Applying Recent Innovations from NLP to MOOC Student Course Trajectory Modeling

Anonymous
Anonymous Institution
anonymous@anonymous.edu

ABSTRACT

This paper presents several strategies that can improve neural network-based predictive methods for MOOC student course trajectory modeling, applying multiple ideas previously applied to tackle NLP (Natural Language Processing) tasks. In particular, this paper investigates LSTM networks enhanced with two forms of regularization, along with the more recently introduced Transformer architecture.

Keywords

next-step prediction, predictive modeling, course trajectory, mooc, lstm, transformer

1. INTRODUCTION AND MODEL OUTLINES

1.1 Fundamentals of Predictive Modeling

Recent innovations in deep learning methods for NLP (Natural Language Processing) tasks such as [7] in the past few years have consistently pushed the state of the art in a wide range of benchmark NLP tasks, while yielding new strategies that can be applied to predictive modeling tasks in a more general sense. This is because the majority of these NLP tasks within the scope of these innovations can be parameterized in terms of modeling the function f in the equation

$$P(y|x_0, \dots x_t) = f(x_0, \dots x_t; \theta) \tag{1}$$

where f is a probability mass function with parameters θ over the random variable y, and $x_0, \dots x_t$, drawn from a discrete set of tokens T, represent the context from previous time steps. Unfortunately, there is little literature in the domain of education analytics exploring the effectiveness of innovations from NLP for education analytics tasks that also conform to this predictive modeling paradigm. Nevertheless, the LSTM (Long-Short-Term Memory) DNN (Deep Neural Network) architecture, an earlier innovation which was the architecture of choice for predictive modeling tasks in NLP before the past few years, has been successfully applied to several education analytics tasks. These papers demonstrate potential for further exploitation of the similarities between predictive modeling tasks in education analytics and NLP, while providing a baseline to compare with more recent innovations presented in this paper.

1.2 Previous Work with MOOC Course Trajectory Modeling

One of the first papers to present an application of DNN models for predictive modeling tasks in education analytics is [4], where the authors specifically investigate the applicability of DNN models for modeling student course trajectories in MOOCs (Massive Open Online Courses). Specifically, the authors of [4] demonstrate the effectiveness of LSTM DNN models for this task over other strategies, such as using n-gram models that condition their predictions over small number of past course nodes. Finally, the authors provides suggestion for incorporating such a predictive model in a wider context, including tie-ins with the MOOC service to provide user-facing suggestions and live feedback to monitor the predictive model's performance.

1.3 Baseline LSTM

This model is identical to the Baseline LSTM model featured in [4], using the same LSTM architecture and a nearly identical hyperparameter set and training scheme, further detailed in Section 2.2. This model is intended as a control baseline to assess the performance of other models tested in this paper.

1.4 Transformer Architecture

As noted in Section 1, the shared abstraction of both course trajectory modeling and many NLP tasks as a discrete next-step predictive task suggests applying innovations from NLP to improve performance in course trajectory modeling. In particular, the Transformer architecture, first featured in [7], is one such major architectural innovation.

Transformer Architecture Details. In [7], the authors construct a DNN model architecture centered around modular Transformer blocks as described in Section 1.4. In contrast with the need for $O(n)$ forward and backward passes per input sequence through each of the LSTM recurrent nodes, the entire Transformer model is designed to only require one forward and backward pass through the entire model to process each input sequence. In addition to forming next-step predictive models from these transformer blocks, The authors also provide additional architectural topologies for

models tailored to other tasks such as machine translation or text classification, emphasizing the applicability of the Transformer blocks for a diverse array of NLP tasks. Please refer to [7] for more information about the composition of a Transformer-based next step predictive mode.

Multi-Head Dot-Product Self-Attention. The multi-head dot product self-attention mechanism is the core architectural innovation which enables a Transformer block to fit to temporal correlations present in the training set in one forward and backward pass. On an abstract level, the operation used by the dot product self-attention mechanism with h heads to compute temporal correlations is the the scaled and masked outer product of each $k_i, q_i : i \in \{1, \ldots, h\}$ (derived from the input tensor $x \in \mathbb{R}^n \times \mathbb{R}^d$) as shown in Equation (2):

$$t_i = \text{softmax}\left(\frac{\text{mask}(k_i q_i^\top)}{\sqrt{\lfloor d/h \rfloor}}\right) \tag{2}$$

The results $t_i \in \mathbb{R}^n \times \mathbb{R}^n : i \in \{1, \ldots, h\}$ then directly capture how the features of each k_i and q_i are correlated over each pair of time steps in the input sequence. Note that the operator mask$(\cdot)$ in Equation (2) zeros out lower triangular entries in $k_i q_i^\top \in \mathbb{R}^n \times \mathbb{R}^n$, corresponding to the dot product of features in q_i with features in k_i from previous time steps. Please refer to [7] for more information about the multi-head dot product self-attention mechanism and the the Transformer block as a whole.

1.5 LSTM Enhancements

This section features two different enhancements featured in Kirill Mavreshko's [1] implementation of a Transformer-based next-step predictive model that could be independently used with LSTM models to yield performance improvements for student course trajectory modeling in MOOCs. These enhancements have also been independently backed with theoretical justification and empirical experiments, demonstrating performance improvements in coordinated NLP tasks when applied to LSTM models, as further detailed in [2] and [5].

Confidence Penalty Term in Loss. The baseline LSTM model already features some form of regularization, particularly dropout in the weights of the recurrent layers during training. However, for any classification task with a correct output label y_{true} in a set of possible labels T, examining the equation for the cross-entropy classification loss

$$L(\theta) = -\log P(y_{true}; \theta) = -\log p_{true} \quad p_j = P(y_j; \theta)\ \forall j \in T \tag{3}$$

suggests an additional regularization term that penalizes highly confident distributions to reduce overfitting. The confidence penalty uses $H(p(y;\theta))$ as quantitative measure of confidence in a model's output distribution, where a higher value represents a lower level of confidence that the model predicts for each outcome $j \in T$. As a result, the new loss function expands to

$$L^*(\theta) = -\log p_{true} - \beta H(p(y;\theta)) = -\log p_{true} + \beta \sum_{j \in T} p_j \log p_j \tag{4}$$

where β is a scalar hyperparameter weight for the Confidence Penalty loss term. For theoretical arguments and empirical evidence for adding a confidence penalty term, please refer to [5].

Tied Embedding Layers. Another opportunity to introduce additional regularization to any sort of discrete next-step predictive model is found when examining the model's embedding and output layers, specifically

- The embedding layer L with dimension $|T| \times d_{embed}$, mapping input tokens in T to vectors in a latent feature space.
- The output layer W with dimension $d_{final} \times |T|$, mapping the final intermediate layer output h to an probability logit over the set of all input tokens T.

After enforcing the condition $d_{embed} = d_{final}$ by inserting a feed-forward layer between the rest of the model and the output later, W is tied to the embedding layer L fixing $W = L^\top$. For theoretical arguments and empirical evidence for the effectiveness of tying the output layer in this fashion, please refer to [2].

2. EXPERIMENTS AND EMPIRICAL RESULTS

2.1 Dataset Cleaning and Processing

2.1.1 Procedures for Dataset Cleaning and Processing

Given that the task of student course trajectory modeling requires predicting where a student will navigate next given the student's previous navigation patterns, extensive processing of raw MOOC server logs is required before any training can occur. This process is explained in great detail in [4], with the main steps listed below:

1. Given the raw server log records, select the `basic_action` column, timestamp, username, and title columns necessary to build unique course node tokens in step 3.
2. Filter out all log records except those with `basic_action` label `seq_next`, `seq_prev`, or `seq_goto`, representing the full set of navigation actions a student can take for each of the MOOCs.
3. Construct a unique positive integer token ID for each course node through concatenating each component of the full course path to construct a unique name for each course node, then assigning each unique name to the token ID.
4. Assemble the full sequence of navigation records for each user by grouping by user ID, then ordering within in each group by timestamp.

[1]Copyright 2018 by Kirill Mavreshko. Source code at `https://github.com/kpot/keras-transformer`

Table 1: MOOC Course Trajectory Dataset Summary Statistics

Institution	Course	Term	Nodes	Users
DelftX	AE1110X	Fa. 2015	291	14496
UCBX	EE40LX	Fa. 2015	287	30633
UCBX	Fin101X	Sp. 2016	114	2951
UCBX	ColWri2.2X	Sp. 2016	54	40698
UCBX	CS169.2X	Sp. 2016	204	940
UCBX	Policy01X	Sp. 2016	129	1804

Table 2: Main Hyperparameters by Architecture Type

Architecture Type	LSTM	Transformer
Max. Seq. Length	256	256
Main Layer Width	128	128
Layer/Block Count	2	2
Attention Heads	N/A	8
Optimizer	Adam	Adam
Learning Rate	0.01	0.0005
Batch Size	128	64

5. Prepend the token ID representing the course homepage to every sequence that does not already begin with this token ID, then pad or truncate of the resulting sequences to the maximum sequence length, adding `0` tokens if necessary.

2.1.2 *Additional Notes on Dataset Selection*

In [4], additional criteria are included for selecting courses used to demonstrate the utility of a student course trajectory model, including approximating of the entropy of each dataset as a set of discrete random processes via fitting a HMM (Hidden Markov Model) to each dataset. For the experiments in this paper, limited access to MOOC trajectory records preempts the utility of filtering out datasets with low entropy over all course sequences. Table 1 provides summary statistics for the six courses chosen for this paper, hailing from the MOOC offerings of these two universities:

- DelftX from the Delft University of Technology in Delft, Netherlands
- UCBX from the University of California, Berkeley in Berkeley, California

2.2 Hyperparameters and Training Context

All training and evaluation was completed on a remote Linux server CPU equipped with 2 GeForce Titan X GPUs (Graphics Processing Units). The script for training and evaluation is written in Python 3 using the Keras [1] deep learning API over a Tensorflow backend. Table 2 provides the full set of hyperparameters used for training and evaluating each model on each course record dataset. As the goal of this paper is to demonstrate specific differences in model architecture and training that lead to performance gains relative to the earlier results, hyperparameter tuning was not done for any of the LSTM models to facilitate comparison with results in [3]. Additionally, minimal hyperparameter tuning was done on for the Transformer models in order to minimize the risk of overfitting to the datasets for each course.

Simultaneous Fitting to Multiple Datasets. Since records from each of the 6 courses were processed as described in Section 2.1 independently, attempting to fit models on multiple courses would result in collisions between different sets of course node tokens. Nonetheless, building a predictive model that can fit to datasets from a wide range of courses is a well-defined area for future research.

2.3 Empirical Results

Table 3 presents summary statistics for each model's final test accuracy and total training time per batch for each of the six datasets listed in Table 1. Table 4 presents additional metrics for the Transformer model pertinent to the analysis in Section 2.4.3. All statistics in both Table 3 and Table 4 are recorded using the default set of Keras command line logging tools. [2]

2.4 Analysis and Further Considerations

2.4.1 *Baseline LSTM Comparison with Previous Results*

At face value, the results for the Baseline LSTM model corroborate those presented in [4], with the caveat that average accuracy metrics reported in [4] are calculated in a different fashion that effectively gives more weight to correctly predicting tokens that occur in shorter course trajectory sequences.

2.4.2 *Comparison of Final Test Accuracy Between Models*

Table 3 and Table 4 show that the Transformer model achieves an average final test accuracy of around 63 percent, approximately on par with the average final test accuracy of both LSTM models without tied embeddings, in contrast with a marginally yet consistently higher 64 percent average for the LSTM models that use Tied Embeddings (as described in Section 1.5). On the other hand, including the Confidence Penalty (as described in Section 1.5) does not provide any meaningful improvement in final test accuracy for any of the six datasets. As the training scheme for all models featured in this paper invoke early stopping after 3 epochs without improving validation loss, training any of the above models for more epochs will most likely lead to overfitting on the training set.

2.4.3 *Further Analysis of Final Test Accuracy for Transformer Models*

At a first glance, the final test accuracy results in Table 4 seem to contradict Transformer models' considerable performance improvements over LSTM models demonstrated in [7]. Nevertheless, the largest dataset featured in this paper only includes 40,698 course trajectory sequences, which is multiple orders of magnitude smaller than the WMT machine translation datasets used in [7] with millions of sentence pairs per language pair. This discrepancy in dataset size can cause overfitting for a particular deep learning architecture optimized to train with much larger datasets, even while controlling for model and training hyperparameters. Furthermore, the final training accuracies listed in Table 4 suggest that the Transformer model has overfit to

[2]BaseLogger and ProbarLogger Callback utilities. [1]

Table 3: Overall Performance Metrics by Architecture

Model	Baseline LSTM	LSTM w/ Conf. Penalty	LSTM w/ Tied Emb.	LSTM w/ Both Enh.	Transformer
Final Test Accuracy					
Average	0.6373	0.6355	0.6418	0.6388	0.6383
Std. Dev.	0.05623	0.05558	0.05728	0.05592	0.05455
Training Time per Batch					
Average	35 ms	35 ms	35 ms	35 ms	2 ms
Std Dev.	0.94 ms	0.92 ms	0.89 ms	0.86 ms	0.04 ms

Table 4: Additional Performance Metrics for the Transformer Model

	Test Acc.	Train Acc.
Average	0.6307	0.6383
Standard Deviation	0.06081	0.05455
Avg. for Large Datasets	0.6438	0.6411
Avg. for Small Datasets	0.6175	0.6355

the smaller datasets in this paper despite the use of early stopping, particularly for the following two datasets from courses with fewer than 2,000 unique users as recorded in Table 1:

- UCBX CS169.2X with 904 unique users
- UCBX Policy01X with 1,804 unique users

In conclusion, these results provide evidence that Transformer-based models do not yield benefits in accuracy over LSTM models when trained with datasets of similar size to the MOOC course trajectory datasets featured in this paper, in contrast with the much larger datasets common to certain NLP tasks such as machine translation.

2.4.4 Further Analysis of LSTM Enhancements

Given that both the Tied Embedding Layers and the Confidence Penalty are theoretically motivated by a search for new forms of model regularization, the empirical results in Table 3 indicate the Tied Embeddings are a marginally more effective form of regularization than the Confidence Penalty for this task. Furthermore, since the Tied Embedding enhancement specifically targets the input embedding and output layers of a discrete next-step predictive model for regularization in contrast to the Confidence Penalty altering the entire model's loss function, the embedding and output layers of each of the LSTM models play a disproportionately important role in the model's performance as a whole for this task.

2.4.5 Analysis of Training Time Results

In contrast to the Transformer model's lack of improvement in final test accuracies for all six datasets, the results in Table 3 and Table 4 suggest that the Transformer model outperforms all types of LSTM models by more than an order of magnitude with respect to total training time per batch. Given that both the LSTM and Transformer models are built to accommodate a maximum sequence length of 256 as indicated in Table 2, the results in Table 4 are consistent with the reduced number of training passes through the model's computational graph per input sequence, as encapsulated in the Transformer architecture's design goals from Section 1.4.

2.5 Directions for Future Research

2.5.1 Task-Specific Model Enhancements

As mentioned in multiple sections of this paper, certain task-specific strategies for improving performance on MOOC course trajectory prediction covered in [4] are not investigated in this paper, even if applying these strategies in conjunction impact performance in a noteworthy manner. Some of these additional strategies used to improve performance on these two tasks include:

- Calculating the entropy of each dataset's best-fit HMM transition matrix as a criterion for selecting MOOC course trajectory datasets used to evaluate the enhanced LSTM and Transformer models.
- Incorporating auxiliary data inputs, including the time difference between course navigation actions, into evaluating the benefits of enhanced LSTM models and Transformer models over baseline results from [4].

2.5.2 Model Pre-Training and Multitask Learning

Another more ambitious goal for further research involves constructing one model that can provide meaningful predictions for multiple tasks with minimal training needs. Given the wide applicability of models with generalized predictive modeling capabilities, this model would most likely incorporate innovations originally designed to provide multitask capabilities for NLP applications. For example, [6] presents a NLP model that is first trained to perform a next-word prediction task on large text datasets before undergoing fine-tune training for more specific downstream NLP tasks, which include tasks such as text classification, sentence embedding, question answering and free-form text generation. In the context of education analytics tasks, an analogous suite of tasks for such a model could include modeling overall course performance and individualized suggestions for instructors assisting students with course material.

3. REFERENCES

[1] F. Chollet et al. Keras. https://keras.io, 2015.

[2] H. Inan, K. Khosravi, and R. Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:arXiv:1611.01462*, 2016.

[3] Z. Pardos, Z. Fan, and W. Jiang. *Connectionist recommendation in the wild: On the utility and scrutability of neural networks for personalized course*

guidance, volume 29, pages 487–525. Springer Nature B.V., Netherlands, 2nd. edition, 2019.

[4] Z. Pardos, S. Tang, D. Davis, and C. V. Le. Enabling real-time adaptivity in moocs with a personalized next-step recommendation framework. In *Proceedings of the Fourth ACM Conference on Learning @ Scale*, L@S '17, pages 23–32, Cambridge, Massachusetts, USA, 2017. Association for Computational Linguistics.

[5] G. Pereyra, G. Tucker, J. Chorowski, L. Kaiser, and G. Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548v1*, 2017.

[6] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkorei, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.