Co-embeddings for Student Modeling in Virtual Learning Environments

Milagro Teruel
Universidad Nacional de Córdoba
Córdoba, Argentina
mteruel@unc.edu.ar

Laura Alonso Alemany
Universidad Nacional de Córdoba
Córdoba, Argentina
lauraalonsoalemany@unc.edu.ar

ABSTRACT

We present a neural architecture to model student behavior in virtual educational environments using purely unsupervised information. A crucial part of this architecture is the optimization of a joint embedding function to represent both students and course elements into a single shared space. This joint representation is more adequate than disjoint representations because it elicits insights on the relations between students and contents. Moreover, the model is trained only with interactions of the student with online learning platforms, without requiring any additional manual labeling by experts.

We obtain state-of-the-art results using this approach in two types of task: first, dropout prediction in online courses (MOOCs), and second Knowledge Tracing in Intelligent Tutoring Systems (ITS). We explore how the deep architecture is flexible enough to capture variables related to different phenomena, such as engagement or skill mastery.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Learning latent representations**; • **Applied computing** → *Interactive learning environments*; *E-learning*;

KEYWORDS

Deep Representation Learning, Student Modeling, Educational Data Mining, Dropout Prediction, Knowledge Tracing

ACM Reference Format:
Milagro Teruel and Laura Alonso Alemany. 2018. Co-embeddings for Student Modeling in Virtual Learning Environments. In *UMAP '18: 26th Conference on User Modeling, Adaptation and Personalization, July 8–11, 2018, Singapore, Singapore.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3209219.3209227

1 INTRODUCTION

Educational Data Mining (EDM) is a complex area, with hidden, unknown causes governing the behavior of students and the success or failure of courses. In this context, Student Modeling, a specialized area of user modeling, has been growing steadily in the past decade. The increased development of Massive Open Online Courses (MOOCs) and the new availability of Intelligent Tutoring Systems (ITS) allow researchers to gather large amounts of data. A clear example is the Carnegie Mellon University DataShop [1], that acts as a dataset repository and also provides standard learning analytics tools. These large datasets facilitate the application of machine learning and data science approaches, for example to address tasks like dropout prediction and prevention, or personalization to improve learning.

However, various factors are hindering advances in this area. First, the scarcity of manually labeled datasets freely available for research, which are costly to develop and to anonymize. Secondly, it is hard to obtain adequate abstractions, because the available datasets are composed of low-level logs generated by learning platforms. In this paper we try to address both these problems by applying unsupervised methods for representation learning. These methods can be applied to datasets without manual labels, and they provide generalizations over the low-level data that may be useful to attain human-level abstractions. These abstractions, although not directly interpretable, can be used in visualization, personalization, or to support decision-making in general.

Educational content has such a diversity that it becomes problematic to discover patterns and systematize it in a uniform manner. Different courses will have a completely different set of lessons and exercises, can be designed for other levels of engagement and work load, the units can be independent or highly correlated, among others.

Diversity is even more acute when we compare data from MOOCs with data from ITSs, where the learning environment and the organization of materials is quite different. For example, the content used by ITSs are selected and labeled with meta-information necessary for the recommendations produced by the system, which involves expert human annotation. The static format of MOOCs makes them less expensive to create and usually do not include such detailed meta-information.

We expect a general method based on deep architectures will be flexible enough to deal with different kinds of data and still preserve basic properties. In addition, neural networks have been proved to generalize well without the need of additional meta-information, as the models construct their own representations (embeddings) only from access traces.

[1]https://pslcdatashop.web.cmu.edu/about/

Therefore, to tackle the challenge of modeling students with low-level unlabeled data, we propose not only to model student actions but also course elements using neural embeddings. Embeddings will provide an abstraction and aggregation layer over the low-level data, which can then be used by domain experts to interpret student behavior. As neural models do not require an intensive effort of the experts to previously annotate the data, we can encompass unlabeled content developed with less resources.

Along with the modeling of students and course content through embeddings, we explore the impact of joint representations where both embeddings are in the same latent space. A shared space allows us to explicitly model properties of the relations between students and content, rather than letting the model infer them. Through a modification in the recurrent architecture, we can inject in the model with knowledge of the phenomena it is trying to discover, which can have a significant impact in tasks with few examples available.

In summary, the main contribution of this paper as the proposal to model student interactions with a **joint embedding of course elements and students**, using a recurrent neural architecture. We describe the general model and how to adapt it to solve two tasks: dropout prediction in MOOCs using the KDDCup 2015 competition dataset, and knowledge tracing in ITSs using the ASSISTments 2009-2010 dataset.

We compare the performance of the system with an architecture without embeddings and with disjoint embeddings. We include the state of the art and an upper bound of performance provided by a manually labeled dataset. Results show that, besides providing a conceptually more adequate representation of students and contents, embeddings provide an improvement in performance, and that co-embeddings perform at least as well as disjoint embeddings, and outperform them in some contexts, like smaller courses.

We also asses the impact of pretrained embeddings for course elements obtained with the word2vec unsupervised algorithm, using them as a starting point to train the final model. Results show that the pretraining improves performance only in some courses with few examples.

2 RELEVANT WORK

Representation of students and course elements has been usually carried out by explicitly incorporating expert domain knowledge into the data. For example, Bayesian Knowledge Tracing (BKT) [1], one of the most used methods for knowledge tracing, requires each exercise to be labeled with a set of skills. This type of annotated information is not usually available, for example in the case of MOOCs. The underlying representations for such approaches are strongly based on the manually added information, and as a consequence they can suffer from a theoretical bias. This implies that the main variation factors are set by the domain expert and not discovered by the model.

However, in recent years there has been an increase in unsupervised approaches to modeling students and course elements. These approaches are more flexible than supervised ones because they serve as general methods to obtain representations from any dataset. In other words, they are not based on aspects of specific courses or platforms that have to be adapted later, or that acquire different relevance in a different context. For example, Performance Factor Analysis (PFA), [8] and SPARse Factor Analysis (SPARFA) [5], show that factor analysis has a positive impact in knowledge tracing, discovering or refining the skills associated to each exercise. These works use a form of embedding based on variable correlation to discover a useful underlying representation of phenomena.

Deep learning architectures have also proven to be very helpful in discovering latent causes from observable phenomena. The multiple layers in a deep architecture build a number of internal representations of the input data, optimized for a certain prediction task. These internal representations, when the model is applied to tasks like dropout prediction, are liable to capture underlying causes like the diversity in student background and interests. In this context, we can train a model only to obtain these internal representations, called embeddings, which will then be used to represent examples for a different task. The task used to train embeddings is called a *pretext task*. When the model trained to obtain such embeddings is a neural network, we call them *neural embeddings*.

Neural models, and in particular deep neural models, have shown to be very effective for a wide range of problems, also for EDM. Deep Knowledge Tracing (DKT) [10] is a method that proposes to model student knowledge using the hidden state of a recurrent neural network. The pretext task for the model is to predict if the student will be able to solve the next exercise presented by a tutoring system. In contrast with BKT, it does not depend on manually assigned skills. Furthermore, it can be used to automatically detect relations between exercises and cluster them. The work of [13] models students also using RNNs trained to predict the next action to be performed by the student. This approach does not require exercises to be graded, as is often the case in MOOCs.

In addition to representing student state with neural embeddings, our proposal is to embed course elements in the same space. This implies learning a joint representation for both students and course contents. Co-embeddings have been successfully used before in MOOC environments [11]. In this study, the authors propose an embedded representations of students in a “latent skill space" that can be interpreted as the knowledge level of the student on each skill. Along with them, they find embeddings for course elements in the same skill space. The pretext task used to obtain these embeddings was performance prediction. Our approach differs from [11] in the use of a neural architecture to obtain the embeddings, and in the pretext task selected for optimization.

Modeling students and course elements in the same space is convenient in two aspects. On the one hand, it allows us to model better the relations between them using distances in the shared space. On the other hand, it opens the door for possible joint visualization and interpretation, leading to better insights on the causes of a certain phenomenon, for example dropout. A study of 2017 [7] shows how visualizations of student states can be interpreted and contribute to the work of course designers.

Dropout prediction is a task that differs from Knowledge Tracing in several aspects. One of the most important ones is the lack of a consensus about the definition of dropout. For example, some lines of work focus on the prediction of student behavior at the end of the course, while others seek to prevent dropout and focus on the prediction of actions in the near future [14]. It is also worth noticing that the factors involved in dropout prediction are not throughly understood. In [4] an analysis is done on the difference of background and expectations from students, which leads to diverse levels of engagement. We expect a deep learning model can capture this type of variability, represent it with an adequate level of abstraction, and help course creators understand it.

Some neural approaches have been applied to dropout prediction, for example [2], who reports an increase in performance compared to Markov models, logistic regressions and support vector machines. However, instead of using the full sequence of interactions, the data is aggregated in weekly periods.

3 NEURAL CO-EMBEDDINGS FOR STUDENT STATES AND COURSE ELEMENTS

As presented in previous sections, our aim is to obtain a representation of students and course elements that highlights the major factors involved in the human learning experience. To do that, we learn how to project students and course elements to a shared space, bringing forth their most indicative aspects and the relations between them. In this section we describe the concrete computational architecture to obtain the neural co-embeddings that represent students and course elements.

Our goal is to find the embedding functions φ_E and φ_S that project course elements and students from their representation in log data into a shared space, respectively. In the shared space each course element is represented as a vector. Students, on the other hand, are continuously changing as they progress through the course. As a result, they are represented as a sequence of vectors, corresponding to a given point of time. Individual vectors represent the state of the student after each interaction with a course element.

To find this shared space and the functions embedding students and course elements in this space, we propose a recurrent neural architecture to optimize φ_S and φ_E, shown in Figure 1. The base of this architecture is a Recurrent Neural Network (RNN), which is designed to selectively *remember* some aspect of previously seen input. The network is trained with some identifier (ids) of the course elements seen by students and possibly the outcome of the interaction. The output of the network will depend on the pretext taks used for the optimization. The difference between the output and the true labels is used to calculate the loss of the model over all examples, which is minimized using the BackPropagation Through Time algorithm (BPPT), until the model converges.

3.1 Integrating embeddings in the RNN architecture

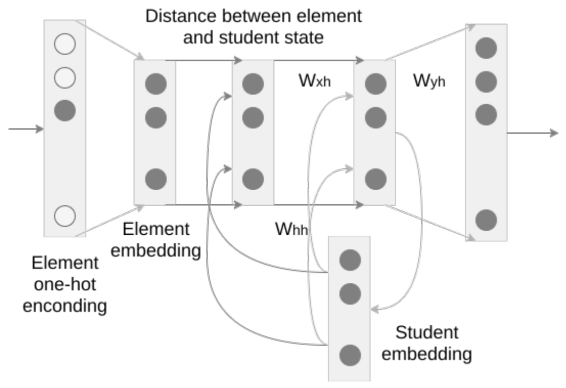
A basic RNN structure has tree layers: input, hidden state and output. What characterizes a RNN is that the new hidden state h_t is calculated using the information from the input layer x_t and from the previous hidden state h_{t-1}, where t represents the position of the interaction in the sequence. Following, the output o_t is calculated as a transformation of the hidden state h_t.

The specific update equations for a basic RNN are the following:

$$y^{(t)} = \sigma(W^{(hy)}h^{(t)} + b^{(y)})$$
$$h_t = tanh(W^{(xh)}x^{(t)} + W^{(hh)}h^{(t-1)} + b^{(h)}) \quad (1)$$

Inspired by [10], we will interpret the hidden state after t interactions (h_t) of the RNN classifier as the embedding of the student at

Figure 1: RNN architecture with co-embeddings.


Distance between element and student state
Wxh
Wyh
Element embedding
Whh
Element one-hot enconding
Student embedding

time $T + 1$. Once trained, the network has learned which information from the input is useful to keep during time, and a codification of such input is recorded in the hidden state. This hidden state is our function φ_S that encodes the student states.

To integrate the embedding of course elements φ_E into the RNN classifier, we add a new embedding layer between the input layer and the recurrent layer. An embedding layer is nothing more than a matrix that works as a look-up table, where the column j is the embedding of the element with index j.

At this point we have an architecture that projects students and course elements to a space, but we haven't ensured yet that this space is shared. In other words, the dimensions of this space do not necessarily have the same meaning for students and for course elements. To ensure that both embeddings are indeed in the same space, we will calculate the new hidden state as a combination of the previous state and a *point-wise function between the student current embedding and the embedding of the course element* in the current interaction. An illustration of these architecture is depicted in Figure 1

The relation between the course element embedding and the student embedding will have different impacts depending on the pretext task. For example, in KT, the dimensions of the latent space are interpreted as the mastery level of the student in a given skill. Then, the course element embedding can be seen as a prerequisite vector, as in [11]. If students are close enough to the course element, then they will get the major possible gain of interacting with the course element. If they are too far, then the lesson content will be either too easy or too hard.

The point-wise aspect of the function is also important, as we want to distinguish the influence of each dimension in the resulting state. This is an important distinction between our model and [11], which uses only the length of projection of the student embedding over the lesson embedding. As a result, our approach does not compensate a small value on a dimension (seen as skill level) with a large value in other one, but preserves the relation between student and course element for every dimension.

After introducing these changes, Equation 1 is rewritten as:

$$h_t = tanh(W^{(xh)}\delta(\varphi_E(x^{(t)}), h^{(t-1)}) + W^{(hh)}h^{(t-1)} + b^{(h)})$$

where δ is the pointwise function measure. We propose several δ functions, some examples are: $\delta(x, y) = (x - y)^2$, $\delta(x, y) = |(x - y)|$ and $\delta(x, y) = norm(x - y, 0, std)$ (all operations are pointwise). In the last function, *norm* refers to the probability of the vector following a normal distribution centered in 0 and with standard deviation *std*. The value of *std* is originally fixed, but it can be also optimized with the network.

More sophisticated networks can be used instead of a vanilla RNN, like Long Short Term Memory (LSTM) or Gated Recurrent Unit (GRU) networks. Their variation is not in the layers but in the neurons, so they can be trained to produce co-embeddings as long as the input vector to the recurrent layer is modified with the δ function.

3.2 Neural co-embeddings for Knowledge Tracing

For the task of KT, the RNN network will be used to produce one output for each element of the sequence, in a configuration known as *sequence labeling*. Just as in DKT, our model estimates the probability of success for every exercise, given all the previous interactions of the student. In consequence, the model's output layer has one neuron for each possible exercise. However, only the probability for the exercise that the student actually faced on the next time step is used to optimize the classifier, as we do not know what the performance could have been in other exercises. The output layer does not have a softmax activation, but rather a sigmoid one to normalize each individual neuron.

The loss function selected is the mean binary cross entropy or log loss between the model prediction and the true label over all examples. The loss for a student is sum of the cross entropy for all the interactions in the sequence. If we define the exercise index in the interaction at time t as e_t, the loss for a student is:

$$loss = \sum_t log(y_{t,e_t})o_t + log(1 - y_{t,e_t})(1 - o_t) \qquad (2)$$

The representation of the input data is also modified with respect to the base model. This task provides two values for each interaction: the id of the exercise and if the student solves it successfully on the first attempt. To model both aspects, we use the sum of two embeddings for each exercise: one that corresponds to the base exercise, and one for the successful outcome of the interaction. If the student does not solve the exercise in the first try, only the base embedding is used.

3.3 Neural co-embeddings for dropout prediction

Neural embeddings and co-embeddings can be useful as well for less defined tasks like dropout prediction. In this case, we do not expect them to model skill mastery or knowledge, but mainly other variables, like engagement.

It is important to note that, from a classification point of view, this task is very different from KT. For every sequence, a single output is expected instead of one for each time step. This scenario is usually called sequence classification. As a result, the vanishing gradient problem associated with RNNs can be more problematic: we need to propagate a very weak signal of error up to the very first interaction of the sequence, which can be many steps away from the training label.

There are two possible configurations in a neural network to predict a binary variable like dropout: to output only the probability of the positive layer, or to output the probability of both classes as a multilabel classification task. This affects the size of the output layer, as one neuron is needed per output class. The loss function is in both cases the average of the cross entropy.

3.4 Pre-trained embeddings

A possible variation in this model is to initialize the embedding layer with pre-trained course element embeddings, obtained with an unsupervised method.

Pre-trained embeddings have the advantage of including information about the order in which students access course elements, using algorithms specialized for this kind of task. Indeed, methods like word2vec [6] and GloVe [9] have been shown to adequately model the underlying distribution of sequences of events with incomplete samples. These models seem to capture latent causes, like word semantics in language modeling. We expect pre-trained models will capture aspects of the datasets pertinent to the relation between course elements independently of their impact in the pretext task. In [7], the author modeled student sequences using only word2vec and analyzed the usefulness of visualizations from those embeddings, rated by course creators. Results show that high-level organization of the course content was captured by the embeddings, and they even clustered successful and unsuccessful students.

Additionally, these embeddings have the possibility to be optimized or fine-tuned along with the optimization of the model for the pretext task. If we do not allow fine-tuning, the impact of the pre-trained embeddings on the student embedding will be higher. Another important advantage is that pre-trained embeddings, as an unsupervised method, can also be trained with instances from the same domain but without labels for the pretext task.

4 EXPERIMENTAL SETTING

4.1 Datasets

The KDDCup 2015 competition [2] proposed predicting the dropout of students in MOOCs. The data was provided by XuetangX, a Chinese MOOC learning platform initiated by Tsinghua University, a partner of EdX.

Although the information is no longer available in the competition website, this dataset was, to our knowledge, one of the few freely available big datasets of detailed logging in a MOOC environment. The data provided logs of events like access to video content, resolution of a problem, etc., for 39 different courses. Events are timestamped and identified with the corresponding student and course.

For the competition the dropout event was defined as the absence of student activity in the ten days following the end of the course. With this definition, 79% of the students in this dataset (19072) are labeled as dropouts. The distribution of dropouts is similar across courses and it ranges between 70% and 90%.

[2] https://biendata.com/competition/kddcup2015/

One model was optimized for each course. We have found that courses with different numbers of students have very different results, therefore we decided to distinguish results in three different segments of courses: 5 big courses with more than 6000 training students, 9 medium courses with between 5000 and 2000 students and 24 small courses with less than 2000 students.

On the other hand, we explored the performance of our approach in the Knowledge Tracing task using the ASSISTments 2009-2010 dataset, described in [3]. This dataset is a typical example of the data generated by an ITS system. It also has other desirable properties: it contains a fairly big amount of student interactions, nearly 350000, and the class imbalance is not pronounced, with 65% of positive labels. Furthermore, it is a reference dataset for the field of EDM, as it has been used for experimentation by several works.

As noted by Xiong et al. [15], the interactions between a student and an exercise labeled with multiple skills are stored duplicating the row for the interaction and labeling each of them with a different skill. This leads to duplicated data and can affect the test performance of any classifier if there is leaked information between the training and testing dataset. Following [15], we represent exercises with multiple skills with a new skill that serves as the merge of the original ones.

To reduce the noise of the dataset, the interactions involving exercises that appear less than 5 times in total were deleted. It is a common technique, used for example by word2vec, as no meaningful embedding can be trained from such a small amount of instances.

For all experiments, the entire student sequences of actions in both datasets were divided in three portions, training (70%), testing (20%) and validation (10%). The performances reported are obtained by applying the trained model over the testing dataset, after the best hyperparameters have been chosen using the validation dataset.

4.2 Variations in classifiers and learned representations

The neural architectures we tested are all based on RNNs. The simplest one (**LSTM**) has a single layer of LSTM cells. The input for this model is the one-hot encoding representations of the course element id. The output of the recurrent layer is connected to a dropout layer, and later to a regular dense layer with sigmoid activation and L2 regularization. The output layer is composed of two neurons with softmax activation, one for each class.

The second model (**E-LSTM**) has the same structure as the LSTM model, but the input layer is replaced by an embedding layer. Since students and course elements are not forced to share the same space, we call this approach *disjoint embeddings*. It is followed by a dropout layer using the same ratio as the one after the recurrent layer.

As mentioned before, the embedding for the KT task is composed of two parts. There is a base embedding for each exercise. Later, another embedding is added if the student has successfully solved the input exercise. In dropout prediction, one embedding is created for the merged module id and event id of the element seen by the student. We experimented using only the module id, with consistently worse results.

The final model (**CoE-LSTM**) is the one described in section 3, implementing the proposed co-embeddings. It has the same dropout layers as the E-LSTM.

For the two embedded models, we also explored the use of pre-trained embeddings, as developed in 3.4. To obtain the embedding we used the training sequences as input to the word2vec SkipGram algorithm. The parameters used are: window size of 5 events, minimum frequency of 5 events, an alpha initial learning rate of 0.01 and a negative sampling of 5 examples. Both settings with and without fine-tuning are explored.

The hyperparamenters of the networks optimized included embedding size (20, 50, 100 and 200), hidden layer size (20, 50, 100 and 200), number of events in the sequence used (20, 50, 100, 200 and 300) and dropout ratio (0, 0.2, 0.3 and 0.5). The optimizer used is the Adam implementation of Tensorflow with a learning rate of 0.001. Other configurations were also tested: RNN and GRU cells; bidirectional networks; RMSEProp, SGD, Adagrad and Adaboost optimizers; and higher and lower learning rates. Results for these other parameters are not reported as the performance decreased.

In the case of dropout prediction, we report the results of the model with the better AUC score for each course.

4.3 Optimization

The algorithm used to optimize a recurrent neural architecture is called Back Propagation Through Time (BPTT). However, propagating the gradients over very long sequences of time can be a prolonged process, producing vanishing gradients for the upper most steps. LSTM networks are designed to avoid vanishing gradients, but in practice they also have a limited propagation point. A technique used to overcome this problem is truncating the gradients after certain amount of steps, leading to a Truncated BPTT (TBPTT).

In dropout prediction, TBPTT decreased the performance significantly. As a result, for this task we used only the last portion of the sequences to train the model.

4.4 Metrics

To assess performance of different approaches, we measure the difference between the predicted probability of dropout or success and what is actually found in the dataset. Note that the prediction is probabilistic, while the true label value is binary. The metrics used are the Area Under the ROC Curve (AUC), the reference metric in the KDDCup, the Root Mean Square Error (RSME) and the coefficient of determination R2. The R2 score [12] estimates the proximity of the performance of the classifier to a random classifier.

5 RESULTS

5.1 Dropout prediction

In table 1 we present the results for AUC, RMSE and R2 for the best performing models for dropout prediction. In general, we can see the three metrics increase with the use of embeddings, and raise even more with joint representations. This supports our initial hypothesis that co-embeddings capture the underlying causes of dropout better than other representations.

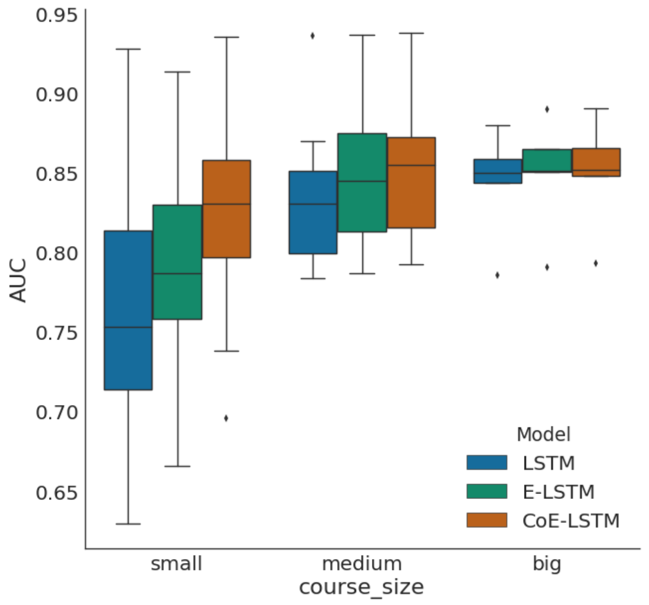
It is worth noticing that the winner team at KDDCup 2016 reached an AUC of 0.91 with an ensemble model. Those results

Table 1: Model performance for dropout prediction

Model	AUC	RMSE	R2
LSTM	0.831	0.333	0.331
E-LSTM	0.842	0.327	0.356
CoE-LSTM	0.852	0.322	0.368

were obtained with the official test dataset, to which we don't have access. In addition to that, the winning solution used all the information available, while we only use the identifier of each course element, in order to evaluate the models in a scenario of minimal information. In this work, we have not focused in performance, hyperparameter tuning or ensemble of different models, but in assessing the impact of embeddings with simpler models, and also assessing the utility of different kinds of embeddings.

Figure 2: AUC for dropout prediction, grouped by course size

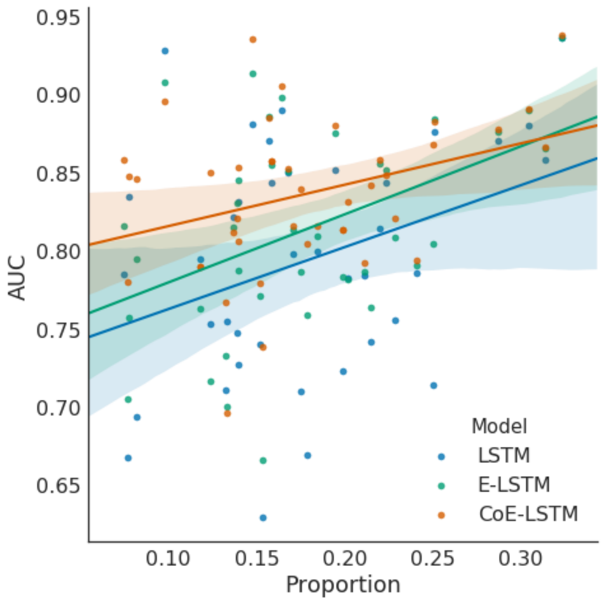


When we disaggregate courses by size, we can see that co-embeddings perform much better in smaller courses, even if the general performance in such courses is worse than for bigger courses. In Figure 2 we can see that for the biggest courses all classifiers perform indistinguishably, with small variations, hence the smaller spread of the boxplot. However, for smaller courses, where data is more scarce and performance is worse, co-embeddings seem to provide useful generalizations over the low-level data. It is noteworthy that, while co-embeddings do not seem to impact positively on the performance on bigger courses, they do not impact negatively either.

Exploring the differences in performance for different courses we found another factor of correlation. The proportion of students that dropped out, i.e. the class imbalance, impacts on the difficulty of the task. The less examples in one of the classes, the harder it is to learn the differences with the majority class. In figure 3 we plot the AUC values for each type of model in each course, according to the dropout rate in the test dataset. The lines represent regression models fitted to the data. They helps us to see that, in average, CoE-LSTM models have a greater impact on the performance over courses with more class imbalance.

For this task, embedding sizes of 50 and 20 are more successful. The number of steps used in training varies from 50 to 300, and they are inversely proportional to the size of the recurrent layer.

Figure 3: AUC for dropout prediction, in relation with the course no-dropout rate



5.2 Knowledge Tracing

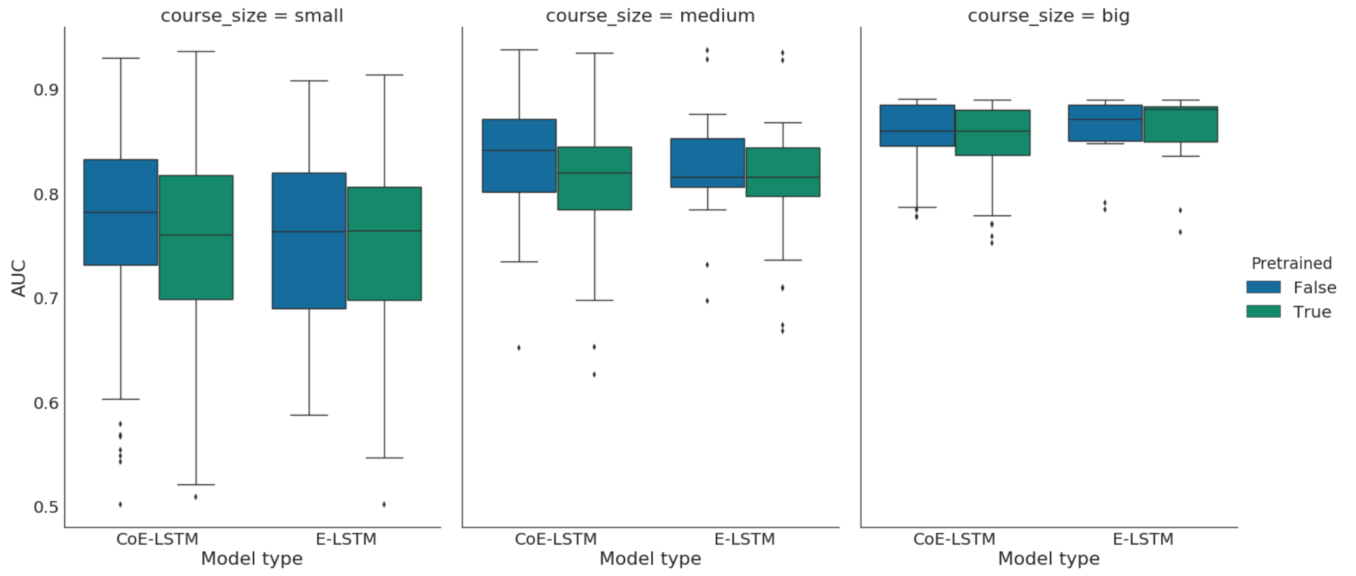
In table 2 we present the model performances for the Knowledge Tracing task. To facilitate comparison with the state of the art, we display the results reported by Xiong et al. [15] for the same dataset, marked with an asterisk. However, it must be noted that these results are not directly comparable because they don't use the same dataset for testing. In particular, the whole set of problems is used, instead of filtering less common problems as we do.

Table 2: Model performance for knowledge tracing

Model	Identifier	AUC	RMSE	R2
DKT*	Skill ID	0.75		
LSTM	Skill ID	0.746	0.432	0.176
LSTM	Problem ID	0.721	0.462	0.069
E-LSTM	Problem ID	0.740	0.443	0.131
CoE-LSTM	Problem ID	0.741	0.448	0.130

The wording in Xiong et al. [15] seems to indicate the method is using DKT with the skill ID. Indeed, when we use our LSTM with skill IDs (second row in the table), we obtain a comparable

Figure 4: Impact of pre-training embeddings for dropout prediction, grouped by model type and course size, reporting AUC.



performance. However, the original intention of Piech et al. [10] was to avoid using this manually added information, treating the problem with unsupervised information only, using the problem ID as input.

The performance of the LSTM classifier with skill IDs is presented as an upper bound of performance, a point of comparison of the model in the optimal case: when labeled information is present. We can see that using only unsupervised information (the problem ID), the performance of the embedded methods is comparable to the performance using manually labeled information (skill IDs). Results indicates that, with this setting, embedded models have a positive impact with respect to the basic LSTM network. The joint representation obtains slightly better results than the disjoint representation, but further exploration needs to be carried out to discover the causes for these differences in performance.

The selection of the δ function had a great impact on the co-embedding performance. The best results where obtained with $\delta(x, y) = tanh(x - y)$, and the absolute value of the difference displayed good results as well. Other possibilities, like a sigmoid or square function lowered the AUC up to 4 points below the results shown. Such impact suggest that this function is vital to model the relations between student state and course elements correctly. In the case of dropout prediction, the results are highly dependent on the course and not so drastically variable. In general the the square function and the normal function with a fixed 0.5 standard deviation were the best performing ones.

For the sake of reproducibility, it is worth reporting that the best hyperparameter configuration for KT is an embedding size of 50, a maximum number of steps in the TBPTT of 50, between 200 and 500 training epochs and a dropout rate of 0.3. This contrasts with results reported in previous work, where they use a recurrent layer size of 200 neurons. Note that embeddings for this task are the same size as embeddings for the task of dropout prediction, even if the dataset for each course was smaller.

5.3 Impact of pre-trained embeddings

Finally, we have compared the performance of embeddings trained together with the whole model, and embeddings pre-trained as explained in Section 3.4.

In Figure 4 we present the AUC scores of *all* experiments conducted with the KDDCup 2015 dataset, to compare the general performance of embedded methods with and without a pre-training step. We see that pre-trained embeddings are just as sensitive to course size as models without pre-training. Indeed, for bigger courses the performance is better, regardless of pre-training. In contrast, for medium and smaller courses, we can see that CoE-LSTM models perform slightly better without pre-training, while E-LSTM models are not remarkably affected by the use of pre-trained embeddings. Our strongest hypothesis to explain this difference is that co-embeddings are able to capture relevant information of the dataset more adequately than a pre-training step. This seems to advocate for a superiority of co-embeddings over disjoint embeddings, because the latter perform indistinguishably from pre-trained embeddings, thus they seem to be capturing roughly the same information, and they are both surpassed by co-embeddings.

For the case of Knowledge Tracing, the use of pre-trained embeddings had a smaller impact. Some δ functions performed better with pre-trained embeddings and others without, but the results varied in less than 0.01 points. One consistent result across all experiments is the improvement for fine-tuning the pre-trained embeddings while learning the pretext task.

The differences in the impact of pre-trained embeddings between dropout prediction and knowledge tracing may be due to the difference in the classification task, as described in Section 3.3. Dropout prediction is a sequence labeling task, while knowledge tracing predicts an output for each item in the sequence. The information needed to detect dropout is encoded in the entire sequence, not in the properties of the individual course elements. Methods like word2vec are not intended to characterize elements with respect to the entire sequence but rather in relation with their surrounding elements. It is coherent then that injecting that kind information does not help, and even hinders performance.

6 CONCLUSIONS

We have proposed a purely unsupervised approach to model student behavior in learning platforms with joint embeddings of course elements and students. The joint embeddings are obtained with a recurrent neural architecture is modified to directly model the relation between both types of embeddings, using knowledge of the task to be solved. We have evaluated and compared the architecture in two different tasks: dropout prediction and knowledge tracing.

Results indicate that co-embeddings are able to capture the latent causes involved in dropout, outperforming disjoint and not-embedded representations. This improvement in performance increases in courses with less students, and courses with higher dropout rate.

For the knowledge tracing task, results indicate that embedded representations reach state-of-the-art performance, even without labeled information used in previous work, like manual annotation of skills. For this task, disjoint representations do not outperform joint ones (co-embeddings), but the difference in performance is only within 0.001 point of AUC. However, we expect that joint embeddings will add value to other applications in Learning Analytics, like visualizations or other interpretation tools.

Indeed, this work has shown promising results from performance point of view, but there is still work to do on the effectiveness of the joint representation for interpretation of human learning. We are currently exploring this line of research through visualization techniques and recommendation of content (personalization).

Future work will include the application of these methods to other datasets from ITS and MOOC platforms to further evaluate how different configurations affect the obtained embeddings.

Another interesting line of future work is to further analyze the impact of pre-trained and possibly fine-tuned embeddings, with special attention to cases that suffer from data sparseness. The diversity of results obtained for different courses in the KDDCup dataset indicate there is a potential for the inclusion of this unsupervised information when the number of examples is limited.

REFERENCES

[1] Albert T. Corbett and John R. Anderson. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction* 4, 4 (1994), 253–278. https://doi.org/10.1007/BF01099821

[2] Mi Fei and Dit-Yan Yeung. 2015. Temporal models for predicting student dropout in massive open online courses. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on Data Mining*. IEEE, 256–263.

[3] Mingyu Feng, Neil Heffernan, and Kenneth Koedinger. 2009. Addressing the Assessment Challenge with an Online System That Tutors As It Assesses. *User Modeling and User-Adapted Interaction* 19, 3 (Aug. 2009), 243–266. https://doi.org/10.1007/s11257-009-9063-7

[4] René F Kizilcec, Chris Piech, and Emily Schneider. 2013. Deconstructing disengagement: analyzing learner subpopulations in massive open online courses. In *Proceedings of the third international conference on learning analytics and knowledge*. ACM, 170–179.

[5] Andrew S. Lan, Andrew E. Waters, Christoph Studer, and Richard G. Baraniuk. 2014. Sparse Factor Analysis for Learning and Content Analytics. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 1959–2008. http://dl.acm.org/citation.cfm?id=2627435.2670314

[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'13)*. Curran Associates Inc., USA, 3111–3119. http://dl.acm.org/citation.cfm?id=2999792.2999959

[7] Zachary A. Pardos and Lev Horodyskyj. 2017. Analysis of Student Behaviour in Habitable Worlds Using Continuous Representation Visualization. *CoRR* abs/1710.06654 (2017). arXiv:1710.06654 http://arxiv.org/abs/1710.06654

[8] Philip I. Pavlik, Hao Cen, and Kenneth R. Koedinger. 2009. Performance Factors Analysis –A New Alternative to Knowledge Tracing. In *Proceedings of the 2009 Conference on Artificial Intelligence in Education: Building Learning Systems That Care: From Knowledge Representation to Affective Modelling*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 531–538. http://dl.acm.org/citation.cfm?id=1659450.1659529

[9] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162

[10] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas Guibas, and Jascha Sohl-Dickstein. 2015. Deep Knowledge Tracing. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS'15)*. MIT Press, Cambridge, MA, USA, 505–513. http://dl.acm.org/citation.cfm?id=2969239.2969296

[11] Siddharth Reddy, Igor Labutov, and Thorsten Joachims. 2016. Learning Student and Content Embeddings for Personalized Lesson Sequence Recommendation. In *Proceedings of the Third (2016) ACM Conference on Learning @ Scale (L@S '16)*. ACM, New York, NY, USA, 93–96. https://doi.org/10.1145/2876034.2893375

[12] R.G.D. Steel and J.H. Torrie. 1960. *Principles and procedures of statistics: with special reference to the biological sciences*. McGraw-Hill. https://books.google.com.ar/books?id=o6FpAAAAMAAJ

[13] Steven Tang, Joshua C. Peterson, and Zachary A. Pardos. 2016. Deep Neural Networks and How They Apply to Sequential Education Data. In *Proceedings of the Third (2016) ACM Conference on Learning @ Scale (L@S '16)*. ACM, New York, NY, USA, 321–324. https://doi.org/10.1145/2876034.2893444

[14] Jacob Whitehill, Kiran Mohan, Daniel Seaton, Yigal Rosen, and Dustin Tingley. 2017. MOOC Dropout Prediction: How to Measure Accuracy?. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale (L@S '17)*. ACM, New York, NY, USA, 161–164. https://doi.org/10.1145/3051457.3053974

[15] Xiaolu Xiong, Siyuan Zhao, Eric G Van Inwegen, and Joseph E Beck. 2016. Going deeper with deep knowledge tracing. In *Proceedings of the 9th International Conference on Educational Data Mining (EDM 2016)*. 545–550.