Turkish Journal of Computer and Mathematics Education Vol.12 No.13(2021), 2339-2342

Research Article

Development of an Personalized Adaptive Learning Systemto improve students learning performances

Pratik Jadhav[a], Sachin Patil[b]

Rajarambapu Institute of Technology, Rajaramnagar, Uran Islampur, Maharashtra, 415414

Article History: Received: 11 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 4 June 2021

Abstract: Computer-assisted education promises open access to top-class instruction and a reduction in the cost of learning. Also, the modern education system has achieved great success in knowledge education, but its role in quality education is unclear. In the traditional one-on-many instructional approach, learning models and continuing education programs are only based on a uniform approach. Due to the limitation of factors like a teaching process, human power, learner's abilities, and so on, instructors only provide all students with the same materials, and they tend to use an identical teaching approach and the same rate of progress. In this approach, it is difficult to consider the learning needs of individual students. This implies that students dependent on instructors teaching skills so they cannot make effective use of class time to learn. On the other hand, students with insufficient proficiency levels may not be able to understand the content of the course.

Researchers are developing adaptive techniques that provide a better educational experience for students in some ways. Researchers offer accurate and perso-nalized content to students in an intelligent way, that may allow for adjustments in course content based on student's most recent performances. This technique allows the student to skip unnecessary learning activities by providing automated and personalized support for the student. These systems provide personalized course units that meet different student's educational needs.

Keywords: Adaptive learning, Deep knowledge tracing, Knowledge modeling

1. Introduction

Supervised training of a neural network requires a dataset with examples that already contains the expected answers. With this in mind, this work uses the examples contained in the latest version of the public dataset called "ASSISTments Skill-builder data 2009-2010" and available free on the web (Feng, Heffernan and Koedinger, 2009; ASSISTments, 2010; Piech et al., 2015). In Table below is defined some characteristics of this dataset

Table 1 - Characteristics of the dataset used in this project.

Characteristic	**Total**
Answered Questions	401.756
Types of questions	26.688
Types of knowledge components	124
Students	4.217

This dataset consists of more than 400,000 answered binary questions in which most of them are already classified in one knowledge component. Of all the attributes available in this dataset, only three were selected to be used in this project: the user identifier (ID), which is the student ID; the ID of the question knowledge component; and a binary variable that indicates whether the question was answered correctly or not by the student. All other attributes have been discarded and the questions that are not classified into a knowledge component were removed from the dataset.

2. Algorithms and Techniques

LSTM networks are a type of Recurrent Neural Networks (RNN) that takes into consideration the dependencies and relationship between the elements in time series. As the probability of a student answering correctly a question B may depend if the previous question was answered correctly or not, a LSTM network can be a possible solution to the problem of this project. Therefore, in this project we'll build a LSTM network to find the relationship between the problems in the dataset and predict the probabilities of a student answering a future question correctly. In Figure it's possible to see the architecture of the proposed model.

To build the model, it was used Keras and because of this tool we've chosen a Masking Layer to be the first layer in the model. This layer is responsible for taking care of the masking value which is used for padding the sequences and fill in incomplete batches. As input, this layer will receive a batch of 20 sequences of the same size (this size can vary in each batch) containing 246 features. The next layer is a LSTM Layer composed of 250 units. This layer is responsible for finding the relationship between the problems in a time series.

3. Work Design

The theoretical workflow defined in this project for approaching the proposed solution for the given problem can be seen in the figure . This workflow is composed of six main steps that should be done and result in a good performance before ending. The architecture of the LSTM network and the parameters defined for training should be changed in a way that results in a better performance in each try.

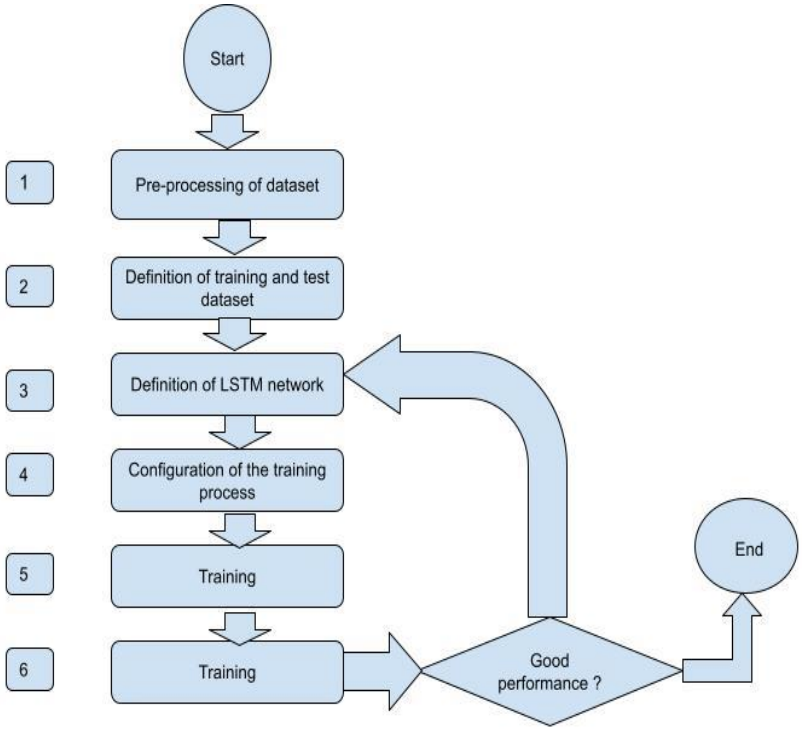

Figure 1 – Defined workflow for approaching a solution

In the first step, examples that do not contain a defined knowledge component are removed from the dataset. Then, the samples are grouped by student identification and padded with a constant value of -1 to normalize the sequences size. After that, a one-hot encoding is applied on the knowledge component ID of each question in the sequence to transform its integer value into vectors of the same size.

In the second step, the dataset is divided into two sets where 70% of the data is reserved for training and the remainder (30%) for testing. In the third step the architecture and parameters of the LSTM network are defined, some of them are: the number of LSTM layers and units; the weights initialization method; the activation function; the loss function; the optimizer; and the regularization techniques used to avoid overfitting.

In the fourth step, the number of epochs and minibatches are defined for training. In the fifth step the model is trained and in the last step it is evaluated and compared with related works. If the model does not have a good performance, all the steps from step 3 are repeated until a good performance is achieved

4. Implementation

As it could be seen on section 2, in the implementation of this project it was used Keras together with Tensorflow backend. Keras made the implementation a lot easier than just using Tensorflow by its own. The Tensorflow backend was chosen because the model was trained using the GPU on a Windows computer and it does this job out of the box. If the computer does not have a supported GPU to be used, then the Tensorflow that uses the CPU can be installed. All the environment setup used to implement this project can be found on the "Requirements" directory and the instructions to prepare your GPU can be found on Tensorflow documentation [7]. It was implemented two versions of the model, one to be using on a Jupyter Notebook and another one to be

used on a command line. Feel free to use any you want the results will be the same. If your computer does not have sufficient resources to build this model, it's possible to reduce the batch size and the number of epochs. I could not use my personal computer to build the model due to low memory available and I had to allocate a virtual machine on cloud with a Tesla K80 GPU to be able to build and train it. The model was training for 10 epochs with a batch size of 20. It took more or less 5 minutes per epoch and 50 minutes to finish. To be able to inject the input data into the model I had to create a custom data generator. This data generator is a class responsible for injecting the batches of data into the model and do some pre- processing that cannot be done on the whole dataset at once due to memory constrains. The logic used to fill up the incomplete batches, pad sequences and encode the skill id together with the label can all be seen on its source code, which can be found on the "StudentModel.py" file. To build the model I've created a class that encapsulates all of its functions in one place and make it clear to the user. This class is responsible for: creating and compiling the model; record the metric logs; save and load the network's weights; training, validation and evaluation. The main parameters it expects in its constructor and functions are: the parameters to build the model architecture; the validation, testing and training data generators; the number of epochs for training; the metrics to be used on evaluation; the weights to load, if available; and the files to save the model and logs. This class is defined together with the Data Generator class. To save the model weights and the metrics into a txt file, it was used two classes provided by Keras that does these jobs out of the box. These classes are defined in the parameters of the model's training function as callbacks. More information can be found on its documentation [6]. Due to the peculiarities of the elected problem, to be able to calculate performance metrics in the validation data at the end of each epoch during the training process, it was necessary to customize a callback class provided by Keras. To sum up, this customized callback class will compute the defined performance metrics using the validation data when an epoch ends and will store it on the model logs to another callback class save it in a txt file. More details about this implementation can be found on the same file as the previous two classes. As it can be seen on the mentioned file, the function used for model's evaluation and prediction are defined separately because it is reused in more than one place, not just in the model by itself but in the callback class that calculate the validation metrics. Other two functions that are responsible for splitting the dataset and applying some pre-processing techniques on the data can be found on the "Utils.py" file. The main difficulties faced during the implementation of this project was due to the complexity of LSTM Networks, which is not quite easy to understand, and the needed of customization of some functions due to the peculiarities of the problem itself. Because the model predicts the answer probabilities on all 123 skills, I had to customize the loss function to be able to calculate the loss only on the target skill ids (the problems solved by the student at that moment). It was not an easy task to understand how to do this. Also, I had to customize some other functions to calculate performance metrics, inject the data into the model and record log. To sum up, a lot of customizations were needed

5. Results

It was made an exhaustive job to refine the parameters. Each parameter was refined after the model being trained and evaluated. Therefore, this process took a long time to be done and this is reason I could not use an extensive GridSearch to do this task. In Figure it's possible to see the validation and training loss and the validation AUC in each epoch of training using the mentioned model's configuration.

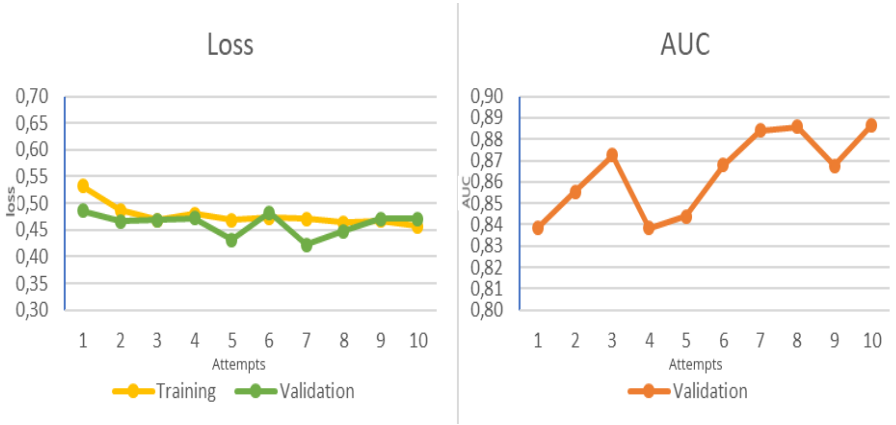

Figure 2 - Evolution of the final model during each epoch.

As it's possible to see in the results, the model did a great job finding the relationship between the problems on the dataset to accomplish the task. Comparing this results with the PFA and BKT approaches that can be seen on section 2, the proposed model is considerably better. Also, its AUC on the testing dataset is similar to the DKT model on related works.

Table 2 - Final results

Best Epoch Results	7
Best Validation Loss	0,42
Best Validation AUC	0,88
Evaluation AUC	0,85

The model was submitted to unseen data during both validation and evaluation of its performance on the test data. Analyzing the results, it's possible to conclude that the model is very robust because in both validation and testing the resulting AUC was very good. The model weights were saved and can be reused to validate the mentioned results. It can be found inside the 9th attempt folder in the "Log" directory. Therefore, the results can be trusted and validated by anyone

References

1. Gwo-Jen Hwang a, Han-Yu Sung b, Shao-Chen Chang, Xing-Ci HuangA: fuzzy expert sys-tem-based adaptive learning approach to improving students' learning performances by con-sidering affective and cognitive factors. Elsevier - Computers and Education: Artificial In-telligence.1 (2020) 100003
2. Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli: Deep knowledge trac-ing(2012)
3. Mohamed Boussakssoua, Bader Hssinab , Mohammed Erittal: Towards an Adaptive E-learning System Based on Q-Learning Algorithm. International Workshop on Web Search and Data Mining.(2020).
4. Shashank Sonkar, Andrew E. Waters,Andrew S. Lan, Phillip J. Grimaldi, Richard G. Bara-niuk.: Question-centricDeep Knowledge Tracing. From https://arxiv.org/abs/2005.12442(2020)
5. Dong Liu, Huanhuan Dai, Yunping Zhang, Qinpeng Li, Congpin Zhang.:Deep Knowledge Tracking based on Attention Mechanism for Student Performance Prediction. IEEE 2nd In-ternational Conference on Computer Science and Educational Informatization(2020)
6. Ibrahim Alkore Alshalabi, Samir E. Hamada, Khaled Elleithy, Ioana Badara, Saeid Mosleh-pour.: Automated Adaptive Mobile Learning System using Shortest Path Algorithm and Learning Style. International Journal of Interactive Mobile Technologies (2016)
7. Peter Brusilovsky: Developing adaptive educational hypermedia system: From design models to authoring tools. International Conference on User Modeling.(2004).
8. Chun-Hui Wu, You-Shyang Chen, Ta-Cheng Chen.An.: Adaptive e-Learning System for Enhancing Learning Performance: Based on Dynamic Scaffolding Theory. EURASIA Journal of Mathematics, Science and Technology Education (2017)
9. Murray, M. C., & Pérez, J.: Informing and Performing: A Study Comparing Adaptive Learning to Traditional Learning. Informing Science: the International Journal of an Emerging Transdiscipline.(2015)
10. ALOSI Homepage,http://www.alosilabs.org2020.