Applying Recent Innovations from NLP to Two Predictive Modeling Tasks in Education Analytics

Clarence Chen
University of California
Berkeley, California, United States
clarencechenct@berkeley.edu

Zachary Pardos
University of California
Berkeley, California, United States
pardos@berkeley.edu

ABSTRACT

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

KEYWORDS

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

ACM Reference Format:
Clarence Chen and Zachary Pardos. 2019. Applying Recent Innovations from NLP to Two Predictive Modeling Tasks in Education Analytics. In *Proceedings of LAK '20: Learning Analytics & Knowledge Conference (LAK '20).* ACM, New York, NY, USA, 7 pages. https://doi.org/?

1 INTRODUCTION

1.1 Fundamentals of Predictive Modeling

Recent innovations in deep learning methods for Natural Language Processing (NLP) tasks such as [10] in the past few years have consistently pushed the state of the art in the tasks they have been designed for, while yielding a wealth of new strategies that can be applied to predictive modeling tasks in a more general sense. This is because the NLP tasks that the majority of these innovations are built for can be parameterized in terms of modeling the function f in the equation

$$P(y|x_0, \ldots x_t) = f(x_0, \ldots x_t; \theta) \quad (1)$$

where f is a PMF over the random variable y, and $x_0, \ldots x_t$ represent the context of previous time steps, drawn from a discrete set of tokens T. Unfortunately, there is little literature in the domain of education analytics exploring the effectiveness of these recent innovations from NLP for relevant tasks that also fit within the general predictive modeling paradigm. Nevertheless, the LSTM (Long-Short-Term Memory) neural network architecture, an earlier innovation which was the architecture of choice for predictive modeling tasks in NLP until the past few years, has been successfully applied to a number of important predictive modeling tasks in the domain of education analytics. These papers demonstrate the potential for further exploitation of the similarities between predictive modeling tasks in education analytics and NLP, while providing a baseline to compare with the effectiveness of the more recent innovations presented in this paper.

1.2 Previous Work with Tasks in Education Analytics

1.2.1 Previous Work with MOOC Course Trajectory Modeling. One of the first papers to present an application of neural network models for predictive modeling tasks in education analytics is [7], where the authors specifically investigate the applicability of neural network models for modeling student course trajectories in MOOCs (Massive Open Online Courses). Specifically, the authors of [7] demonstrate the effectiveness of LSTM neural network models for this task over other strategies, such as returning the next most common course node for all students, or 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.

1.2.2 Previous work with Student Enrollment Modeling. Following in the footsteps of [7], the authors of [6] present a similar investigation into the effectiveness of neural network models for the task of predicting student course enrollment. After testing the performance of LSTM neural network models with alternative strategies such as using n-gram models or returning the most popular course in each student's major for all students, the authors of [6] find that LSTM neural network models are also effective for this predictive modeling task. In a similar vein to [7], the authors of [6] also make sure to provide information situating such a predictive model in a wider context, specifically conducting experiments on the usability of predictions while embedded in a student-facing course recommendation system.

2 MODEL AND ARCHITECTURE DESCRIPTIONS

This section is a detailed description of the three different architectural features assessed in the report, namely two enhancements on top of a baseline LSTM (Long-Short Term Memory) neural network architecture, and the Transformer architecture first introduced in [10].

2.1 Baseline LSTM

This model is identical to the Baseline LSTM next-step predictive model featured in [7], using the same LSTM architecture and a nearly identical hyperparameter set and training scheme, further detailed in Section 3.2. This model is intended as a control baseline to assess the performance of the two other models.

Auxiliary Inputs. There were also several LSTM models in [7] that featured a per-token auxiliary input derived from the difference between timestamps of consecutive navigation actions as described in Section 3.1. These models yielded higher final test accuracy than the Baseline LSTM model in [7], but the drastic performance gains observed in the enhanced LSTM and Transformer models in this report suggest that including this auxiliary input is only of secondary importance for the task of predicting student course trajectories.

2.2 LSTM Enhancements

This section features two different enhancements that could yield performance improvements on the online course trajectory datasets from Section 3.1 when applied to the Baseline LSTM models.

2.2.1 *Confidence Penalty Term in Loss.*

Introduction. The baseline LSTM model already features some form of regularization, particularly dropout with probability 0.2 active in the weights of the recurrent layers during training. However, for any classification task with input $\mathbf{x}$ and correct output label y_{true} in a set of possible labels T, examining the equation for the cross-entropy classification loss

$$L(\theta) = -\log P(y_{true}; \theta) \tag{2}$$

suggests the possibility of constructing a regularization term that penalizes highly confident distributions to reduce overfitting. In [8], the authors establish the entropy of the output distribution

$$H(p(y;\theta)) = -\sum_{j \in T} P(y_j;\theta) \log P(y_j;\theta) \tag{3}$$

as a quantitative measure of confidence in a model's output distribution. Now for brevity, after defining the probability for each course node $y_j \in T$ produced by the model as

$$p_j = P(y_j;\theta) \qquad \forall j \in \{1, \ldots, T\} \tag{4}$$

the new loss function expands to

$$L^*(\theta) = -\log p_{true} - \beta H(p(y;\theta)) = -\log p_{true} + \beta \sum_{j \in T} p_j \log p_j \tag{5}$$

where β is a scalar hyperparameter weight for the Confidence Penalty loss term.

Empirically Measured Benefits. In addition to theoretical justification for using the Confidence Penalty as a form of regularization, the authors also present empirical results that demonstrate that an LSTM model with a Confidence Penalty loss term outperforms LSTM models with the following alternative regularization schemes for word-level language modeling of texts in the Penn Treebank text dataset [5].

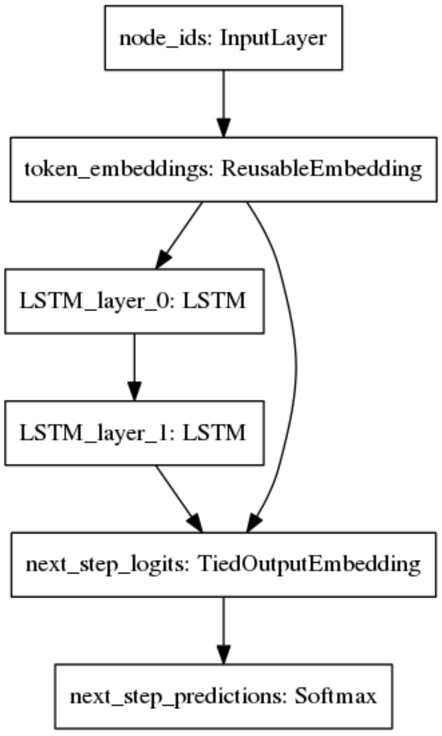

node_ids: InputLayer
token_embeddings: ReusableEmbedding
LSTM_layer_0: LSTM
LSTM_layer_1: LSTM
next_step_logits: TiedOutputEmbedding
next_step_predictions: Softmax

Figure 1: Architectural topology of the LSTM model with Tied Embeddings used in Section 3.

- Label smoothing, which adds a different positive penalty $\beta D_{KL}(u||p)$ to the loss function, a measure of the divergence of the output distribution p from the uniform distribution u over all possible labels.
- Label noise, which directly perturbs each of the probabilities in output distribution by a random amount.

2.2.2 *Tied Embedding Layers.*

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

In [4], the authors explore one strategy for additional regularization involving these layers, specifically fixing $W = L^\top$. In order for the dimensions of W and $L^\top$ to match, the condition $d_{embed} = d_{final}$ is enforced through inserting one feed-forward layer between the rest of the model and the output layer.

Theoretical Arguments. One immediate benefit of fixing $W = L^\top$ is the elimination of the parameter W from the model while still maintaining the same representational power under non-degenerate conditions. Specifically, the authors in [4] explore of a custom loss penalty that also provides an empirically assessed benefit, where they show

$$hW = (y_{true}L)L^\top \implies hW \text{ lies in the row space of } L^\top \tag{6}$$

when a discrete next-step predictive model with $d_{embed} = d_{final}$ is trained to incur zero custom loss penalty under certain general conditions. As long as W and $L^\top$ are of full rank, and the set of intermediate layer outputs h span the entire hidden state space of dimension d_{embed}, W and $L^\top$ must span the same row space, implying that there exists a matrix A satisfying

$$W = AL^\top \ : A \in \mathbb{R}^{d_{embed} \times d_{embed}} \tag{7}$$

This matrix A could then be encoded in the rest of the predictive model, which now includes the added feed-forward layer.

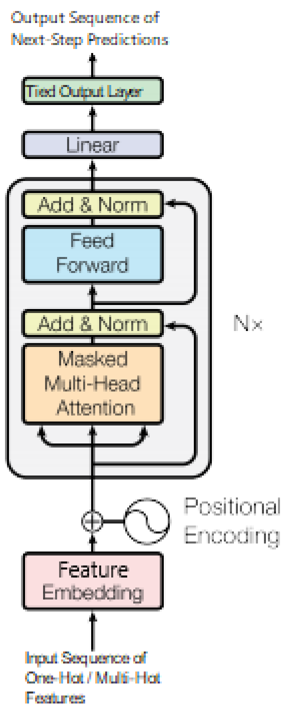
Empirically Measured Results. The authors also demonstrate that LSTM models with Tied Embedding Layers outperform LSTM models equipped with other regularization strategies on the same word-level Penn Treebank [5] language modeling task as in [8], but [4] uses an LSTM model with variational dropout as their baseline, a more sophisticated dropout scheme developed by [2].

2.3 Transformer Architecture

2.3.1 Introduction. As noted in Section 1, the shared abstraction of both course trajectory modeling and language modeling as a discrete next-step predictive task points to applying further innovations from language modeling to improve performance in course trajectory modeling. In particular, the Transformer architecture, first featured in [10], is one such major architectural innovation.

2.3.2 Multi-Head Dot-Product Self-Attention. Instead of relying on LSTM recurrent nodes, which require $O(n)$ sequential passes of forward computations and backpropagation with a sequence of n tokens, the Transformer architecture only uses $O(1)$ sequential passes to process data over the entire sequence, using Multi-Head Dot-Product Self-Attention to allow the model to fit to temporal correlations present in the training set. In [10], given a 2D input tensor $x \in \mathbb{R}^n \times \mathbb{R}^d$ with d features for each of n time steps, the core Transformer block (using Dot-Product Self-Attention with h heads) computes a output tensor $y_{out} \in \mathbb{R}^n \times \mathbb{R}^d$ using the following list of weights

$$W_k, W_q, W_v, W_o \in R^{d \times d} \tag{8}$$
$$W_1, W_2 \in \mathbb{R}^{d \times d}, b_1, b_2 \in \mathbb{R}^{n \times d} \tag{9}$$



Output Sequence of Next-Step Predictions
Tied Output Layer
Linear
Add & Norm
Feed Forward
Add & Norm
Masked Multi-Head Attention
N×
Positional Encoding
Feature Embedding
Input Sequence of One-Hot / Multi-Hot Features

Figure 2: Transformer Model Architectural Topology derived from Vaswani, et. al.

according to the operator definitions and formulas below:

$$\begin{aligned}
k, q, v &= xW_k, xW_q, xW_v && (10)\\
k_i, q_i, v_i &= \text{split}_i(k), \text{split}_i(q, \text{split}_i(v) \quad \forall i \in \{1, \dots, h\} && (11)\\
t_i &= \text{softmax}\left(\frac{\text{mask}(k_i q_i^\top)}{\sqrt{\lfloor d/h \rfloor}}\right) \quad \forall i \in \{1, \dots, h\} && (12)\\
a_i &= \text{dropout}(t_i)v_i \quad \forall i \in \{1, \dots, h\} && (13)\\
y_0 &= \text{concat}(a_1, \dots, a_h)W_o && (14)\\
y_1 &= \text{lnorm}\,(x + \text{dropout}(y_0)) && (15)\\
y_2 &= \max(0, y_1 W_1 + b_1)W_2 + b_2 && (16)\\
y_{out} &= \text{lnorm}\,(x + \text{dropout}(y_2)) && (17)
\end{aligned}$$

- The operator split_i in Equation (11) splits each of the tensors k, q, v across the set of d features into h parts, with each part sent to one of the h self-attention heads. Likewise, the operator $\text{concat}(\cdots)$ in Equation (14) concatenates the outputs of each self-attention head back into a tensor with dimension $n \times d$.
- The operator $\text{mask}(\cdot)$ in Equation (12) zeros out the lower triangular entries in $k_i q_i^\top \in \mathbb{R}^n \times \mathbb{R}^n$ that correspond to the dot product of queries in q_i with keys from previous time steps in k_i.
- The operator $\text{lnorm}(\cdot)$ performs layer normalization on the input, first normalizing the set of d features (in this case), then re-centering and re-scaling them according to learned mean and variance parameters.

2.3.3 Transformer Architecture Details. In [10], the authors take inspiration from deep Computer Vision models such as Resnet [3] to construct a Deep Neural model architecture using the Transformer blocks described in Section 2.3.2, avoiding the need for an increasing number of sequential computations with respect to the (maximum) sequence length. Figure 2 outlines how the Transformer blocks are organized to form a next-step predictive model. 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 block for a diverse array of Natural Language Processing tasks.

3 EXPERIMENTS ON MOOC STUDENT COURSE TRAJECTORY DATA

3.1 Dataset Cleaning and Processing

3.1.1 Procedures for Dataset Cleaning and Processing. Given that the task of predicting student course trajectories requires predicting where a student will navigate next given the student's previous navigation patterns, extensive cleaning and labeling of raw online course server logs is required before any model training can occur. This process is explained in great detail in [7], with the main steps listed below:

(1) Dropping most data columns from the server log records for each course, except for the `basic_action` column, timestamp, username, and title columns necessary to build unique course node tokens in step 3.
(2) Filtering out all log records except those with `basic_action` label `seq_next`, `seq_prev`, or `seq_goto`, which represent the full set of navigation actions recorded in the server log records for each course.
(3) Constructing a unique token 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 node to a token represented by a positive integer.
(4) Assembling the full sequence of navigation records for each user by grouping by user ID, then ordering the records in each group by timestamp.
(5) Prepending the token representing the course homepage to every sequence that does not already begin with this token.
(6) Padding or truncation of the resulting sequences to the maximum sequence length hyperparameter using `0` tokens.

3.1.2 Dataset Selection Caveats. In [7], additional criteria are included for selecting courses used to demonstrate the utility of a predictive course trajectory model, including approximation of the entropy of each sequence as a discrete random process by fitting a Hidden Markov model and calculating the entropy of the transition matrix on the model's hidden states. For the experiments featured in this report, the limited access to MOOC trajectory records preempts the utility of filtering out courses with low variation in student trajectories. In fact, most of the available courses had records from two or more different iterations over time. Investigating adaptations of existing models to updated iterations of known courses provides ample opportunity for future research, but is not as useful for demonstrating that the innovations featured in this report generalize across different courses. Table 1 provides summary statistics

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

for the six courses chosen for this report, hailing from the online course offerings of these two universities:

- Delft University of Technology in Delft, Netherlands (marked as DelftX in all tables)
- University of California, Berkeley in Berkeley, California (marked as UCBX in all tables)

3.2 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 and Table 3 provides the full set of hyperparameters used for training and evaluating each model on each course record dataset. As the goal of this report is to demonstrate specific differences in model architecture and training that lead to performance gains relative to the results in [7], hyperparameter tuning was not done in any of the experiments, even if it may provide further gains in final test accuracy.

Simultaneous Fitting to Multiple Datasets. Since records from each of the 6 courses were processed as described in Section 3.1 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 online courses is a well-defined area for future research.

3.3 Empirical Results

Table 4 and Table 5 together present the following statistics for each of the different models over each of the 6 available course navigation datasets.

- The total number of model parameters (marked as 'Params.' in all tables)
- The total number of epochs trained before early stopping (marked as 'Eps.' in all tables)
- The total time spent training before early stopping (marked as 'Time' in all tables)
- The final test accuracy for next-step predictive (marked as 'Acc.' in all tables)

Table 2: Hyperparameters by Task for LSTM Models

Task	MOOC Course Trajectory Prediction				Student Course Enrollment Prediction			
LSTM Model Type	Base	Tied Emb.	Conf. Penalty	Both	Base	Tied Emb.	Conf. Penalty	Both
Max. Sequence Length	256				25			
Embedding Size	128				256			
LSTM Cell Width	128				256			
LSTM Layer Depth	2				1			
LSTM Kernel Dropout	0.2				0	0.2		
Embedding ℓ_2 Reg. Weight	0	10^{-6}	0	10^{-6}	0	10^{-6}	0	10^{-6}
Tied Output Kernel Dropout	N/A	0.6	N/A	0.6	N/A	0.6	N/A	0.6
Conf. Penalty Weight	0	0	0.1	0.1	0	0	0.1	0.1
Optimizer	Adam				Adam			
Initial Learning Rate	0.01				0.001			
Batch Size	64				128			

Table 3: Hyperparameters by Task for Transformer Models

Task	Course Trajectory	Student Enrollment
Max. Sequence Length	256	25
Embedding Size	128	256
Transformer Width	128	
Transformer Depth	128	
Transformer Heads	8	
Transformer Dropout	0.1	
Embedding ℓ_2 Reg.	10^{-6}	
Tied Output Dropout	0.6	
Conf. Penalty Weight	0.1	
Optimizer	Adam	
Initial Learning Rate	0.001	0.0001
Batch Size	128	

All statistics in both Table 4 and Table 5 are recorded using the default set of Keras command line logging tools, and total training times are rounded to the nearest minute. [1]

3.4 Analysis and Further Considerations

3.4.1 Baseline LSTM Comparison with Previous Results. At face value, the results for the Baseline LSTM model corroborate those presented in [7], with the caveat that average accuracy metrics reported in [7] are calculated in a different fashion that effectively gives more weight to correctly predicting tokens that occur in shorter course trajectory sequences.

3.4.2 Comparison of Final Test Accuracy Between Models. Table 4 and Table 5 together show that the Transformer model and both of the LSTM models which use Tied Embeddings (as described in Section 2.2.2) achieve an average final test accuracy of around 95 percent, compared to the approximately 63 percent average final test accuracy of both the Baseline LSTM model and the Confidence Penalty-only LSTM model. On the other hand, including the Confidence Penalty (as described in Section 2.2.1) seems to not provide any meaningful improvement in final test accuracy for any of the six datasets, even if training stops after a fewer number of epochs for some 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 not yield significant improvement in final test accuracy

3.4.3 Further Analysis of Final Test Accuracy Results. 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 4 indicate the Tied Embeddings are a much more effective form of regularization than the Confidence Penalty for this modeling task. Furthermore, the Tied Embedding Layers specifically target the embedding and output layers of a discrete next-step predictive model for regularization, while the Confidence Penalty imposes a regularizing effect on the model as a whole through altering the loss function. These differences between the two regularization methods point to poorly learned representations of the different course nodes as a potential proximate cause for the Baseline LSTM model's relatively poor performance in accurately modeling student course trajectories.

3.4.4 Analysis of Training Time Results. While the LSTM models with Tied Embedding Layers and the Transformer model achieve approximately equal final test accuracies for all six datasets, the results in Table 4 and Table 5 also suggest that the Transformer model outperforms all types of LSTM models with respect to total training time. These training time savings are especially large for the following three courses with more than 10,000 unique users as recorded in Table 1:

- AE1110x Fall 2015 offered by DelftX
- EE40LX Fall 2015 offered by BerkeleyX
- ColWri2.2x Spring 2016 offered by BerkeleyX

Given that both the LSTM and Transformer models are built to accommodate a maximum sequence length of 256 as indicated in Table 2 and Table 3, the results in Table 5 are consistent with the

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

Table 4: LSTM Properties and Performance Metrics by Architecture and Dataset

Dataset	Baseline LSTM				LSTM w/ Conf. Penalty		LSTM w/ Tied Embeddings				LSTM w/ Both	
Course Name	Params.	Eps.	Time	Acc.	Eps.	Acc.	Params.	Eps.	Time	Acc.	Eps.	Acc.
AE1110x	338K	14	120 min	0.6091	18	0.6079	317K	30	115 min	0.9356	20	0.9353
EE40LX	337K	13	147 min	0.6925	12	0.6981	316K	7	49 min	0.9517	9	0.9387
Fin101x	293K	10	13 min	0.7058	14	0.7098	294K	31	23 min	0.9762	16	0.9759
ColWri2.2x	277K	11	219 min	0.6428	9	0.6418	287K	14	134 min	0.9737	14	0.9736
CS169.2x	316K	28	7 min	0.5978	20	0.5867	306K	46	38 min	0.9342	59	0.9357
Policy01x	297K	20	8 min	0.5587	18	0.5537	296K	36	14 min	0.9474	9	0.8942
Avg. Test Acc.	N/A	N/A	N/A	0.6345	N/A	0.6330	N/A	N/A	N/A	0.9532	N/A	0.9416

Table 5: Transformer Properties and Performance Metrics by Dataset

Course Name	Params.	Eps.	Time	Acc.
AE1110x	878K	28	19 min	0.9339
EE40LX	877K	34	26 min	0.9532
Fin101x	855K	43	5 min	0.9758
ColWri2.2x	847K	24	39 min	0.9734
CS169.2x	867K	93	3 min	0.9300
Policy01x	857K	49	3 min	0.9463
Average Test Acc.	N/A	N/A	N/A	0.9521

$O(1)$ versus $O(n)$ sequential computation hypothesis for a sequence of length n, as encapsulated in the Transformer architecture's design goals from Section 2.3. Section 4 also features additional experiments involving UC Berkeley course enrollment data, where the maximum sequence length for all models is set to 25. The results of these experiments in Table 7 also provide further evidence supporting the sequential computation hypothesis. Specifically, the Transformer model's training time savings relative to the LSTM models as presented in Table 7 are less impressive compared the savings in Table 5, which is expected given the sequential computation hypothesis.

4 EXPERIMENTS ON STUDENT COURSE ENROLLMENT DATA

4.1 Background, Dataset, and Metrics

4.1.1 Background. In addition to the experiments on modeling student navigation data from MOOCs, LSTM architectures have also been employed in modeling student enrollment data. This related yet distinct modeling task presents another avenue to explore the effectiveness of Transformer-based models and enhanced LSTM models for predictive modeling tasks in education analytics.

4.1.2 Description of Dataset and Metrics. Five types of LSTM and Transformer models, with hyperparameters listed in Table 2 and Table 3 designed to align with those in [6], were also trained to model an updated version of the UC Berkeley student enrollment dataset from (Pardos 2019). This course enrollment dataset includes student enrollment data for Fall 2017 and the 25 previous academic

Table 6: UC Berkeley Course Enrollment Data Summary Statistics

Number of Academic Terms in Dataset	26
Number of Students in Dataset	83892
Number of Courses in Dataset	4487

terms (Fall, Spring, and Summer terms for each calendar year), with more detains in Table 6. The main evaluation metric employed in (Pardos 2019) is Recall@10 over the **last semester** of course predictions, defined as the recall value over the top 10 predicted courses, or

$$Recall@10 = \frac{1}{|S|}\sum_{i \in S}\frac{|y_{i,true} \cap y_{i,top10}|}{|y_{i,true}|} \quad (18)$$

where $y_{i,true}$ is the ground truth set of courses that student $i \in S$ enrolled in during the last term of course predictions, and $y_{i,top10}$ is set of 10 courses with the highest predicted output probability in the last term of course predictions.

4.1.3 Models with Auxilliary Inputs. In [6], experiments also included LSTM models which accepted auxiliary inputs such as the declared major, entry type and discretized GPA of each student for each academic term, and found small improvements in final test Recall@10 over LSTM models that did not use such inputs. The models evaluated in this experiment do not use any auxiliary inputs, but evaluating models with auxiliary inputs is certainly another direction for future research.

4.2 Empirical Results

Table 7 presents the following results of fitting five different LSTM and Transformer models to the UC Berkeley student enrollment dataset from Section 3.2, with hyperparameters for each model listed in Table 2 and Table 3.

- The total number of model parameters
- The total number of epochs trained before early stopping
- The final test Recall@10 for next-step predictive as defined in Equation 18
- The total time spent training before early stopping
- The average time spent training per epoch

Table 7 presents empirical results of fitting five different LSTM and Transformer models to the UC Berkeley student enrollment dataset

Table 7: Model Test Accuracy on UC Berkeley Student Enrollment Dataset

Model Type	Parameters	Epochs	Test Recall@10	Train Time (m)	Avg. Train Time/Epoch (m)
Baseline LSTM	2,827,143	19	0.4281	843	44
LSTM w/ Confidence Penalty	2,827,143	12	0.4201	518	43
LSTM w/ Tied Embeddings	1,739,520	22	0.4327	985	45
LSTM w/ Both Enhancements	1,739,520	21	0.4312	962	46
Transformer	2,798,336	9	0.4379	371	21

from Section 3.2, with hyperparameters for each model listed in Table 2 and Table 3.

4.3 Analysis and Further Considerations

In contrast to the experiments on student trajectory data from Section 3, the various enhanced LSTM models and Transformer model yield much smaller performance gains over the Baseline LSTM. Additionally, the Transformer model still presents improvements in total training time over all types of LSTM models, but these improvements are less pronounced than improvements seen in the course trajectory prediction task. Note that both of these results are consistent with the sequential computation time hypothesis first introduced in Section 2.3, which predicts that using a Transformer model will yield a less dramatic performance improvement relative to the improvements in Section 3 due to the student enrollment modeling task's shorter maximum sequence length.

5 DIRECTIONS FOR FUTURE RESEARCH

5.1 Additional Task-Specific Model Enhancements

As mentioned in multiple sections of this paper, certain task-specific strategies for improving performance on MOOC course trajectory prediction and course enrollment prediction covered in [7] and [6] 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 Markov transition matrix as a criterion for selecting the set of MOOC course trajectory datasets used to evaluate the performance of the enhanced LSTM and Transformer models.
- Incorporating auxiliary data inputs including the time difference between course navigation actions for MOOC course trajectory modeling, or the declared majors, entry status, and cumulative GPA of each student for student course enrollment modeling, into evaluating the benefits of enhanced LSTM models and Transformer models over baseline results from [7] and [6].

5.2 Model Pre-Training and Multitask Learning

In addition to strategies to further improve performance on the specific tasks already presented in this paper, a more ambitious goal involves constructing one model that can provide meaningful predictions for multiple tasks in multiple contexts with minimal training needs. Given that the success of LSTM and Transformer models for tasks in both NLP and education analytics derives from general capabilities in discrete next-step predictive modeling, this model would most likely incorporate innovations originally designed to provide multitask capabilities in NLP applications. For example, [9] 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 instructor assistance in addition to course enrollment or trajectory modeling.

ACKNOWLEDGMENTS

REFERENCES

[1] François Chollet et al. 2015. Keras. https://keras.io.
[2] Yarin Gal. 2015. A theoretically grounded application of dropout in recurrent neural networks. *arXiv preprint arXiv:1512.05287* (2015).
[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385* (2015).
[4] Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:arXiv:1611.01462* (2016).
[5] Mitchell Marcus et al. 1994. The Penn Treebank: Annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology (HLT '94)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 114–119.
[6] Zachary Pardos, Zihao Fan, and Weijie Jiang. 2019. *Connectionist recommendation in the wild: On the utility and scrutability of neural networks for personalized course guidance* (2nd. ed.). Vol. 29. Springer Nature B.V., Netherlands, 487–525. https://doi.org/10.1007/s11257-019-09218-7
[7] Zachary Pardos, Steven Tang, Daniel Davis, and Christopher Vu Le. 2017. 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)*. Association for Computational Linguistics, Cambridge, Massachusetts, USA, 23–32.
[8] Gabriel Pereyra, George Tucker, Jan Chorowski, Lukasz Kaiser, and Geoffrey Hinton. 2017. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548v1* (2017).
[9] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving Language Understanding by Generative Pre-Training. (2018).
[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762* (2017).