast majority of the questions are answered correctly, the simulations with a good question sequence have a total number of steps close to a sequence with all questions being immediately answered correctly twice.

5.10 Convolutional neural networks

Machine learning models involve feature extraction as one of the crucial pre-processing steps (Zheng and Casari (2018)). The input to the model is not necessarily the raw data, but properties and mathematical transformations thereof, which are called features. The performance of the models can be increased by the choice of an informative feature; many standard software libraries like "scikit-learn" [1] provide the importance of each feature after training. Therefore, one of the data scientist's tasks is to invent input features that are semantically relevant for the modelled domain.

Convolutional neural networks (CNN) manage to discover the necessary features for the accomplishment of the required task themselves. Before them, the feature that was used to provide information about boundaries of objects in the image was the image gradient, which was computed by the subtraction of neighboring pixel values (Zheng and Casari (2018)). Gradients can be computed in the horizontal, vertical, or any other direction (from 0°to 360°degrees); their distribution over each angle can be provided as an input to a machine learning model that makes predictions based on those input features. Nevertheless, there were still design decisions that data scientists had to make as the angle resolution and the size of the neighbourhood. Furthermore, a feedforward neural network required unrolling the two-dimensional image to a flat vector which does not promote the recognition of the same shape in different positions. For example, if a rectangle is indicative for identifying a car in the input image, one cannot expect it to appear always in the exact same position and be mapped in to exactly the same neurons every time.

The convolution operation is usually applicable to two-dimensional inputs, but it is also used for one-dimensional or three-dimensional inputs. Image processing

[1] `https://scikit-learn.org/`, Last accessed 17 March 2019

applications typically use three RGB (red, green, blue) channels, so in this case the overall input is composed by three two-dimensional grids. In this respect, the input can be composed by several two-dimensional elements consisting the channels or depth of the input. Each of those elements will have different corresponding filter components applied to it. Part of the filter structure is also a bias element, as in dense neural networks, which is also learned during the training phase.

The discrete two-dimensional convolution is a linear transformation described by the following equation 5.18:

$$(f * g)[i, j] = \sum_{u=0}^{m} \sum_{v=0}^{n} f[u, v] g[i - u, j - v] \tag{5.18}$$

The f represents the input image and the g is a two-dimensional kernel (the operation is symmetric thereby f and g can be interchanged). The goal of the training is to learn the values of the kernel g; the process is the same as with feedforward networks, but the weights have no longer vector form but are two dimensional filters, which are called kernels. In image processing there is a vast palette of known kernels that, when convoluted with an input image, produce an output image with a desired effect such as blurring (Gaussian filter) or edge detection (Sobel filter). It is expected that the convolutional neural network will need to learn and use several different kernels to solve the requested machine learning task, but even if its architecture differ, the training process with gradient descent remains the same.

The kernel is multiplied with several parts (patches) of the image in a windowing scheme that starts from the upper left side of the image continues column-wise; once it reaches the last column of the image, it proceeds row-wise correspondingly in the next row. The filter is aligned with the image and the dot product of its elements with the image region is computed. The same procedure co-occurs to all input channels, each of them with their own filter. All finished dot product results of all channels at each position are aggregated to compute the value at the same position of the sole output per filter, which is called feature map. The semantical interpretation of this addition operation expresses the need for different learned patterns combined together for the recognition of entities relevant to the task.

The number of possible alignments between the image and the learned filter is configurable. The step of the sliding window is specified by the stride parameter; if

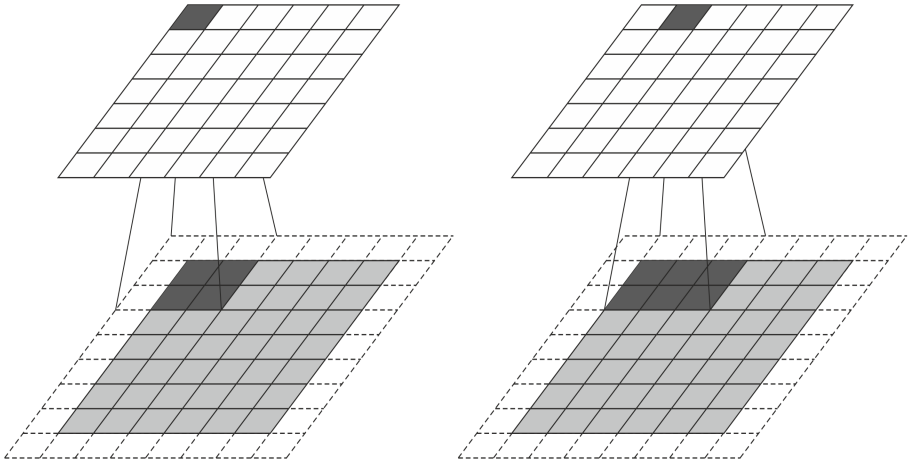


Figure 5.6: Input and output grid of a two dimensional convolution operation with a kernel of size 3×3, zero padding, and stride 1. The kernel itself is not shown in the figures. The left part depicts the first operation whereas the right one the second, as the kernel slides over parts of the input grid.

its set to 1, then the next alignment will be in the next pixel and there will be an overlap between consecutive patches. To involve the edge pixels of the input image as much as the middle ones in the computations, the image can be surrounded (padded) by zeros prior to the computation of the dot product. This can keep the size of the output image same to the size of the input image of each layer. Finally, pooling is another nonlinear operation, which can be averaging, summing or maximum, which can be applied on parts of the feature maps. Figure 5.7 shows the max pooling operation with the maximum value in a neighbourhood of values is selected; it also works in a windowed function as the convolution. The operation performs subsampling on its input, whereas zero-padding contributes in retaining the size of the feature maps. After that, the nonlinearity (ReLU in the newest architectures, because of its accuracy and speed improvements Aggarwal (2018)) is applied.

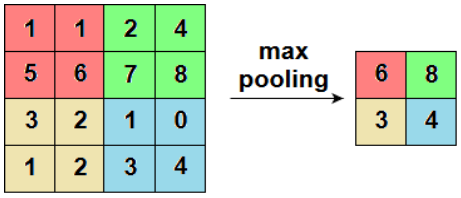


Figure 5.7: Max pooling operation: the input is split into four regions denoted by the different colors. The maximum value of each region is kept for the result of this operation.

The size of the feature map depends on those configurations, for example, if the stride is higher, then the size of the resulting feature map is smaller. The number of different filters also impacts the number of parameters of the network. In contrast to the fully connected neural networks, not all possible inputs contribute to one particular output. Thereby the model has fewer parameters compared to a dense network. Figure 5.6 represents an operation on one input grid. An image will typically have 3 channels; in this respect, each filter will be composed by 3 different kernels (in other words, by as much channels as the input has). The number of kernels equals the number of output grids.

The way convolution operations proceed influences the patterns they detect. The filters have similar values for comparable patterns in the input, regardless of where exactly the features are. Relating to this application it means, that a pattern in relative difficulty of the questions or in the percentage of an error type in questions with neighbouring operands will be detected by the same filter. The feature vector is comprised by measurable characteristics (in images those are the shapes) that are used in distinguishing the target. Bigger weights are given to patterns that appear more often, because they are the most informative for the decision.

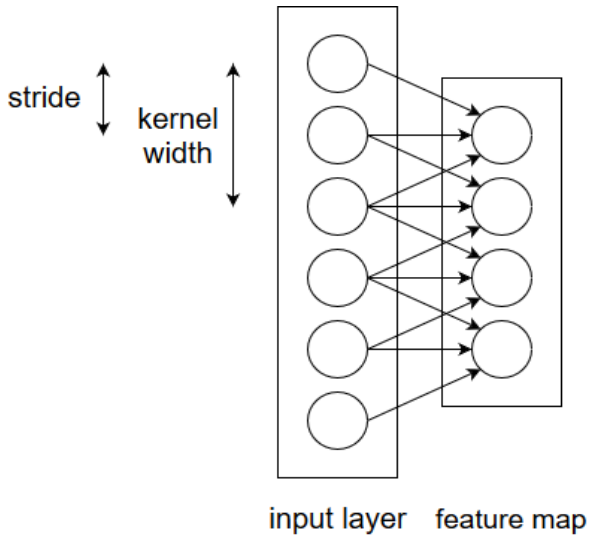


Figure 5.8: One-dimensional convolution operation and its parameters. The number of weights and connections is lower than the one of the corresponding fully connected network.

One characteristic that speaks for the convolution operation is the fact, that input data often have similar values around a neighbourhood. Nevertheless, this is not a hard requirement since there are successful applications where this property

does not apply strictly. The more deeper the layer in the network is, the more elements of the input are indirectly used for the computation of the feature maps. The area of the input that contributes to the computed value of a feature map is called “receptive field”. Its size can be influenced by padding, strides, and the pooling parameters.

The learned patterns of a densely connected neural network are based on the whole image; every pixel value contributes to the output of each layer. The learned patterns have global meaning and all pixels are necessary for its detection. On the contrary, since the convolution operation is a computation involving an area of the image, the patters are local and the detected pattern can be anywhere in the image. The shared parameters, as seen in the simpler one-dimensional case 5.8, contribute to translation equivariance. This fact ensures that a shift in the input pattern towards a specific direction will be preserved in the feature map.

Deep CNNs learn patterns hierarchically and the depth of the network contributes to their creation. The complex patterns that are learned in later layers are compositions of simpler patterns learned in the first layers. In Aggarwal (2018) the example of first layers detecting lines that in further layers are assembled to shapes like hexagons, which then in the last layers are composed to a honeycomb, describes the phenomenon in an exemplified way. Typical CNN architectures have an increased number of filters in later layers compared to earlier ones.

As in dense neural networks, different architectures with different configurations must be tried out to discover the one with the best performance. Typical filter sizes are 3×3 and 5×5; the larger filters are usually found in the first layers of the network. Smaller filter sizes result in larger feature maps and lower parameter footprint. They also support the need for a deeper network, which in turn increases the receptive field thereby capturing complex patterns in larger areas of the image. Padding and small stride ensure that relevant information contributes equally and also increases the size of the feature maps. Max pooling account for local invariance, meaning that small input shifts do not influence the values of the feature maps; one of the primary goals of this thesis was to prohibit this property.

5.11 Applications of Convolutional Neural Networks

Convolutional neural networks are used extensively in image processing applications (Patterson and Gibson (2017); Aggarwal (2018)). Along with image classification, which is used for tasks in medical diagnosis and autonomous vehicles, object recognition and localization are one of the main application fields used in industry (Zhao et al. (2019)). Video processing is an obvious extension to three dimensional convolution operations, sometimes combined with recurrent neural networks. Different tasks require different architectures (Springenberg et al. (2014)); for example, the number of pooling layers or dense layers in the end of the network, but the basic components remain the same. Popular architectures are the AlexNet, VGG Net, GoogLeNet, and DeConvNet (Patterson and Gibson (2017); Buduma and Locascio (2017); Canziani et al. (2016)). Natural language processing can also benefit from CNNs, despite the fact that the importance of a word depends on its position in a sentence (Zhang and Wallace (2015)). Recent advances in deep reinforcement learning in the field of video games like Atari (Mnih et al. (2013)) use a preprocessed image of the game as input to the Deep CNN. Board games like Go and chess (Pumperla and Ferguson (2018)) encode their input state as a two-dimensional grid. The network finds a favourable strategy to win the game supported by the features found by the network.

6. Evaluation

6.1 Evaluation for the simulated case

6.1.1 Formulation of the problem

In the learning-aware application the Markov property, as described in equation 5.1, is satisfied, because the probability of the next state is the probability of the answer to the posed question. The transition probability $P(s_{t+1}|s_t, a_t)$ of being at state s_{t+1} from state s_t given the posed question a_t depends on the student's answer and can be computed by the equation 3.6. For each question there are 100 possible answers that lead to 100 different learning competence states. The probabilistic graphical model can provide this probability; assuming that it accurately represents the learning state, the probability of each answer of a particular question can be found by probabilistic query and can be computed directly from the corresponding conditional probability tables. Then, assuming this answer was given, our parameter-learning algorithm can change our belief about the parameters of the overall learning state. It can be regarded as a new training sample that helps to update the parameters of the model and compute the next state s_{t+1}. The state transition model is also fully formulated.

The starting state is the probabilistic graphical model of each user with its currently learned parameters. The final state is the one where all questions were answered correctly the last two times they were posed. This could result the choice of only "easy" questions. Therefore, it is not enough to specify the terminal state as the one where merely all $\textbf{Learning State}_{\mathbf{q}}$ random variables have a probability near 1 for the "correct" outcome.

Each question should be answered correctly at least two times, because an acci-

dental correct answering can be misleading. As seen in figure 6.1, a history of the current answers of all questions must be kept. Each question can be posed different number of times. If the last two answers of each question are answered correctly, then the user has learned all questions correctly. The algorithm was tested first in the trivial case where the user always answers the posed question correctly. That means that the reinforcement algorithm's proposal was to present each possible question twice so that after training the terminal state would be reached after 180 steps.

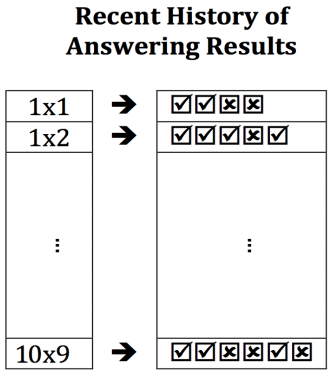


Figure 6.1: Recent history of answering results: The last two times each question was posed were answered correctly.

The architecture of the application is depicted in figure 6.2. The input of the Deep Reinforcement Learning network is the current learning competence (which is fully captured by the Probabilistic Graphical Models) and the recent history of answering results. The outputs are the Q-values of the posed questions, which drive the selection of the next question. Posing that question results in an answer, which in turn updates the beliefs about the learning competence of the student and the recent history of the results. This comprises the new input state being fed to the network; the process is repeated until the terminal state is reached.

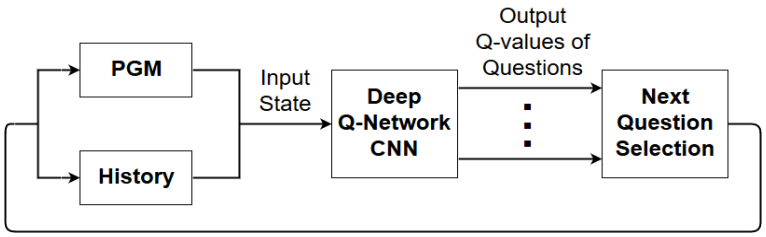


Figure 6.2: Architecture of the Deep Q-Network and its interaction with the environment.

The training of the network was made by simulated interaction with the learned learning competence model. This gave the opportunity to create the necessary amount of artificial training data that current game-playing Deep Reinforcement Learning algorithms use (with some exceptions, such as the AlphaGo, which partially also uses data from games of human experts (Pumperla and Ferguson (2018))). The answer to the proposed question was sampled with the method described in section 4.3.

The reward scheme currently used is not the one that was initially desired. The "Minimum Time to Goal" encourages the fastest possible trajectory to the terminal state. It gives -1 to all states except the last one (terminal), which has reward 0. This "deprived" the gradients and the maximum Q-value was not increasing with time. Game applications also use a different reward scheme where the reward is defined by the score difference. If a game action increase the score, the reward is positive; in the opposite situation it becomes negative. By the same means, a reward scheme that worked successfully gives $+1$ to every correct answered question and -1 to every wrongly answered one.

6.1.2 Concrete Architecture

All parameters of learning competence models consist the input of the deep Q-Network. After analysing its properties in chapter 2 and 3 it is observable that questions with similar operands have similar characteristics, such as the proportion of wrong answers or the impact of a specific error type. There are certainly exceptions to this property, as for example the tie questions where both operands are the same. But the same applies to images that are frequently used as an input to Deep Reinforcement Learning networks basing on the convolution operation.

The Deep Reinforcement Learning network with its inputs and its outputs is depicted in figure 6.3; this figure is a detailed zoom in the "Deep Q-Network CNN" component of the figure 6.2. The input has nine channels; the first seven correspond to each error type, the eighth contains the proportion of correct or wrong answers, and the ninth an encoding of the recent history of the answered behavior. The array of each channel has the size of 10×9, which is the total number of posed questions. The input does not have the same characteristics as an image. Nevertheless, there

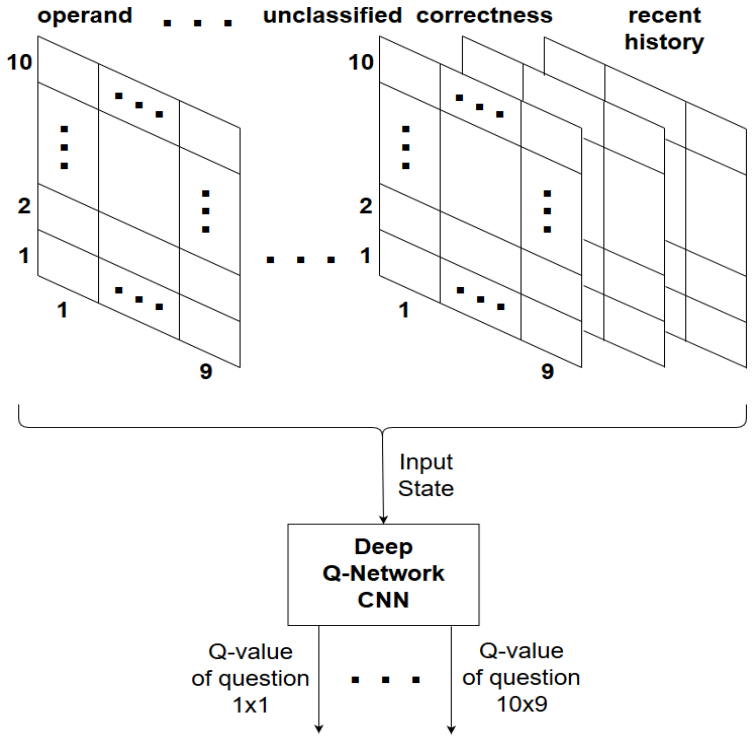


Figure 6.3: Convolutional neural network's detailed view of input and output

are examples in recent deep learning literature where, for example, the game state is represented by a vector or a matrix (Pumperla and Ferguson (2018)) and is used as an input to a CNN.

The input of the CNN is $9 \times 10 \times 9$ for $10 \times 9 = 90$ questions. The CNN has three layers with 200 filters each. The first layer has 200 filters of size 2×2 with 9 elements each for the 9 input "channels" (operand, ..., history). The second layer has 200 filters of size 3×3. Each of them has 200 elements, which equals the number of outputs of the previous layer. The third layer has similarly 200 of size 4×4 with 200 elements each. So although the input has 9 "channels", the output of the first layer has 200 "channels" and the same applies for the outputs of the second and third layer. There is one fully connected layer with $10 \times 9 \times 200 = 18000$ inputs and 90 outputs. The overall architecture is depicted in figure 6.5.

The activation function applied after each layer is the Rectified Linear Unit (ReLU) (equation 5.10). There was no pooling to avoid subsampling and loss of information, as explained in section 5.10. For similar reasons and to keep the influence of each error type in every question, the "same" padding scheme was used.

The total number of steps were two million, as seen in figure 6.6. The maximum

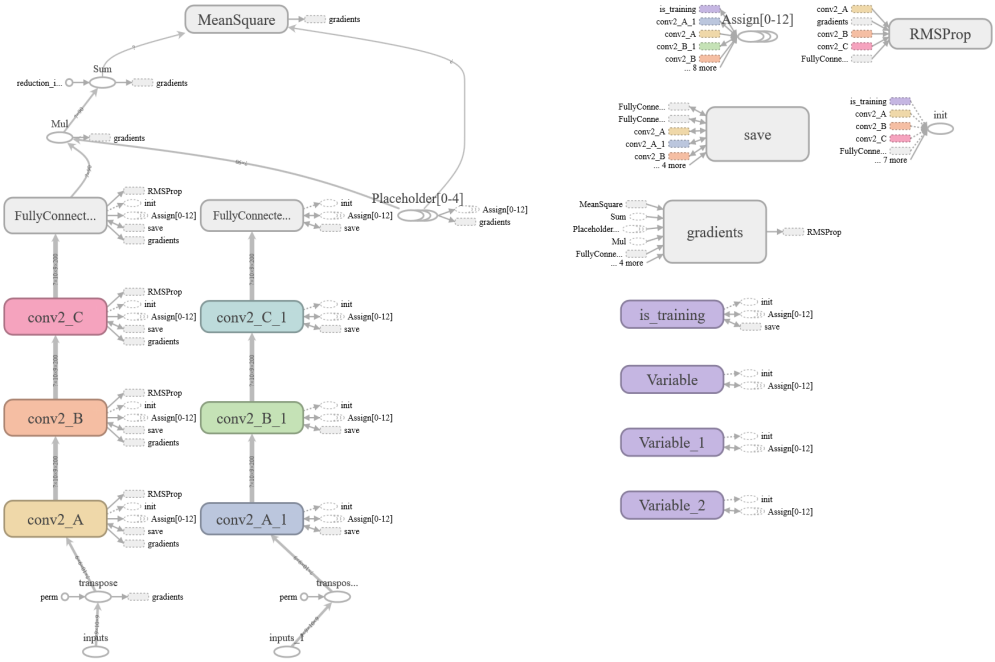


Figure 6.4: Graph of the two Convolutional Neural Networks in Tensorboard (`https://www.tensorflow.org/guide/summaries_and_tensorboard`, Last accessed 17 March 2019), the visualization tool of tensorflow (`https://www.tensorflow.org/`, Last accessed 17 March 2019)

Q-value may take negative values in the first steps, but tendentiously grows into positive values over the course of the algorithm. After trying out several combinations of the parameters and observing mostly the evolution of the maximum Q-value, the chosen learning rate was set to the value 0.0001 and the γ parameter to 0.9.

The ϵ parameter is tendentiously decreasing with time - thereby allowing more exploration in the first steps and more exploitation in the last ones - containing randomness created by the different workers annihilation range: Each of the 8 workers which work asynchronously picks a different ϵ annealing rate; 40% of them pick anneal the value until 0.01, 30% until 0.1 and 30% until 0.5 Mnih et al. (2016).

The size of the CNN and particularly the number of filters is a hyperparameter, as explained in 5.7. The sizes of the filters were chosen according to the principle that they increase the deeper the model gets (section 5.10). Nevertheless, the experimentation of different hyperparameters as well as their influence in the performance of the task, is part of future work. By no means, the network is unique or optimal,

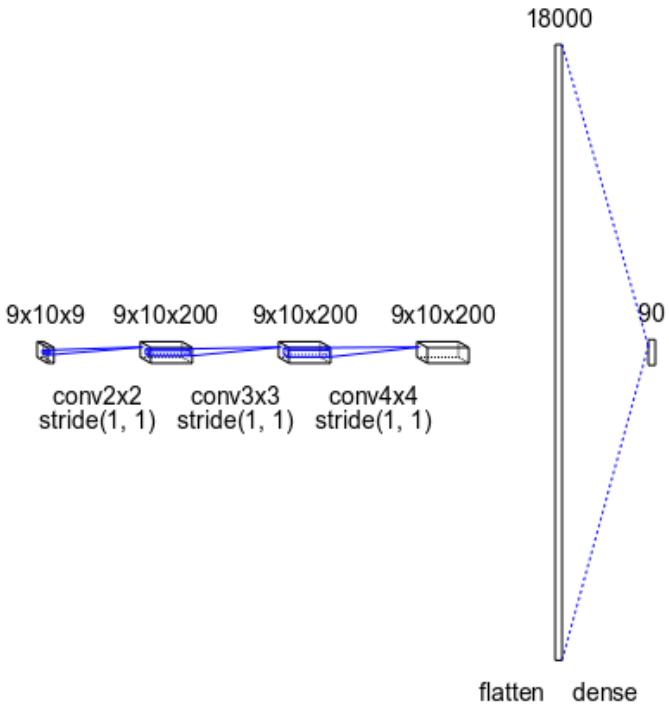


Figure 6.5: Convolutional neural network architecture. The input on the left side is $9 \times 10 \times 9$ and the output of the three layers have exactly the same size.

but solves the problem effectively. It creates the base for further research.

6.2 Visualisations

Vizualization of the convolutional network elements are an important method to gain insight in their comprehension of the problem as well as the means they encode and process of the input information to fulfil the needs of the problem. There are several techniques that can be used to plot the characteristics of the filters. Apart from the visualization of the filters, the input states, this thesis uses two of them in order to achieve the interpretability (Hall and Gill (2018); Chollet (2018); Holzinger (2018a); Holzinger et al. (2017a,b); Holzinger (2018b)) of the decisions of the deep neural network to a certain extent. The visualizations were made with the Python library seaborn [1]

[1] `https://seaborn.pydata.org/`, Last accessed 17 March 2019

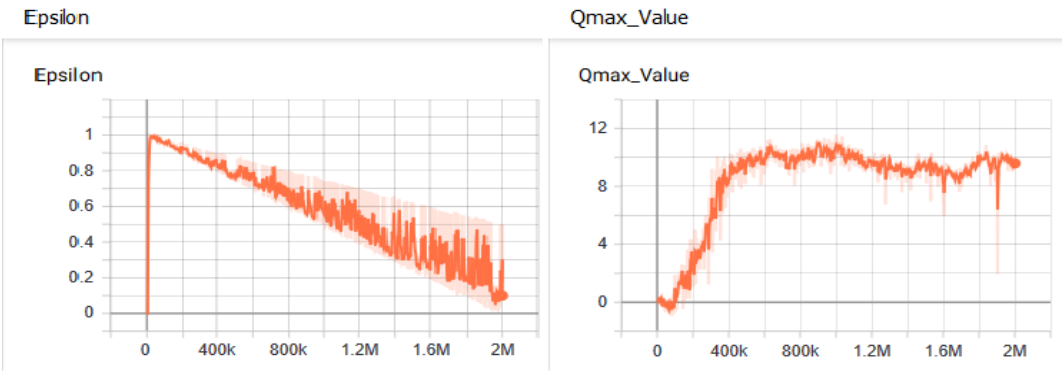


Figure 6.6: Evolution of ϵ parameter and the maximum Q-value of the Reinforcement Learning algorithm during training steps 5.9

6.2.1 Visualization of the filters

Overall $200 \times 9 = 1800$ filter elements of the size 2×2, $200 \times 200 = 40000$ of size 3×3 and $200 \times 200 = 40000$ of size 4×4 can be plotted.

6.2.2 Visualization of the starting input state

Figure 6.7 depicts the starting input state. Each of the nine elements of the figure is one "channel" starting from the operand error until the encoding of the recent history (two latest answers). Although not precisely quantified, the figures indicate that nearby grid elements have usually similar values. The starting input state is not each learner's individual learning competence, but the learning competence of the students in the training set.

6.2.3 Visualization of the outputs of activations layers

The output of each layer also provides insights about the information that "passes through" it w.r.t. a particular input state. In image processing tasks, if the outputs have a particular pattern that is recognizable, then the combination of the convolution operation and the activation function's effect on the input image is directly interpretable.

The framework provides the functionality to create every possible state (any valid probabilistic graphical model and recent history parameter) as well as to plot the state itself and the corresponding outputs of the layers after a feedforward pass.

6.2.4 Visualization of the input grid that maximizes the value of the activation function of each filter

Instead of trying out several input states and trying to recognize the functionality of one filter by its output, which is the input information in a transformed form, it is possible to begin with the filter and find out the input grid to which each filter "responds" the most. In this context, maximum response means that the gradient descent, as described in 5.7, searches for an input grid that maximizes the output of the activation function (Chollet (2018)). The process applies gradient descent to an input grid that is initialized as if it were a blank image.

The results of this method are depicted in figure 6.8. The state has 9 elements ordered in a 3×3 plot. Many of the filters of the first layer are simple and can be interpreted. For example, the input state can be one where all operand errors have a uniform value or an increasing one, as one proceeds from smaller to larger operands. Although, there are some patterns that can be detected and the combination of the patterns of the previous layer is reasonably detectable in some cases, currently there is not a straightforward combination functionality that can be derived from those images.

Also, it is observable in figure 6.8 that the detected grids are quite similar to each other, although they represent different error types. The learning competence state's correctness and recent history (second to last and last subplot) are different from the rest; in those images they appear similar. Some exceptions exist, as shown in figure 6.9 where the history part stands out.

Future work must contain different methods, like sensitivity analysis and relevance propagation providing different interpretations about the impact of each input grid element in the output value (in this application: the question decision) (Bach et al. (2015)).

6.3 Reasoning of proposed sequences

On average, the simulation that takes the wrong answers into account needs 192.72 steps to reach the terminal state, which is almost 12 more than in the case where

all provided answers are correct. Starting with the same input state, the proposed sequences start almost the same as well as different episodes proceed quite similar. Though, the wrongly answered questions do not happen at the same question index, as seen in figure 6.10. That means that the trained network does not propose a rigid sequence of questions, but adapts itself to the learning competence of the user. A wrongly answered question (by the process of sampling) affects the next posed question through the probabilistic graphical model; the recent history contents change.

Overall, the episodes have some commonalities. The question sequences almost always seem to start with the same questions. Nevertheless, the step at which the first error happens as well as the question that was posed at it, effects the path from that time on. In that means, a basic adaptation functionality is achieved. The reinforcement learning algorithm contains an internal logic that gives different priority to the actions of the agent, reacting to the behaviour of the environment. Each student has some similarities with the others, but at the same time, is not confronted with exactly the same learning sequence, regardless of his or her learning competence.

Reoccurring patterns consist of posing the same question and then immediately posing it again, so the proposed sequences are comprised mostly by pairs of the same question posed next to each other. Although this is not always the case, several episodes where a posed question is answered false for the second time, "send" this question at the end of the episode. This is in accordance to the fact that the overall return will be less affected by a question that has a higher probability to be answered wrongly.

These effects are influenced by the degree of change of the probabilistic graphical model of the posed question and the fact that the input state does not change immensely (as a result of the fractional EM-update) after the answer was sampled. Although, one can detect some expected patterns in the reinforcement learning's algorithm logic, like encountering relatively easy questions in the beginning, or trying out the same question twice, the criteria that turns the balance is not straightforward. As for example, in a chess game or the game of Go it is easier for humans to reason the last steps of the strategy, than intermediate steps that may seem surprising in general.

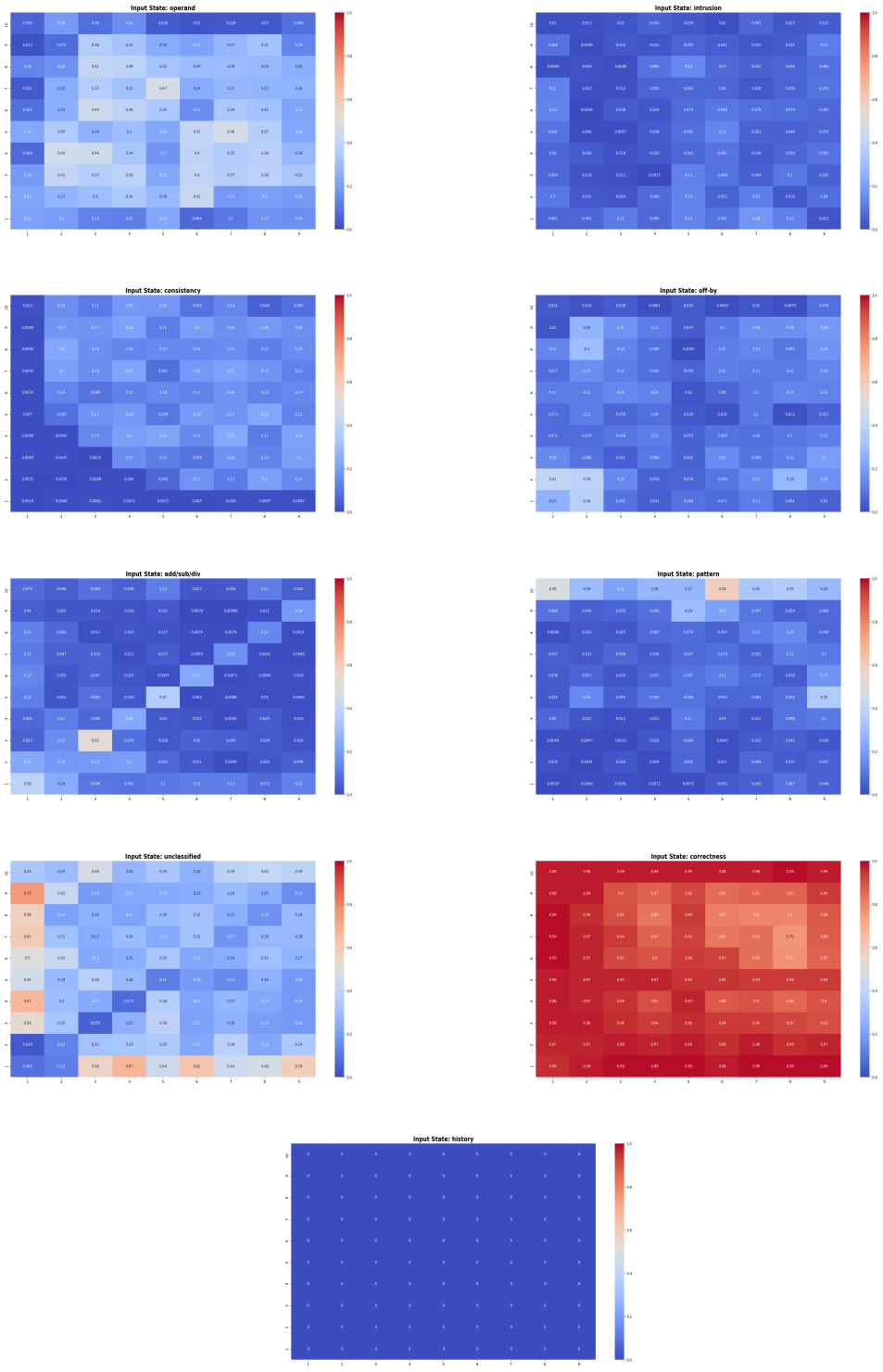


Figure 6.7: Visualisation of the input state: The first seven figures correspond to the error types, the eighth to the correctness proportion, and the last one to the history, which is initialized for all operand combinations to zero. Compare the "correctness" type with the figure 2.5 in chapter 2.

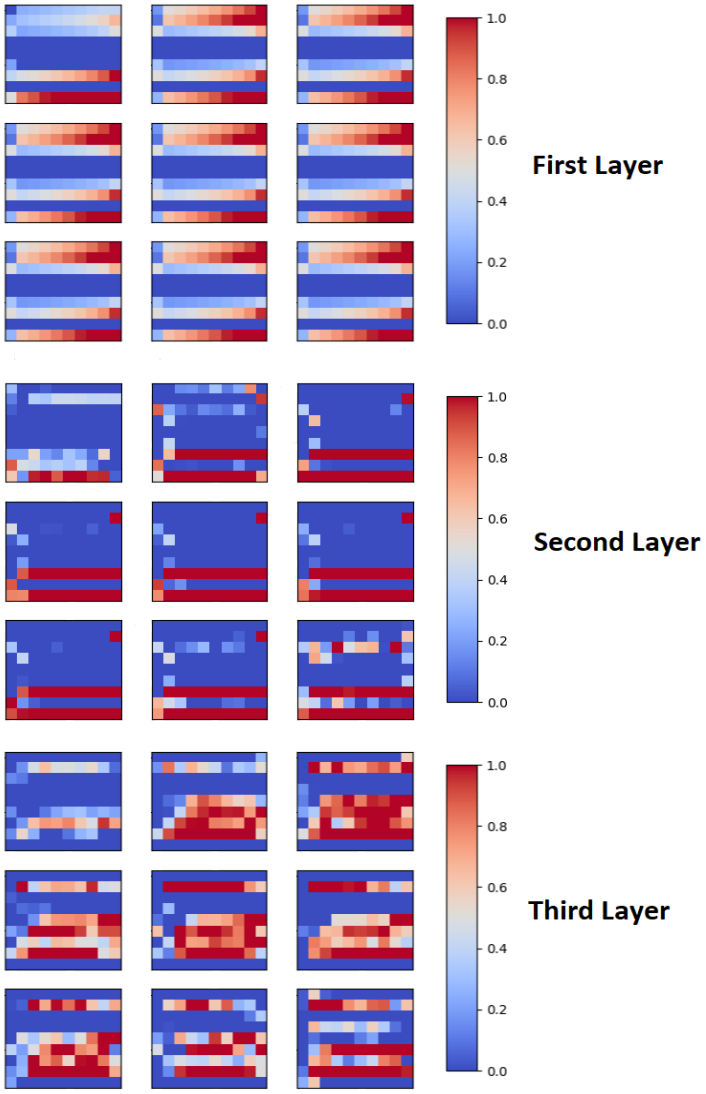


Figure 6.8: Input grid that maximizes the value of the activation function for connected filters in the first, second, and third layer. The values are normalized between 0.0 and 1.0.

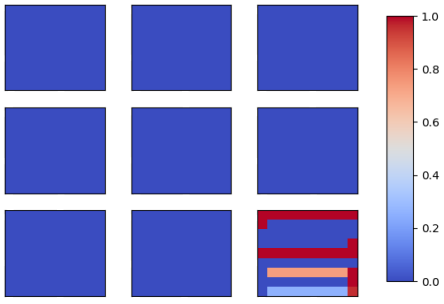
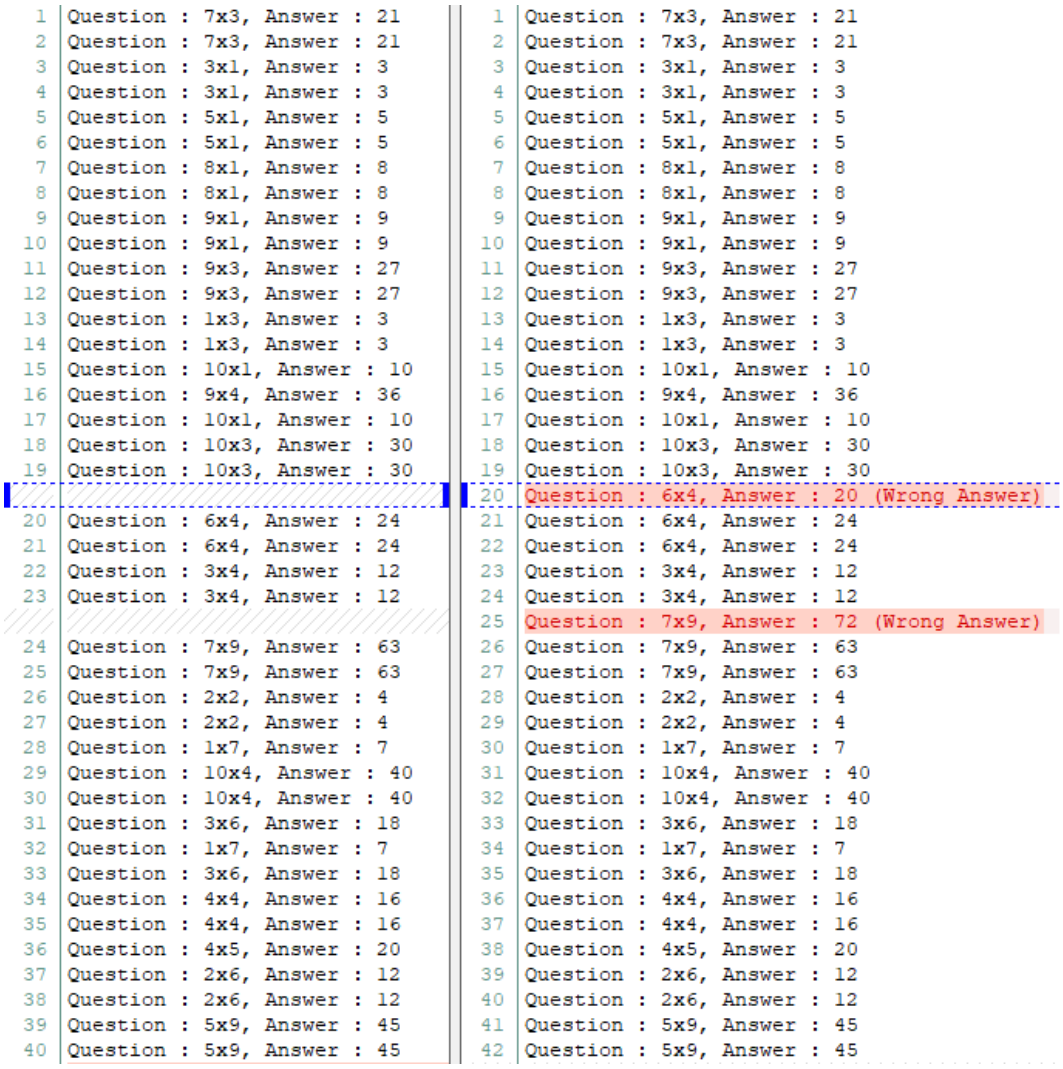


Figure 6.9: Input grid that maximizes the value of the activation function after filter number 63 (out of 200) of the first layer.



1	Question : 7x3, Answer : 21	1	Question : 7x3, Answer : 21
2	Question : 7x3, Answer : 21	2	Question : 7x3, Answer : 21
3	Question : 3x1, Answer : 3	3	Question : 3x1, Answer : 3
4	Question : 3x1, Answer : 3	4	Question : 3x1, Answer : 3
5	Question : 5x1, Answer : 5	5	Question : 5x1, Answer : 5
6	Question : 5x1, Answer : 5	6	Question : 5x1, Answer : 5
7	Question : 8x1, Answer : 8	7	Question : 8x1, Answer : 8
8	Question : 8x1, Answer : 8	8	Question : 8x1, Answer : 8
9	Question : 9x1, Answer : 9	9	Question : 9x1, Answer : 9
10	Question : 9x1, Answer : 9	10	Question : 9x1, Answer : 9
11	Question : 9x3, Answer : 27	11	Question : 9x3, Answer : 27
12	Question : 9x3, Answer : 27	12	Question : 9x3, Answer : 27
13	Question : 1x3, Answer : 3	13	Question : 1x3, Answer : 3
14	Question : 1x3, Answer : 3	14	Question : 1x3, Answer : 3
15	Question : 10x1, Answer : 10	15	Question : 10x1, Answer : 10
16	Question : 9x4, Answer : 36	16	Question : 9x4, Answer : 36
17	Question : 10x1, Answer : 10	17	Question : 10x1, Answer : 10
18	Question : 10x3, Answer : 30	18	Question : 10x3, Answer : 30
19	Question : 10x3, Answer : 30	19	Question : 10x3, Answer : 30
		20	Question : 6x4, Answer : 20 (Wrong Answer)
20	Question : 6x4, Answer : 24	21	Question : 6x4, Answer : 24
21	Question : 6x4, Answer : 24	22	Question : 6x4, Answer : 24
22	Question : 3x4, Answer : 12	23	Question : 3x4, Answer : 12
23	Question : 3x4, Answer : 12	24	Question : 3x4, Answer : 12
		25	Question : 7x9, Answer : 72 (Wrong Answer)
24	Question : 7x9, Answer : 63	26	Question : 7x9, Answer : 63
25	Question : 7x9, Answer : 63	27	Question : 7x9, Answer : 63
26	Question : 2x2, Answer : 4	28	Question : 2x2, Answer : 4
27	Question : 2x2, Answer : 4	29	Question : 2x2, Answer : 4
28	Question : 1x7, Answer : 7	30	Question : 1x7, Answer : 7
29	Question : 10x4, Answer : 40	31	Question : 10x4, Answer : 40
30	Question : 10x4, Answer : 40	32	Question : 10x4, Answer : 40
31	Question : 3x6, Answer : 18	33	Question : 3x6, Answer : 18
32	Question : 1x7, Answer : 7	34	Question : 1x7, Answer : 7
33	Question : 3x6, Answer : 18	35	Question : 3x6, Answer : 18
34	Question : 4x4, Answer : 16	36	Question : 4x4, Answer : 16
35	Question : 4x4, Answer : 16	37	Question : 4x4, Answer : 16
36	Question : 4x5, Answer : 20	38	Question : 4x5, Answer : 20
37	Question : 2x6, Answer : 12	39	Question : 2x6, Answer : 12
38	Question : 2x6, Answer : 12	40	Question : 2x6, Answer : 12
39	Question : 5x9, Answer : 45	41	Question : 5x9, Answer : 45
40	Question : 5x9, Answer : 45	42	Question : 5x9, Answer : 45

Figure 6.10: Starting sequences of different episodes, which are quite similar although the answers are somewhat different. The comparison of the files is made by the program DiffMerge `https://sourcegear.com/diffmerge/`

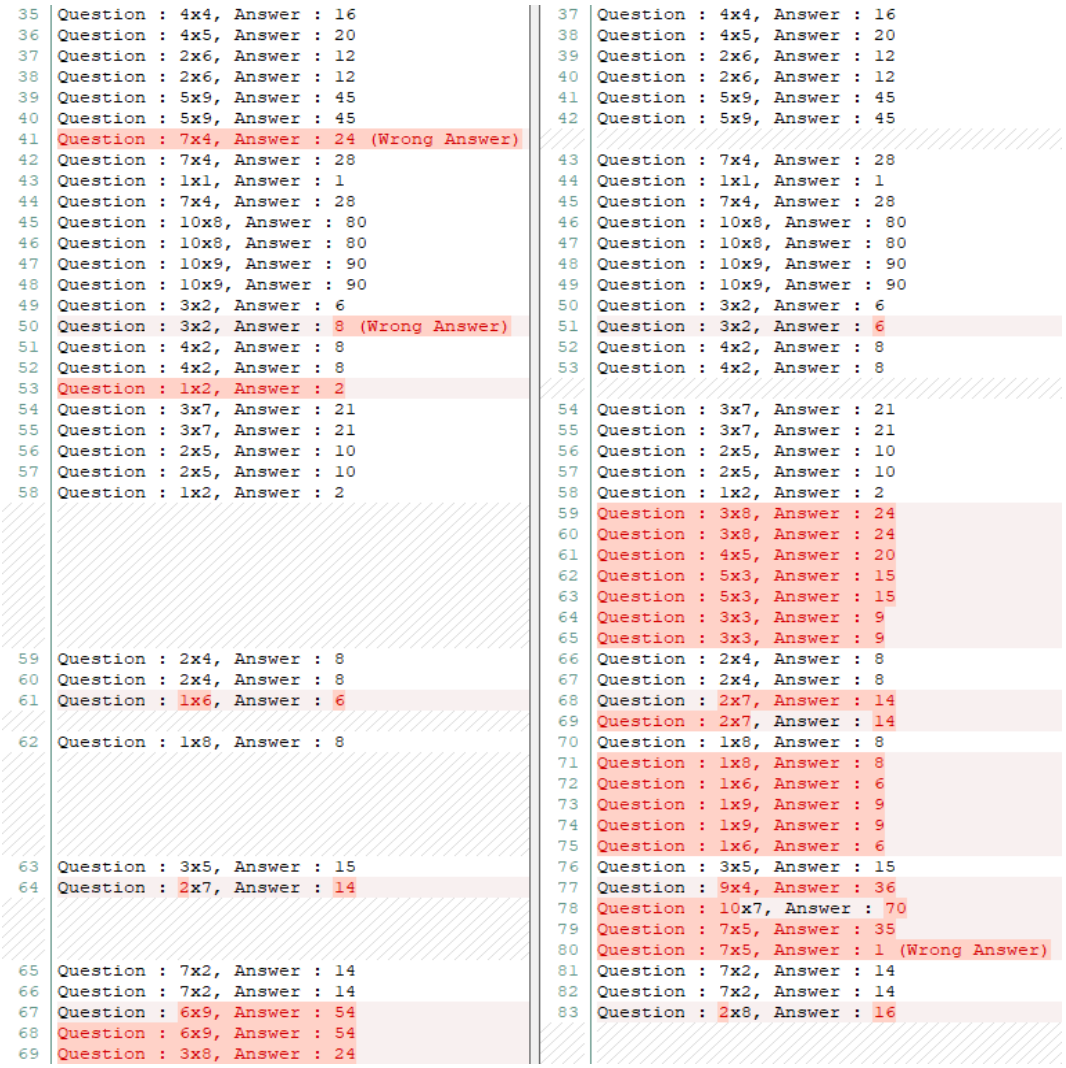


Figure 6.11: Differing sequences of questions, because the simulated students provided different answers somewhere on the line.

7. Software Engineering, Testing and Quality

7.1 Python Libaries

The code was written in Python 3.5. The necessary libraries are:

- `pgmpy`: Library for the definition of the probabilistic graphical model, its sampling and the probability queries [1]
- `matplotlib`: Library for plotting [2]
- `tensorflow`: Libary for neural networks [3]

7.2 Files Hierarchy and Documentation

7.2.1 Directory and files hierarchy

1. `Start.py`: Main module.
2. `InputData`: Directory of input files
3. `OutputData`: Directory of output files, produced by the application - mostly images
4. `import_data`

[1] `https://github.com/pgmpy/pgmpy`, Last accessed 17 March 2019
[2] `https://matplotlib.org/`, Last accessed 17 March 2019
[3] `https://www.tensorflow.org/`, Last accessed 17 March 2019

(a) `csv_import.py`: File containing the class that provides functionality for the import and parsing of the information contained in the "Questions.csv" input file with the mapping from question identifiers to question string format (for example "5x5").

(b) `DataSet.py`: Keeps a class with two lists that need to be kept in sync; one containing the questions and one their corresponding answers comprising the training and test set. Useful methods on both lists provide their length, slicing, and shuffling.

(c) `input_data_parser.py`: Contains many methods providing statistics of the dataset and performing preprocessing functionality. Phenomena, such as answers that were never provided by any user (until now), were quantified. The difference in application usage (how many answers does a particular user answer) as well as how fast the terminal state is reached with the current approach needed corresponding histogram plots. Preprocessing steps described in chapter 2 as well as the training and test set 3.7 in chapter 3 are implemented in the methods contained in this file.

5. `error_types`

(a) `error_types_classificator.py`: Recognize all the error types to which a particular answer belongs to according to the posed question (which can be more than one, as explained in section 2.2).

(b) `error_types_questions.py`: Build up a map data structure that has the questions as keys. The values comprise another map with keys the error types and values the answers of this error type for that question. This data structure builds up the Conditional Probability Tables (CPT) described in sections 3.2 and 3.4.

6. `pgm_em`

(a) `em_batch_learner.py` Application of the batch version of the expectation-maximization algorithm described in section 3.6 on the whole selected training set. Plots that depict the increase of log likelihood with the number of EM steps are called from the implemented methods.

(b) `em_batch_updater.py` All methods that compute the derived parameter updates of the EM-algorithm described in section 3.6.3. The functionalities provided in this file are the ones needed in `em_batch_learner.py`

(c) `em_fractional_learner.py` Application of the fractional updating rules of the expectation-maximization algorithm on the whole selected training set, but only one sample at a time.

(d) `em_fractional_updater.py` All methods that compute the derived parameter updates of the EM-algorithm described in section 3.8.

(e) `em_utils.py` Utilities methods used mainly for computations for the EM-algorithm, such as the check that the likelihood does not decrease with the number of EM iterations.

(f) `generative_model_multi.py` Contains methods that use the learned models as generative models which can be sampled. The sampled dataset that is created from the generative models is used to learn the parameters anew and shows that they are similar enough to the ones learned in the first place.

(g) `pgm_utils.py` Utilities methods for the evaluation and comparison of the probabilistic graphical models such as the Bhattacharyya coefficient, as described in section 4.4.

(h) `PGModel_Multi.py` Definition of all probabilistic graphical models with the use of the Python library `pgmpy` 7.1. The library provided the necessary functionality for checking the consistency of the model, computing the joint distribution, sampling methods and probabilistic queries, which were necessary for all insights presented in chapter 4.

7. `rl`

(a) `one_step_qlearning_la.py` Training and evaluation methods of 1-step Q-Learning algorithm, as described in section 5.9. The definition of the convolutional neural network architecture, the number of parameters, and the training and evaluation methods that produce the plots of chapter 6 are included.

(b) `learning_aware_env.py` Class representing the environment. Transforms the learning competence to a state and triggers the state update, the computation of the appropriate reward (depending on the state), and the choice of next action.

(c) `state_per_worker.py` Holds the state of each worker in the reinforcement learning model. It contains the parameters of the probabilistic graphical models, the correctness percentage and the recent history of the answering results

8. `plots`

   - Plotting methods

9. `tests`

   - Testing methods, described in the next section 7.3

The documentation of the code was made with Sphinx [4]. The PEP 8 Style Guide is followed [5].

7.3 Testing

Software testing in python is made with the use of the pytest framework [6] Pichara and Pieringer (2017); Okken (2017). The expectation-maximization update rules, as described in chapter 3, were tested with some examples and compared to numerically computed results. But since the number of possible cases that must be tried out is very large, property-based testing was used Nilsson (2009). The hypothesis framework [7] gives the ability to write tests in an abstract manner, where the concrete numerical values are generated automatically.

An extended description of property-based testing is out of scope of this thesis, but the main idea is describing the properties of the function that is tested. By that means, generating several numerical examples was proven to be very effective. The fractional expectation-maximization update rule 3.8 has the following properties:

[4] `http://www.sphinx-doc.org`, Last accessed 17 March 2019
[5] `https://www.python.org/dev/peps/pep-0008/`, Last accessed 17 March 2019
[6] `https://docs.pytest.org/`, Last accessed 17 March 2019
[7] `https://hypothesis.readthedocs.io/en/latest/`, Last accessed 17 March 2019

- The sum of each row of the conditional probability tables must equal to 1.0 after the update.
- Only one of the $\mathbf{Answers_q}$ (namely the one that corresponds to the posed question), as well as the corresponding $\mathbf{Learning\ State_q}$ and $\mathbf{Correctness_q}$ will change.
- A sampled answer can belong to one or more error types. The conditional probability of this answer will be increased in the $\mathbf{Answers_q}$ conditional probability table. Because the sum must remain 1.0, the conditional probabilities of the other answers will be decreased after the update. Similarly, the error types that could be responsible for this answer will have an increased conditional probability in the $\mathbf{Learning\ State_q}$, whereas the rest conditional probabilities will decrease. If the answer is wrong, then the belief that the student's ability to answer a question correctly will fall, and the $\mathbf{Correctness_q}$ will change by the same means.

8. Conclusion

8.1 Conclusion

The answer to the research question of the possibility of personalization and improvement of the learning path as well as the method and the insights that are inferred from that, were the main achieved goals of this thesis. There is a basic methodology combining different machine learning methods for the achievement of this goal that can be evaluated in the real-case and expanded according to the needs of the learners and students. It can also be compared to current human and Learning Analytics applications that contain other methods to achieve similar goals.

Data from learning applications provide valuable insights for students and supervisors. The modelling component of each student is an entity that can be learned adaptively with the use of a set of probabilistic graphical models. The commonalities and differences between the students learning competence are quantified from their answers and the answers of their peers without the need for further demographical data. Even if an unknown student starts interacting with the learning application, his or her learning competence will be informed already, but it will be able to be updated immediately as the first answer is provided. The reporting methods can depict the changing status. Predictive modelling can estimate the probable answer to the next posed question.

But a learning aware application goes beyond modelling and reporting. Current reinforcement learning systems decide and act on the basis of their internal models. Although this cannot be even remotely compared to human tutoring, the insights are valuable for indicating how far different question sequencing methods might affect the learning capabilities of the students.

8.2 Future Research

Future work must deal primarily with the application of the learning-aware algorithm to real student questionnaires and measuring its performance in the improvement of the learning path of the users. A quantitative comparison with the previous version of the application and a thorough analysis of the differences of their insights about the personalization of learning is mandatory. Furthermore, one could conceptualize an analogous application in other learning topics, which could even reach the individual sequencing of a curriculum.

At the same time, since the framework's architecture has room for improvement itself and can use different CNN architectures in combination with other Deep Reinforcement Learning algorithms, a further research question concerns the commonalities and differences between those methods. Explainability can play a crucial role in this step; the reasons for the organization of learning as well as the decisive insights may change human learning strategies in an analogous way as AlphaGo's playing strategies suggested different winning paths in the game Go (Pumperla and Ferguson (2018)).

References

Aggarwal, C.C. [2018]. *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing.

Aherne, Frank J, Neil A Thacker, and Peter I Rockett [1998]. *The Bhattacharyya metric as an absolute similarity measure for frequency coded data. Kybernetika*, 34(4), pages 363–368.

Amarjyoti, Smruti [2017]. *Deep reinforcement learning for robotic manipulation-the state of the art. arXiv preprint arXiv:1701.08878.*

Arulkumaran, Kai, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath [2017]. *A brief survey of deep reinforcement learning. arXiv preprint arXiv:1708.05866.*

Bach, Sebastian, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek [2015]. *On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PloS one*, 10(7), page e0130140.

Barber, David [2012]. *Bayesian reasoning and machine learning.* Cambridge University Press.

Barga, Roger, Valentine Fontama, Wee Hyong Tok, and Luis Cabrera-Cordon [2015]. *Predictive analytics with Microsoft Azure machine learning.* Springer.

Bertsekas, Dimitri P and John N Tsitsiklis [2008]. *Introduction to probability.* 2 Edition. Athena Scientific, Belmont, MA, USA.

Bhattacharyya, Anil [1943]. *On a measure of divergence between two statistical populations defined by their probability distributions. Bull. Calcutta Math. Soc.*, 35, pages 99–109.

Bishop, Christopher [2006]. *Pattern recognition and machine learning.* Springer.

Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. [2016]. *End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.*

Brusilovsky, Peter and Eva Millán [2007]. *User models for adaptive hypermedia and adaptive educational systems.* In *The adaptive web*, pages 3–53. Springer.

Brusilovsky, Peter and Christoph Peylo [2003]. *Adaptive and intelligent web-based educational systems. International Journal of Artificial Intelligence in Education (IJAIED)*, 13(2-4), pages 159–172.

Buduma, Nikhil and Nicholas Locascio [2017]. *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms.* " O'Reilly Media, Inc.".

Bunt, Andrea and Cristina Conati [2003]. *Probabilistic student modelling to improve exploratory behaviour. User Modeling and User-Adapted Interaction*, 13(3), pages 269–309.

Busoniu, Lucian, Robert Babuska, Bart De Schutter, and Damien Ernst [2010]. *Reinforcement learning and dynamic programming using function approximators.* CRC press.

Campbell, Jamie ID [1995]. *Mechanisms of simple addition and multiplication: A modified network-interference theory and simulation. Mathematical cognition*, 1(2), pages 121–164.

Campbell, Jamie ID [1997]. *On the relation between skilled performance of simple division and multiplication. Journal of Experimental Psychology: Learning, Memory, and Cognition*, 23(5), pages 1140–1159.

Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello [2016]. *An analysis of deep neural network models for practical applications. arXiv preprint arXiv:1605.07678.*

Chang, Kai-min, Joseph Beck, Jack Mostow, and Albert Corbett [2006]. *A bayes net toolkit for student modeling in intelligent tutoring systems.* In *Proceedings of the 8th International Conference on Intelligent Tutoring Systems*, pages 104–113. Springer.

Chollet, Francois [2018]. *Deep learning with python.* Manning Publications.

Chrysafiadi, Konstantina and Maria Virvou [2013]. *Student modeling approaches: A literature review for the last decade. Expert Systems with Applications*, 40(11), pages 4715–4729.

Conati, Cristina, Abigail Gertner, and Kurt Vanlehn [2002]. *Using Bayesian networks to manage uncertainty in student modeling. User modeling and user-adapted interaction*, 12(4), pages 371–417.

Conati, Cristina, Abigail S Gertner, Kurt VanLehn, and Marek J Druzdzel [1997]. *On-line student modeling for coached problem solving using Bayesian networks.* In *User Modeling: Proceedings of the Sixth International Conference, UM97*, pages 231–242. Springer, Vienna, Austria.

Cover, Thomas M and Joy A Thomas [2012]. *Elements of information theory.* John Wiley & Sons.

Danaparamita, Muhammad and Ford Lumban Gaol [2014]. *Comparing Student Model Accuracy with Bayesian Network and Fuzzy Logic in Predicting Student Knowledge Level. International Journal of Multimedia and Ubiquitous Engineering*, 9(4), pages 109–120.

Danks, David [2014]. *Unifying the mind: Cognitive representations as graphical models.* MIT Press.

Derpanis, Konstantinos G [2008]. *The bhattacharyya measure. Mendeley Computer*, 1(4), pages 1990–1992.

Domahs, Frank, Margarete Delazer, and Hans-Christoph Nuerk [2006]. *What makes multiplication facts difficult: Problem size or neighborhood consistency? Experimental Psychology*, 53(4), pages 275–282.

Ebner, Martin, Benedikt Neuhold, and Martin Schön [2013a]. *Learning Analytics–wie Datenanalyse helfen kann, das Lernen gezielt zu verbessern*. In Wilbers, K. and A. Hohenstein (Editors), *Handbuch E-Learning-Expertenwissen aus Wissenschaft und Praxis-Strategie, Instrumente, Fallstudien*, 48, Erg.-Lfg Edition, pages 1–20. Deutscher Wirtschaftsdienst (Wolters Kluwer Deutschland).

Ebner, Martin and Martin Schön [2013]. *Why learning analytics in primary education matters. Bulletin of the Technical Committee on Learning Technology*, 15(2), pages 14–17.

Ebner, Martin, Martin Schön, Behnam Taraghi, and Michael Steyre [2013b]. *Teachers Little Helper: Multi-Math-Coach. International Association for Development of the Information Society.*

Ebner, Martin, Behnam Taraghi, Anna Saranti, and Sandra Schön [2015]. *Seven features of smart learning analytics-lessons learned from four years of research with learning analytics. eLearning papers*, 40, pages 51–55.

Freedman, D., R. Pisani, and R. Purves [2007]. *Statistics: Fourth International Student Edition.* W.W. Norton & Company.

Gamboa, Hugo and Ana Fred [2002]. *Designing intelligent tutoring systems: a bayesian approach. Enterprise information systems*, 3, pages 452–458.

García, Patricio, Analía Amandi, Silvia Schiaffino, and Marcelo Campo [2007]. *Evaluating Bayesian networks' precision for detecting students' learning styles. Computers & Education*, 49(3), pages 794–808.

Godsey, Brian [2017]. *Think Like a Data Scientist.* Manning Publications.

Goebel, Randy, Ajay Chander, Katharina Holzinger, Freddy Lecue, Zeynep Akata, Simone Stumpf, Peter Kieseberg, and Andreas Holzinger [2018]. *Explainable AI: the new 42?* In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pages 295–303. Springer.

Goguadze, George, Sergey Sosnovsky, Seiji Isotani, and Bruce M McLaren [2011a]. *Towards a bayesian student model for detecting decimal misconceptions.* In *Proceedings of the 19th International Conference on Computers in Education*, pages 34–41. Chiang Mai, Thailand.

Goguadze, George, Sergey A Sosnovsky, Seiji Isotani, and Bruce M McLaren [2011b]. *Evaluating a Bayesian Student Model of Decimal Misconceptions.* In *Proceedings of the 4th International Conference on Educational Data Mining*, pages 301–306. Citeseer.

Goodman, Noah D, Joshua B. Tenenbaum, and The ProbMods Contributors [2016]. *Probabilistic Models of Cognition.* `http://probmods.org/v2`. Accessed: 2018-8-25.

Hall, Patrick and Navdeep Gill [2018]. *Introduction to Machine Learning Interpretability.* O'Reilly Media, Incorporated.

Harmon, Mance E and Stephanie S Harmon [1997]. *Reinforcement Learning: A Tutorial.* Technical Report, WRIGHT LAB WRIGHT-PATTERSON AFB OH.

He, Xuming, Richard S Zemel, and Miguel Á Carreira-Perpiñán [2004]. *Multiscale conditional random fields for image labeling.* In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–II. IEEE.

Hessel, Matteo, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver [2018]. *Rainbow: Combining improvements in deep reinforcement learning.* In *Thirty-Second AAAI Conference on Artificial Intelligence.*

Holzinger, Andreas [2018a]. *Explainable AI (ex-AI). Informatik-Spektrum*, 41(2), pages 138–143.

Holzinger, Andreas [2018b]. *From Machine Learning to Explainable AI.* In *2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)*, pages 55–66. IEEE.

Holzinger, Andreas, Chris Biemann, Constantinos S Pattichis, and Douglas B Kell [2017a]. *What do we need to build explainable AI systems for the medical domain? arXiv preprint arXiv:1712.09923.*

Holzinger, Andreas, Bernd Malle, Peter Kieseberg, Peter M Roth, Heimo Müller, Robert Reihs, and Kurt Zatloukal [2017b]. *Towards the augmented pathologist: Challenges of explainable-ai in digital pathology. arXiv preprint arXiv:1712.06657.*

Jaakkola, Tommi S and Michael I Jordan [1999]. *Variational probabilistic inference and the QMR-DT network. Journal of artificial intelligence research*, 10, pages 291–322.

Jensen, Finn V and Thomas D Nielsen [2007]. *Bayesian Networks and Decision Graphs.*

Karkera, Kiran R [2014]. *Building probabilistic graphical models with Python.* Packt Publishing Ltd.

Käser, Tanja, Severin Klingler, Alexander G Schwing, and Markus Gross [2017]. *Dynamic Bayesian networks for student modeling. IEEE Transactions on Learning Technologies*, 10(4), pages 450–462.

Klinkenberg, Sharon, Marthe Straatemeier, and Han LJ van der Maas [2011]. *Computer adaptive practice of maths ability using a new item response model for on the fly ability and difficulty estimation. Computers & Education*, 57(2), pages 1813–1824.

Kober, Jens, J Andrew Bagnell, and Jan Peters [2013]. *Reinforcement learning in robotics: A survey. The International Journal of Robotics Research*, 32(11), pages 1238–1274.

Kochenderfer, Mykel J [2015]. *Decision making under uncertainty: theory and application.* MIT Press.

Koller, Daphne and Nir Friedman [2009]. *Probabilistic graphical models: principles and techniques.* MIT Press.

Kraja, Eltion, Behnam Taraghi, and Martin Ebner [2017]. *The Multiplication Table as an innovative Learning Analytics Application.* In *Proceedings of EdMedia: World Conference on Educational Media and Technology*, pages 810–820. Association for the Advancement of Computing in Education (AACE).

Lapin, Maksim, Matthias Hein, and Bernt Schiele [2015]. *Top-k multiclass SVM.* In *Advances in Neural Information Processing Systems*, pages 325–333.

Lapin, Maksim, Matthias Hein, and Bernt Schiele [2016]. *Loss functions for top-k error: Analysis and insights.* In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1468–1477.

Lazaric, Alessandro, Marcello Restelli, and Andrea Bonarini [2008]. *Reinforcement learning in continuous action spaces through sequential monte carlo methods.* In *Advances in neural information processing systems*, pages 833–840.

Lee, Michael D and Eric-Jan Wagenmakers [2013]. *Bayesian cognitive modeling: A practical course.* Cambridge university press.

Levy, Roy and Robert J. Mislevy [2016]. *Bayesian Psychometric Modelling.* Taylor & Francis Group, CRC Press.

MacKay, David JC [2003]. *Information theory, inference and learning algorithms.* Cambridge university press.

Markowska-Kaczmar, Urszula, Halina Kwasnicka, and Mariusz Paradowski [2010]. *Intelligent techniques in personalization of learning in e-learning systems.* In *Computational Intelligence for Technology Enhanced Learning*, pages 1–23. Springer.

Millán, Eva, John Mark Agosta, and José-Luis Pérez de la Cruz [2001]. *Bayesian student modeling and the problem of parameter specification. British Journal of Educational Technology*, 32(2), pages 171–181.

Millán, Eva, Tomasz Loboda, and Jose Luis Pérez-De-La-Cruz [2010]. *Bayesian networks for student model engineering. Computers & Education*, 55(4), pages 1663–1683.

Millán, Eva and José Luis Pérez-De-La-Cruz [2002]. *A Bayesian diagnostic algorithm for student modeling and its evaluation. User Modeling and User-Adapted Interaction*, 12(2-3), pages 281–330.

Millán, Eva, Mónica Trella, José-Luis Pérez-de-la Cruz, and Ricardo Conejo [2000]. *Using bayesian networks in computerized adaptive tests.* In *Computers and Education in the 21st Century*, pages 217–228. Springer.

Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu [2016]. *Asynchronous*

methods for deep reinforcement learning. In *Proceedings of the 33rd International conference on machine learning*, pages 1928–1937.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller [2013]. *Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. [2015]. *Human-level control through deep reinforcement learning. Nature*, 518(7540), pages 529–533.

Murphy, Kevin P. [2012]. *Machine Learning: A probabilistic perspective*. MIT Press.

Nilsson, Rickard [2009]. *ScalaCheck*.

Nouh, Yaser, P. Karthikeyani, and R. Nadarajan [2006]. *Intelligent tutoring system-bayesian student model*. In *1st International Conference on Digital Information Management*, pages 257–262. IEEE.

Okken, Brian [2017]. *Python Testing with Pytest: Simple, Rapid, Effective, and Scalable*. Pragmatic Bookshelf.

Papoulis, Athanasios and S Unnikrishna Pillai [2002]. *Probability, random variables, and stochastic processes*. 4 Edition. Tata McGraw-Hill Education.

Pardos, Zachary A, Neil T Heffernan, Brigham Anderson, and Cristina L Heffernan [2010]. *Using fine-grained skill models to fit student performance with Bayesian networks. Handbook of educational data mining*, pages 417–426.

Patterson, Josh and Adam Gibson [2017]. *Deep Learning: A Practitioner's Approach*. " O'Reilly Media, Inc.".

Pfeffer, Avi [2016]. *Practical Probabilistic Programming*. Manning Publications.

Pichara, Karim and Christian Pieringer [2017]. *Advanced Computer Programming in Python*. CreateSpace Independent Publishing Platform.

Pumperla, M. and K. Ferguson [2018]. *Deep Learning and the Game of Go*. Manning Publications Company.

Romero, Cristóbal and Sebastián Ventura [2010]. *Educational data mining: a review of the state of the art. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(6), pages 601–618.

Sallab, Ahmad EL, Mohammed Abdou, Etienne Perot, and Senthil Yogamani [2017]. *Deep reinforcement learning framework for autonomous driving. Electronic Imaging*, 2017(19), pages 70–76.

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver [2015]. *Prioritized experience replay. arXiv preprint arXiv:1511.05952.*

Schiaffino, Silvia, Patricio Garcia, and Analia Amandi [2008]. *eTeacher: Providing personalized assistance to e-learning students. Computers & Education*, 51(4), pages 1744–1754.

Schön, Martin, Martin Ebner, and Georg Kothmeier [2012]. *It's just about learning the multiplication table.* In Buckingham Shum, Simon, Dragan Gasevic, and Rebecca Ferguson (Editors), *Proceedings of the 2nd international conference on learning analytics and knowledge*, pages 73–81. ACM, New York, NY, USA.

Seidenberg, Mark S and James L McClelland [1989]. *A distributed, developmental model of word recognition and naming. Psychological review*, 96(4), pages 523–568.

Shwe, Michael and Gregory Cooper [1991]. *An empirical analysis of likelihood-weighting simulation on a large, multiply connected medical belief network. Computers and Biomedical Research*, 24(5), pages 453–475.

Siemens, George and Ryan SJ d Baker [2012]. *Learning analytics and educational data mining: towards communication and collaboration.* In *Proceedings of the 2nd international conference on learning analytics and knowledge*, pages 252–254. ACM.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. [2017]. *Mastering the game of go without human knowledge. Nature*, 550(7676), page 354.

Sousa, David A [2008]. *How the brain learns mathematics.* Corwin Press.

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller [2014]. *Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806.*

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov [2014]. *Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research*, 15(1), pages 1929–1958.

Stacey, Kaye and Jennifer Flynn [2003]. *Evaluating an adaptive computer system for teaching about decimals: Two case studies.* In *AI-ED2003 Supplementary Proceedings of the 11th International Conference on Artificial Intelligence in Education*, pages 454–460. Citeseer.

Stacey, Kaye, E Sonenberg, Ann Nicholson, Tal Boneh, and Vicki Steinle [2003]. *A teaching model exploiting cognitive conflict driven by a Bayesian network.* In *International Conference on User Modeling*, pages 352–362. Springer.

Stern, David H, Ralf Herbrich, and Thore Graepel [2009]. *Matchbox: large scale online bayesian recommendations.* In *Proceedings of the 18th international conference on World wide web*, pages 111–120. ACM, New York, NY, USA.

Sutton, Richard S and Andrew G Barto [2018]. *Reinforcement learning: Second Edition.* MIT Press.

Sutton, Richard S, David A McAllester, Satinder P Singh, and Yishay Mansour [2000]. *Policy gradient methods for reinforcement learning with function approximation.* In *Advances in neural information processing systems*, pages 1057–1063.

Taraghi, Behnam, Martin Ebner, Anna Saranti, and Martin Schön [2014a]. *On using markov chain to evidence the learning structures and difficulty levels of one digit multiplication.* In *Proceedings of the Fourth International Conference on Learning Analytics And Knowledge*, pages 68–72. ACM.

Taraghi, Behnam, Matthias Frey, Anna Saranti, Martin Ebner, Vinzent Müller, and Arndt Großmann [2014b]. *Determining the causing factors of errors for multiplication problems.* In *European Summit on Immersive Education*, pages 27–38. Springer.

Taraghi, Behnam, Anna Saranti, Martin Ebner, Vinzent Mueller, and Arndt Grossmann [2015]. *Towards a Learning-Aware Application Guided by Hierarchical Classification of Learner Profiles. J. UCS*, 21(1), pages 93–109.

Taraghi, Behnam, Anna Saranti, Martin Ebner, and Martin Schön [2014c]. *Markov chain and classification of difficulty levels enhances the learning path in one digit multiplication.* In *International Conference on Learning and Collaboration Technologies*, pages 322–333. Springer.

Trask, Andrew [2017]. *Grokking deep learning.* Manning.

Van Hasselt, Hado, Arthur Guez, and David Silver [2016]. *Deep reinforcement learning with double q-learning.* In *Thirtieth AAAI Conference on Artificial Intelligence.*

Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas [2015]. *Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581.*

Xenos, Michalis [2004]. *Prediction and assessment of student behaviour in open and distance education in computers using Bayesian networks. Computers & Education*, 43(4), pages 345–359.

Zapata-Rivera, Juan-Diego and Jim E Greer [2004]. *Interacting with inspectable bayesian student models. International Journal of Artificial Intelligence in Education*, 14(2), pages 127–163.

Zhang, Ye and Byron Wallace [2015]. *A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. arXiv preprint arXiv:1510.03820.*

Zhao, Jun, Guang Qiu, Ziyu Guan, Wei Zhao, and Xiaofei He [2018]. *Deep reinforcement learning for sponsored search real-time bidding.* In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1021–1030. ACM.

Zhao, Zhong-Qiu, Peng Zheng, Shou-tao Xu, and Xindong Wu [2019]. *Object detection with deep learning: A review. IEEE transactions on neural networks and learning systems.*

Zheng, A. and A. Casari [2018]. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media.

Zhu, Meixin, Xuesong Wang, and Yinhai Wang [2018]. *Human-like autonomous car-following model with deep reinforcement learning. Transportation Research Part C: Emerging Technologies*, 97, pages 348–368.

List of Figures

List of Tables