Machine Learning Engineer Nanodegree

Capstone Project

Lucas C. Casagrande
February 14th, 2018

I. Definition

Project Overview

The high dropout rate in universities during the first years has been challenging the universities to improve their teaching quality in order to keep their students motivated.

In this context, professors have fundamental importance in the quality of teaching and in the motivation of their students. Effective academic advising is one of the most effective ways for helping students to complete their degree in the expected timeline. Therefore, it is very important that professors have accurate and rapid evaluations of their students to make it possible to adapt their pedagogy to meet the individualities and difficulties of each student.

With this in mind, the present work proposes the use of a LSTM Network to model student's knowledge based on a set of historical problems answered and classified in their domain of knowledge. With this network it'd be possible to predict the probabilities of a student correctly answering a set of future. Based on this information, a professor can identify students at risk of failure and take the necessary precautions in time to improve student success on the class, reducing the rate of evasion.

Problem Statement

Classical evaluation methods such as tests and exams only permit the evaluation of a student after the test has being taken. Therefore, those methods do not allow at the beginning of the course a complete beforehand evaluation of all the expected abilities that a student should already have or learn during the course.

In other hand, predictive models are capable of predicting future information based on historical data. The data collected from a Learning Management System (LMS) can serve as the basis for the training of a model capable of predicting whether a student has sufficient knowledge to answer problems not yet seen, a problem known as Knowledge Tracing (KT).

Therefore, a LSTM network can be trained to find the dependencies and relationship between the solved problems in a dataset and use this to predict the probabilities that a student will correctly answer an exam problem not yet seen by him.

To sum up, the strategy adopted to solve the elected problem consist of four main steps:

1. Download of a large dataset containing solved problems already classified in its knowledge component.

2. Pre-process the data and transform the input to the network expected format.
3. Build and train a LSTM network to predict the probabilities of a student correctly answering a future problem.
4. Evaluate the model and refine it.

Metrics

As related works [1]–[3] computes the Area Under the Receiver Operating Characteristic Curve (AUC) to evaluate their models, this project will use the same metric to allow comparison of its performance. This metric reflects the model's ability to discriminate correct from incorrect responses, where a score of 1.0 reflects a perfect discrimination.

II. Analysis

Data Exploration

Supervised training of a neural network requires a dataset with examples that already contains the expected labels. Therefore, this project uses the examples contained in the most recent version of the public dataset called "ASSISTments Skill-builder data 2009-2010" [4], [5]. Table 1 shows some statistics about this dataset.

Table 1 – Dataset statistics

	Total
Answered Problems	525.534
Students	4.217
Attributes	30
Types of Problems	26.688
Types of Knowledge Components	123

This dataset has more than 500.000 binary problems (where the label is 1, correct, or 0, incorrect) where most of them are already classified in one type of knowledge component. Among all the available attributes, the knowledge component is the most important feature to the model. It will use it to find the relationship and dependencies between the problems when training. Therefore, we need to be sure to remove those problems that have a missing value in this attribute.

By analyzing the dataset statistics, it's possible to see that we have 30 different attributes available. Among all these attributes, only three were selected to be used: the user identifier (called 'user_id'); the ID of the problem's knowledge component (called 'skill_id'); and a binary variable (called 'correct') that indicates whether the problem was answered correctly or not by the student. In Table 2 it's possible to see a sample of data to better clarify the whole idea.

Table 2 – Data sample

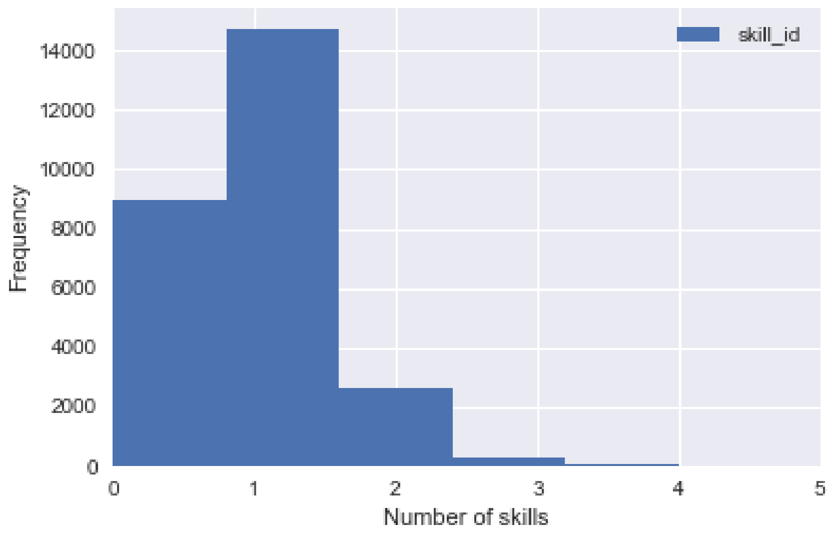
Index	user_id	skill_id	correct
0	64525	1	1
1	64525	1	1
2	64525	325	0
⋮	⋮	⋮	⋮
525.533	70363	1	0
525.534	70363	375	1

As the model will be trained to find the relationship between problems to predict a student's probabilities to respond correctly to a future problem, we need nothing more than these resources to build the proposed model.

Exploratory Visualization

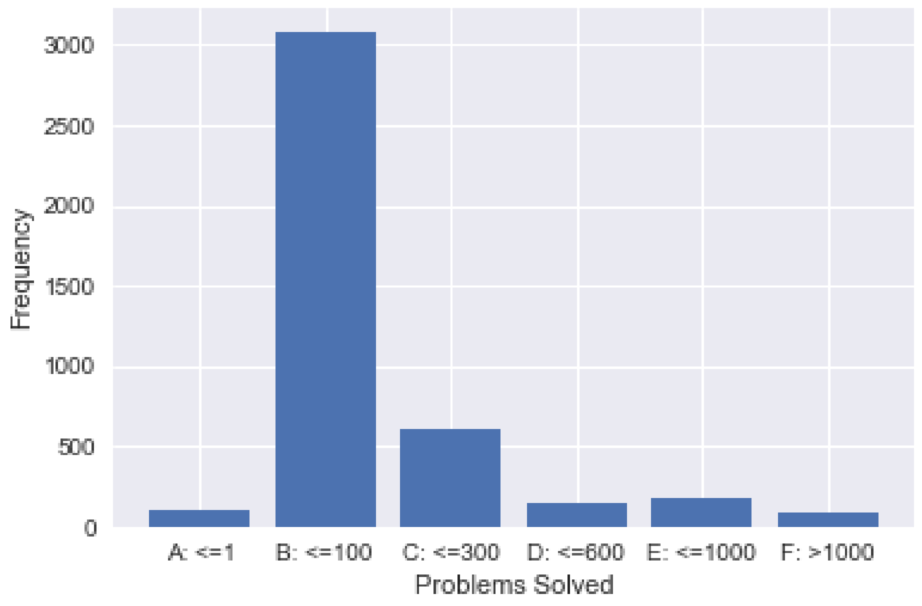
An important question to address in this dataset is the number of skills a problem can have. In Figure 1 it's possible to see that we've some problems that does not have any skill while others have one or more. We can adapt the model or the dataset to handle multi-skill problems but we cannot have problems without a skill tagged. Therefore, we need to remove all problems that does not have a skill from the dataset.

Figure 1- Frequency of problems per number of skills



Regarding multi-skill problems, to handle this in an easy way we can do two things: use the combination of the skills as a new joint skill; or separate the skills into multiple repeated response sequences with only one skill each. This dataset comes with the second approach already applied, for simplicity we'll stay with this approach.

Figure 2 - Frequency of students per number of problems solved



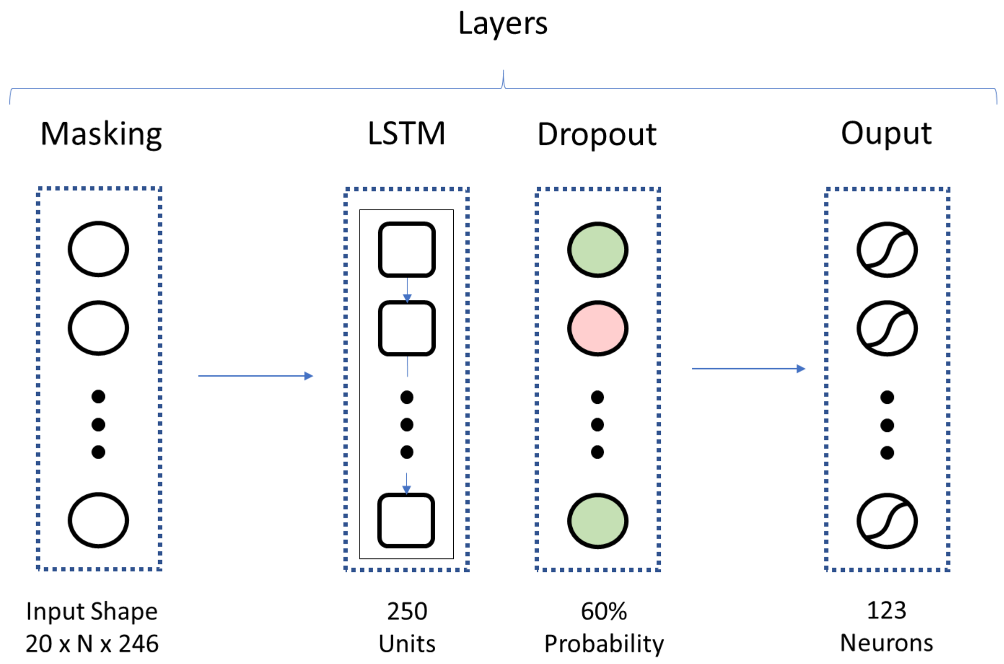
Another question that we need to address is the number of problems solved per student. In Figure 2 it's possible to see that we've some students who answered only one problem (category A) while the majority of students answered between 2 to 100 problems (category B). It is also possible to see that we have some students who answered more than 600 problems, which can cause us a headache due to insufficient memory. To deal with this, we can limit the time window manipulated by the algorithm to a sequence of problems of smaller size. For simplicity, we feed the data in the algorithm by batch, where each sequence of problems in a batch will be filled to have the same size.

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 3 it's possible to see the architecture of the proposed model.

To build the model, it was used Keras and because of this tool I'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.

Figure 3 – Proposed Model Architecture



The third layer is a Dropout Layer with a dropout rate of 60% to helps prevent overfitting. The last layer is a Dense Layer composed with 123 neurons using a sigmoid activation function. Therefore, the expected output will be the probabilities of the skills in the dataset being answered correctly or not by a student. The last two parameters chosen was the Binary Cross-Entropy to be loss function and the Adagrad to be the optimizer. All others parameters that can be set were not changed and its default values can be found on Keras documentation [6].

Table 3 - Example of the input encoding

user_id	Data Sample	Encode Skill and Answer	
	(skill_id; correct)	(skill_id * 2 + correct)	One-hot Encoding
64525	(0; 0)	0	[1,0,0,0,0,0, ..., 0]
64525	(0; 1)	1	[0,1,0,0,0,0, ..., 0]
64525	(1; 0)	2	[0,0,1,0,0,0, ..., 0]
64525	(1; 1)	3	[0,0,0,1,0,0, ..., 0]
70363	(2; 0)	4	[0,0,0,0,1,0, ..., 0]
70363	(2; 1)	5	[0,0,0,0,0,1, ..., 0]

As this type of network only accepts input data of the same size we'll need to apply a one-hot encoding on the "skill_ids" feature and pad the sequences of problems to the same size. To make the model sensitive to the skills who were answered correctly or not by a student we'll encode each skill id together with the label (the problem answer) before applying the one-hot encoding. Therefore, the input dimension will be 2 times the total number of skills, where we've skills ids who were answered incorrectly and skills ids who were answered correctly. The expected network's input is a sequence containing all the problems solved by a student encoded as it can be seen in Table 3.

Benchmark

Piech et al. (2015) was the first to propose the use of LSTM networks to solve the problem of Knowledge Tracking, which was then called Deep Knowledge Tracing (DKT). In the same context, Xiong et al. (2016) replicated the work of Piech et al. (2015) to explore his approach more deeply. In both papers it's possible to see a comparison between their approach and another one known as Bayesian Knowledge Tracing (BKT). In Xiong et al. (2016) paper it's also possible to see a comparison between DKT and Performance Factors Analysis (PFA).

Table 4 - Best AUC results obtained on ASSISTments dataset

	Method		
Paper	**DKT**	**PFA**	**BKT**
Xiong et al. (2016)	0.82	0.73	0.63
Piech et al. (2015)	0.86	-	0.69

As they use the same dataset to train their model we'll be using their results to compare the performance of the proposed model. A brief summary of the best AUC results obtained in their paper can be seen on Table 4. We'll be using it to compare the performance of our model.

III. Methodology

Data Preprocessing

Analyzing the data set, we've found that we'll need to do some pre-processing before training the model. Let's summarize what we've to do:

1. Remove the problems without a skill id.
2. Convert the data samples to sequences grouped by user id.
3. Convert the skill id values to a continuous variable starting from zero.
4. Split the data into three datasets (training, validation and testing).
5. Encode the skill id together with the label (problem answer).
6. Apply one-hot encoding.
7. Fill up incomplete batches.
8. Pad the sequences to the same size

During step 1, it was detected 66.326 samples that needs to be removed from the dataset. Step 2 generated 4.163 sequences while Step 3 rearranged the skill id to the continuous interval of [0,123). In Table 5 it's possible to see a summary of the dataset after these steps.

Table 5 - Summary of dataset after steps 1-3

	Total
Answered Problems	459.208
Students/Sequences	4.163
Types of Problems	17.751

In Step 4 we've split the generated sequences from previous step into three sequences: for training, validation and testing. It was reserved 20% of the dataset for testing and 20% of the remaining data for validation. In Table 6 it's possible to see a summary of each dataset.

Table 6 - Summary of each dataset

	Sequence Length
Training	2.665
Validation	666
Testing	832
Total	4.163

Steps 5-8 are done per batch due to memory constrains. In Step 7, incomplete batches are filled with a constant value of -1 and in Step 8 the sequences are filled with the same constant value accordantly with the maximum sequence size in its batch. One-hot encoding is not applied on Steps 7-8 because we'll use a Masking Layer on our model to take care of it.

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.

Refinement

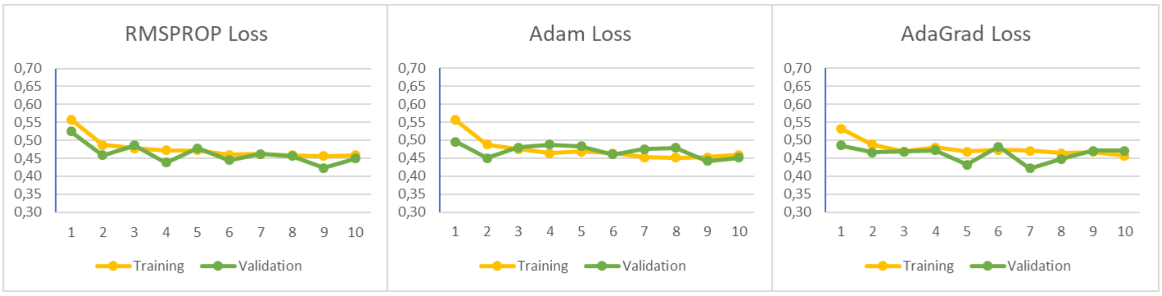
To better adjust the parameters of the model, I had to train and evaluate it a lot of times. The main parameters that were changed in the attempts are:

1. The optimizer.

2. The batch sizes.
3. The dropout rates.
4. The number of LSTM units.

It was tried three different optimizers: the RMSProp; Adam; and Adagrad. Three different batch sizes: 10; 20; and 32. Three different dropout rates: 60%; 30%; and 50%. And two different numbers of LSTM units: 250; and 150. My first attempt was with: the RMSProp; 20 sequences per batch; a dropout rate of 60%; and 250 LSTM units.

Figure 4 - Comparison between chosen optimizers



The results of each try were similar and one or another had minor gains on the validation loss. The logs of each try can be found on “Logs” directory, and a summary of them can be found on the excel file “Results.xlsx” that can be found in the same directory. The weights of the best model in each try can also be found on its respective directory inside the “Logs”. They can be used to validate my results and compare it with another parameter configurations.

Table 7 - Best attempt’s parameters

Optimizer	**AdaGrad**
LSTM Units	250
Batch Size	20
Dropout Rate	60%

In Figure 4 it’s possible to see a comparison of the loss on validation and training between each optimizer used in this project. In this comparison, only the optimizer was changed and other parameters remained the same. In Table 7 it’s possible to see the parameters configuration with the best results on the validation loss.

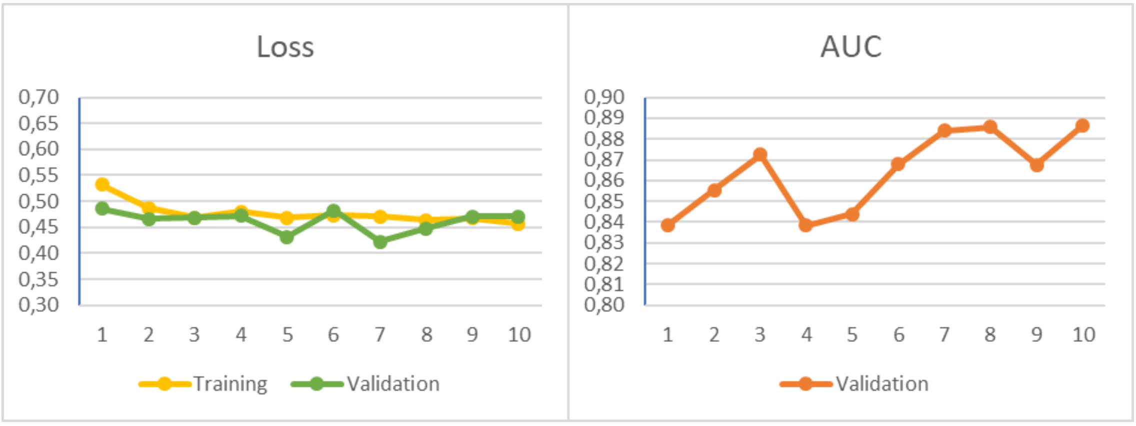
IV. Results

Model Evaluation and Validation

To define the best configuration of parameters and the best model it was chosen the model that had the best validation loss. Therefore, the attempt number 9 performed better than the others. In Table 7 it’s possible to see the parameters used in this attempt.

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 5 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. The final results of this model during validation and evaluation on testing data can be found on Table 8.

Figure 5 - 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 8 - 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.

An important aspect to note and to remember, is that only the weights with the best validation loss are saved. Therefore, in this case the model weights saved correspond to those achieved in the 7th epoch. The results of this epoch can be seen in both Figure 5 and Table 8.

Justification

In Table 9 It's possible to see a comparison between related works and the proposed model on this project, the metric used to compare is the AUC. As we can see, the proposed model was better than PFA and BKT methods and is similar to the implementation of DKT on related works.

Table 9 - Comparison of the AUC results between the proposed model and related works

	Methods		
	DKT	**PFA**	**BKT**
Xiong et al. (2016)	0.82	0.73	0.63
Piech et al. (2015)	0.86	-	0.69
Present Work	0.85		

Both implementations of DKT and this implementation had different AUC metrics. One of the main reasons for this can be due to the techniques used in the pre-processing of the input data and due some minor differences in each parameter used in the model, as the number of LSTM units and the number of epochs to train. The ratio of data used for testing and validation can also impact in the performance. It's hard to tell exactly what is the cause of this difference, we can only make assumptions as they are all different implementations.

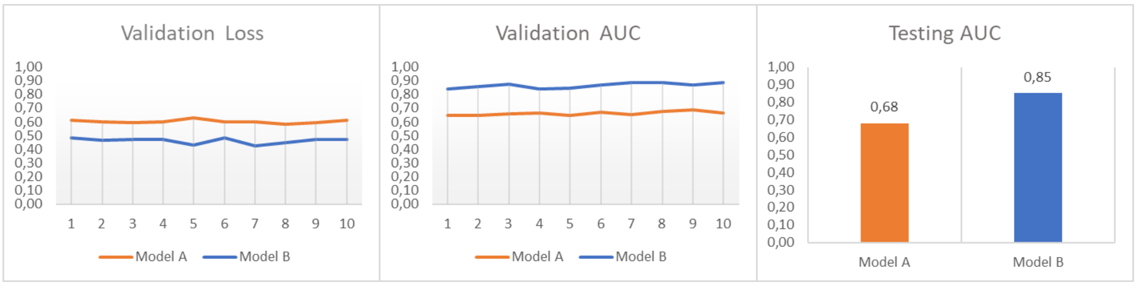
To sum up, the model did a very good job solving the problem. It can predict the probabilities of a student answering correctly a future problem with high credibility. One of the main difficulties of this approach is the complexity of training a LSTM network and finding the best suited parameters. If the model can scale is another question that needs to be addressed before deploying it to production.

V. Conclusion

Free-Form Visualization

A very important aspect that needs to be considered when solving the elected problem is the encoding of the problem skill id together with the label. If the feature does not take the label into consideration, the model will have difficulties to find the relationship between the problems and its expected label. It was done some experiments to prove this and in Figure 6 it is possible to see a comparison between a model trained without this mentioned encoding (model A) and the model proposed in this project (model B).

Figure 6 - Comparison between a model (A) trained without encoding the skill id together with the label and the proposed model (B).



As can be seen, the encoding of the skill id with the label is very important for the model when solving the problem of this project. All metrics are better when considering this encoding, which makes it clear about its relevance. Therefore, it is a very important aspect to consider when solving the problem of Knowledge Tracing using Artificial Neural Networks (ANN).

Reflection

In this project, we aimed to build a model capable of predicting the probabilities of a student answering correctly a future question. With this in mind, a LSTM network was trained and refined to better predict these probabilities. During the evaluation of the final model, an AUC of 0,85 and a Validation Loss of 0,42 was achieved. These results show that the solution proposed has potential on the field and could be deployed to production.

To help in the implementation of the model, Keras with Tensorflow backend was used to make it easier to implement. Tensorflow adds out of the box the possibility to train the model on GPU and was a very important tool in this project. In other hand, Keras facilitate the use of Tensorflow making it simple to use. Therefore, Keras with Tensorflow is the best of the two worlds.

To train the model it was used a dataset containing more than 400.000 questions that are already classified in their domain of knowledge. The knowledge component of a problem is the most important feature to the model as it will use it to find the dependencies and relationship between the problems. After finding the relation between the problems, the model can predict with high credibility the probability of a student answering correctly a not yet seen problem.

The whole dataset was split into three specific datasets: training, validation and testing. For testing, 20% of the data was used while 20% of the remaining data was used for validation. The process of training took more or less 5 minutes per epoch resulting in 50 minutes to train for 10 epochs.

After different attempts with different parameters values, the final model architecture consists of: a masking layer; a LSTM layer with 250 units; a dropout layer with the rate of 0.6; and an output layer with 123 neurons using sigmoid activation function. To optimize the model, AdaGrad was chosen in comparison with RMSProp and Adam optimizers.

As criterion for selecting the best parameter configuration and the best model between each epoch of training it was used the validation loss obtained at the end of each epoch. The AUC was than calculated using the testing dataset and its results explained on section 4.

To conclude, this project had challenged my knowledge and the model surpassed my expectations. The model's weights were saved and can be used to validate the results obtained in this project. Therefore, the results can be validated and the model reused in future works.

Improvement

For sure, more could be done to try to improve the final results obtained in this project. I could've used GRU (Gated Recurrent Unit) or GORU (Gated Orthogonal Recurrent Unit) in the place of the LSTM units, but as I don't know how exactly they work I've not used any of them. They can or not perform better, but we should implement it to see how it behaves. It's an interesting aspect to be considered in future works.

Another thing that could be done to improve the performance is explore more deeply the parameters configuration. Due to some resource limits in this project, it was not possible to apply an extensive GridSearch to find the best parameter values. Therefore, more studies could be done to find the impact of each parameter on the final results.

One very interesting aspect that could be done is the use of Attention Networks instead of a LSTM Network. Attention Networks are new and sounds very exciting. A comparison between each network would be nice and could be a great contribution to the community. As future works I would try to follow this path to improve the final results of this project.

REFERENCES

[1] C. Piech, J. Spencer, J. Huang, S. Ganguli, M. Sahami, L. Guibas, and J. Sohl-Dickstein, “Deep Knowledge Tracing,” in *Neural Information Processing Systems 28*, 2015, pp. 1–9.

[2] X. Xiong, S. Zhao, E. G. Van Inwegen, and J. E. Beck, “Going Deeper with Deep Knowledge Tracing,” in *9th International Conference on Educational Data Mining*, 2016, pp. 545–550.

[3] M. Khajah, R. V Lindsey, and M. C. Mozer, “How deep is knowledge tracing?,” pp. 1–8, 2016.

[4] M. Feng, N. Heffernan, and K. Koedinger, “Addressing the assessment challenge with an online system that tutors as it assesses,” *User Model. User-adapt. Interact.*, vol. 19, no. 3, pp. 243–266, 2009.

[5] ASSISTments, “Skill-builder data 2009-2010,” 2010. [Online]. Available: https://sites.google.com/site/assistmentsdata/home/assistment-2009-2010-data/skill-builder-data-2009-2010. [Accessed: 12-Feb-2017].

[6] F. Chollet, “Keras: Deep learning library for theano and tensorflow.” 2015.

[7] Google, “Tensorflow: An open-source software library for Machine Intelligence.” 2015.