Received October 11, 2020, accepted October 30, 2020, date of publication November 4, 2020, date of current version November 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3035687

Power of Attention in MOOC Dropout Prediction

SHENGJUN YIN, LEQI LEI, HONGZHI WANG, (Member, IEEE), AND WENTAO CHEN
Harbin Institute of Technology, Harbin 150001, China

Corresponding author: Hongzhi Wang (wangzh@hit.edu.cn)

This work was supported in part by NSFC under Grant U1866602 and Grant 61772157.

ABSTRACT The dropout rate of massive open online courses (MOOC) has been significantly high, which makes its prediction an important problem. In this article, we try to transfer the knowledge gained in the field of Natural Language Processing into the field of MOOC dropout prediction, due to the high similarity between them. More specifically, we attempt to study and show the powerful use of attention and conditional random field, both of which have been very popular architectures when solving NLP problems. A novel neural network structure is designed as the combination of these techniques. Extensive experimental results demonstrate that the proposed approach is effective.

INDEX TERMS Deep learning, MOOC, conditional random field.

I. INTRODUCTION

Massive Open Online Courses (MOOC for brief), is a concept of a large number of courses online and accessible to everyone, no matter where, who and when. It has been growing so fast that attracts much attention of teachers, students, workers, investors and even the governments [1],[2],[3]. And the most famous and important MOOC providers should be Coursera, Udacity and edX.

A prominent problem of MOOC since its day of birth is the rather high dropout rate [4]. Since studying online cost little based on the fact that most courses on the MOOC platform are free, and there are usually no supervisors to push the learners, and last but not least, no punishment is for dropping out the course. Thus, making accurate and effective predictions can help much to manage the course on MOOC, and corresponding measures can be taken to strengthen the learning will of the learner. The interest of researchers on this problem never stopped, and some methods have been proposed, basically based on models of machine learning and deep learning [5],[6]. Some of them have achieved high accuracy on specific scenarios [5].

However, the method proposed by previous researches mostly either focus on statistical features only, or use Long Short-Term Memory architecture (LSTM for brief), to make use of the sequential features. One interesting architecture is the ConRec Network proposed by Wei Wang et, 2017 [7]. It only focuses the sequential semantic information, using convolutional layer and LSTM layer to achieve their goal.

LSTM has once been popular when dealing with sequential information, due to its extraordinary ability to extract sequential features of the data [8],[9],[10]. As its variant, the Gated Recurrent Unit is also popular, because of its speed [11]. Feng Xiong *et al.* (2019) dealt with the MOOC dropout prediction problem as a time series prediction problem, and utilized LSTMs on features from back-end records data which not only contains learners interaction on forums, but also contains learners interaction with resources and assignments, with an average accuracy of prediction of about 90 percent [12]. In the work of Shaojie Qu *et al.*(2019), a multi-layer long short-term memory (LSTM) neural network was employed to predict student achievement [13]. The experimental results revealed that this method achieved an accuracy rate of 91% and a recall of 94% on their datasets. Di Sun *et al.* (2019) used a GRU-RNN model formulated based on progress of the course content, instead of presence learning activity [14]. Byungsoo Jeon *et al.* (2019) proposed a time series model that constructs an evolving student state representation using both clickstream data and a signal extracted from the textual notes recorded by human mentors assigned to each student [15].

Despite the effectiveness and successful applications of LSTM and GRU, they have significant shortcomings. Their architecture applies recurrent connections. They firstly process the present input with some kinds of operations and then pass the output to the next layer where the architecture and the weights which the operations use are exactly the same. Thus, they are recurrent. The output produced by the last layer to pass to the next layer is called the hidden state of the recurrent unit. The main idea of recurrent connections, including the unidirectional connections and the bidirectional connections, is that different parts of the sequence should be able to influence each other. In a unidirectional recurrent network from backward to forward, the hidden state of the previous layer contains all the information of all the previous layers. To make the architecture practical, the hidden state must be of a fixed shape, usually above 256 [16],[11],[18],[19],[20].

The associate editor coordinating the review of this manuscript and approving it for publication was Zijian Zhang.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

Obviously, every time a recurrent operation is being applied, the information is being squeezed. Therefore, there will be less information passing through from the more previous layers. That is why LSTM sometimes does not work and need many residual connections [21],[22]. This is its structural flaws which cannot be easily overcome.

Comparatively, the approaches that abandon recurrent architectures avoid such problems. They use attention mechanism as an alternative [23]. Instead of applying recurrent connections and use hidden states to pass the sequence information in a squeezed way, the attention applies a connection which is kind of like fully-connected networks. In such kind of connections, each output is computed from all the inputs, which means letting different each part of the sequence influences each other directly. On the one hand, as the information is not squeezed, the model has stronger representation for sequence information. On the other hand, such connections are surely more reasonable and more organic than the bidirectional recurrent connections to grasp the information from both directions [24].

The most important work when it comes to Attention architecture should be *Attention Is All You Need* [23]. In this article, the researchers proposed several remarkable architectures or techniques, including the self-attention architecture and the multi-head attention architecture. They consider attention as mapping from value to output, parameterized by a pair of key and query. In a more understandable way to explain, attention is a technique to determine how much one part should contribute to the whole.

In a specific case, we get a sentence formed by some words. It is easily recognized that the meaning of one specific word is influenced by others. So we get every word a original meaning, say, value. Then we need to compare one word with another to get their relevance, so we get every word two other features, say, query and key, to be used to compare and be compared. Have got the relevance, we can at last compute a new representation for each word using its original value, its relevance to other words, as well as the original value of the other words. That is how self-attention works and how it makes use of the sequential features. Plus, there is another technique in their work to make use of the position feature, which we see as a kind of sequential feature. They encode the sequence element's position into a vector and then simply concatenate it with their original feature. The mathematical form of this was given in our detailed model.

Thus, we attempt to apply this technique of attention into the field of MOOC dropout prediction. There are some excellent examples to learn from, most of which is the model of Transformer and the model of BERT [23],[24]. Both of them are widely used in Natural Language Processing. In the task of MOOC dropout prediction, the major challenge is how to mask the information from the future.

Models like BERT use Transformer architecture as one of their minimal units and Transformer uses an architecture of encoder-decoder [23]. Such architecture is reasonable in most problem's solutions, but does not work in MOOC dropout prediction where the prediction should be made without the information from the future. For example, a student's learning process is a long time period which can be divided into several subsections. For each subsection, we want to know whether this student will dropout or not in the next subsection. If we use the encoder-decoder architecture, the input contains all the course information including the students' actions in the next subsection where the model should be aware of. That is because that in such architecture, different parts of the input influence each other. Therefore we need to propose a solution to this problem.

Another challenge is how we preprocess the data and extract the features. There are two typical ways of feature selecting. The first is to use statistical features [6]. And the second is to use convolutional layers to extract features from one-hot encoded log data [7] (Another similar way is to assign a weight to every type of actions and use the sum of the weights of the logged actions as the input feature [25]). Statistical features are designed and selected manually, which may not have that strong representation or lack some important information; while taking one-hot encoded log data as input reduces the model's interpretability and makes it even harder to train. Even though it can reach similar performance to that of feature-carefully-designed models, it hardly goes beyond that. Thus, how to design and use features effectively and reasonably is a problem.

To solve these problems, in this article, we introduce a novel deep neural network model for MOOC dropout prediction, using attention to replace LSTM. We use statistical features to construct our feature vector, which is used to predict the label, followed by a linear transformation layer and self-attention layer to strengthen its ability of representation. Different from the Transformer and the BERT model, we abandon the encoder-decoder architecture to mask the information from the future. Instead, we apply a separated attention technique first and then add a masked self-attention layer to make use of the information from the past as well as prevent from the date leakage from the future. Conditional Random Field layer is applied in the end to make better use of the information and prediction from the past. Thus, the two challenges, i.e., to avoid the disadvantage of encoder-decoder and to construct a better representation of features, are solved.

The necessity and advantage of each layer are illustrated as following. The first layer, linear transformation layer, is utilized to map statistical features into a high-dimension space. From the experience from NLP field, such trainable work embeddings help improve the performance. This advantage is quite intuitive: the trainable word embedding enables the word representation to be semantic. The second layer, the self-attention layer enables low-level features to interfere with each other to form high-levels features that could representation both low-level feature and the influence among them. This leads to feature enhancement. The third type of layer is Convolutional neural network (CNNs), which would squeeze the output of the last layer. This enables pattern recognition and produces time-unit-level representation.

Fourthly, again, we used a self-attention layer to reveal the relevance relationship among time series. Surely here it is not desirable if the time units that is supposed to be predicted also get involved; consequently, the masking trick is used here. In this way, future time units no longer contribute information to our present output. Then, transformation, normalization and SoftMax layers are used to convert the output to masked self-attention layer into the label vector and at the same time avoid gradient-vanishing problem. Lastly, a Conditional Random Field layer is used to perform the prediction. Since the labels are predicted sequentially, the sequence features of predictions are taken into account.

We experimentally evaluate our model with other classical models based on an open dataset, and the results show that our model makes a better performance.

II. RELATED WORK

In this section, we review some related work.

A. MODELS IN DROPOUT PREDICTION

The work done by Fisnik Dalipi et, 2018 [5] has concluded the most used machine learning techniques in the problem of MOOC dropout prediction. Since deep learning is sub to machine learning, this also includes some deep models used on this problem. And we will select some to go into more detail. And to make it less confusing, we will separate the deep learning models from machine learning models.

Machine learning models used in this work vary from Linear Regression models, Support Vector Machine models, to Logistic Regression models [5]. Such models usually use statistical features extracted from the original data. To give an example, we have a course and multiple actions a student done over it, like watching lecture videos, asking questions and doing exercises. Then, we could count how many times this student take each action, which can be concatenated into a vector of three integer numbers. At the same time, we get a label which could also be an integer number, where 1 denotes the dropout result, and 0 denotes the non-dropout result. Now the features and the labels prepared, models can be used to fit them and then make predictions.

What we mentioned above is a very simple example. In a real research, features are carefully extracted, where lots of feature engineering tricks are applied. However, work done by Wei Wang et, 2017 [7] is very impressive. They apply an embedding technique which simply denotes user actions with one-hot vectors and piles them in the order of time, forming a two-dimensional array. (One hot vector is a vector where all the elements are 0 except one. The non-zero element is used to to distinguish each other.) This 2-D array then is convolved with a convolutional layer to extract the most important features. Then it is followed by a LSTM layer to grasp the sequential feature of the input. The output is designed to be the user's dropout rate. The difference their work made, is to simplify the manual work done by humans originally, that is, the feature engineering. The model roughly reaches the performance of feature-well-designed machine learning models.

B. CRF

CRF is short for Conditional Random Field, which is a probability model. We don't go deep to its principles in this article. But we are going to mention that in a simple one-dimension CRF model applied into the field of deep learning, it's assumed that the output of the present position or timestep is relevant to the output of last position or timestep.

So in deep learning language models, it's often used with Viterbi Algorithm, which is a dynamic routing algorithm. Specifically, CRF layer takes in the output of the previous layer, which should be a sequence of labels predicted. Each predicted label vector is a n-length vector, where n denotes the number of kind of labels. Then, start from the first position, compute the probability that the predicted label of position 1 convert to other labels, which is also a n-length vector, then add them correspondingly. That should be a n-length vector which denotes the probability of output being each kind of label. After training, we can use the Viterbi Algorithm to compute the best sequence which has the best score computed from the CRF output.

In our model, we will use both self-attention architecture and CRF model with Viterbi Algorithm as well as convolution layers to help improve the performance.

III. MODELS

In this section, we overview the model for prediction.

We designed three network architectures for this problem. The only difference between the first two architecture is whether to use the technique of position embedding. Position embedding emphasizes the position feature of input. The difference of the performances of these two models will tell whether the technique of position embedding is needed in this architecture. And to further explore the effectiveness of the first linear transformation layer of our model, we remove this linear layer and retrain the model to see whether this layer contributes to the accuracy. That's our 3rd model.

A. THE MODEL WITHOUT POSITION EMBEDDING

This model is designed based on the idea of feature enhancement for the proper use of attention and CRF.

Firstly, we treat the statistical features as high-level features and apply a linear transformation layer to map statistical features to high-dimension space. In other words, the first layer of our model is a linear transformation layer, and the goal of this layer is to enhance the feature's ability of representation. Actually, this is an idea originally inspired by word embedding models [26]. In the field of natural language processing, the word was once represented by one-hot vectors. But it turns out that if we represent each word with trainable vectors of high dimensions, the language models will work better. This technique is called word embedding. It's believed that with word embedding, the word vector's representation ability is enhanced, thus bringing about better performance of the model. So this linear transformation layer performs as a feature embedding layer through mapping every feature

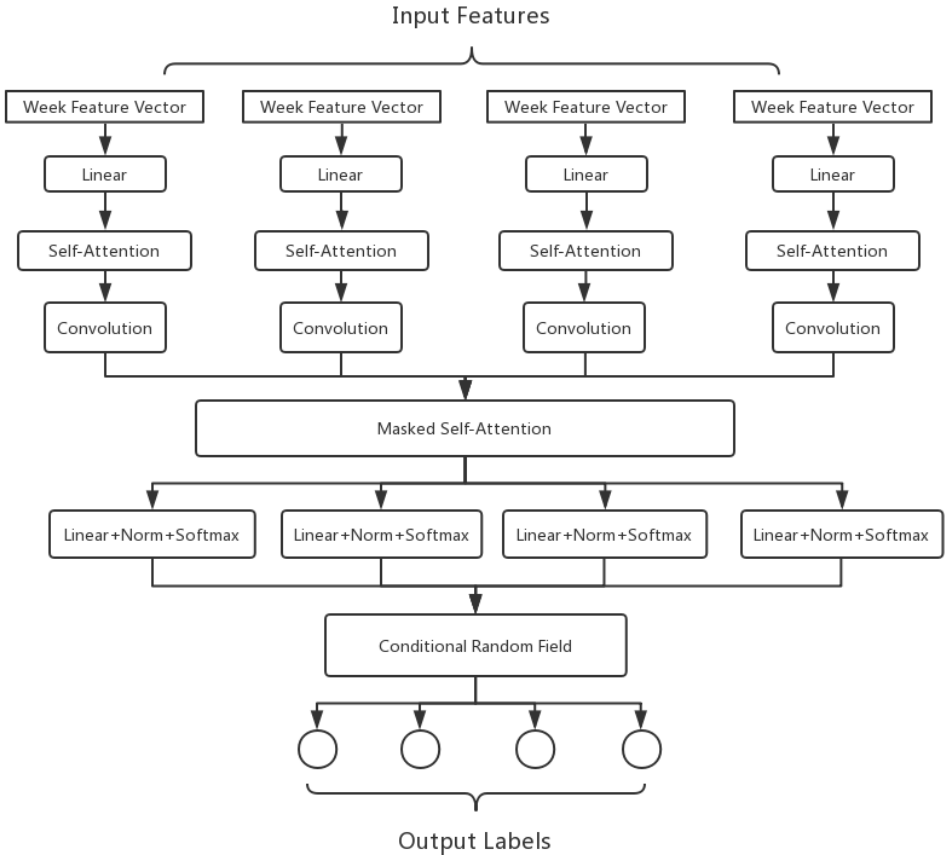

FIGURE 1. Graphical representation of the whole model.

scalar into a high-dimension vector. Consider a single user's dropout prediction. The input is a matrix M of m rows and n columns. M denotes the n features extracted from m time units separately. Then, to obtain a high-dimension representation of each feature, we apply a linear transformation layer with output as a triple (m, n, d), where d is the size of dimensions selected for each feature.

Secondly, we then use attention techniques to reconstruct high-level features from the low-level features obtained, and thus achieve the goal of feature enhancement. After the linear transformation, a self-attention layer is applied to each single feature, with the output as the same shape as the input. The formulation is shown as follows:

One single feature is a vector of several digital numbers like this:

$$feature = \begin{bmatrix} f_0 & f_1 & \cdots & f_{19} & f_{20} \end{bmatrix} \tag{1}$$

self attention first computes the weights of the different units of feature vector influencing each other by using the formulation below:

$$weight = \sigma(feature \cdot feature^T) \tag{2}$$

where $feature^T$ *is the transpose the feature matrix and* $\sigma(x)$ *means to apply every element of the matrix with the following operation:*

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

and then, multiply the weight with the feature matrix:

$$NewFeature = feature \cdot weight \tag{4}$$

The reason why we do not use feed forward neural network is based on the assumption that different low-level features may have relevance to each other and they should be able to influence each other. For example, watching 2 hours of lecture videos in a week is absolutely different from watching them in a single day and rest for the last 6 days. However, when converting to statistical features which is used as input, they are all the same. However, we can use the other statistical features, like frequency, to modify it, thus making the prediction more precise.It is like that the same word in different sentences may have various meanings, but we can always maintain the right one by looking at the context.

We then use convolutional layers to extract useful features and apply attention layer and CRF layer to finally predict the labels. Figure 1 demonstrates our model.

To give an example in a single user's dropout prediction, firstly the input is an array with m rows and n columns, each row of which is a feature vector of n digital numbers in a single time unit. Next, we pass the input through a linear layer to map each feature, which is originally just a single scalar, into a vector of several numbers. Then, we add one self-attention layer to it, which operates on the last dimension of the input matrix. This helps to strengthen the ability of feature representation.

Thirdly, a convolutional layer is applied to squeeze the output of the last layer. Various convolutional layers are used, and their outputs are concatenated to generate the final output of this layer.

With all the features well constructed, predictions are made over them. The first thing to do is to compute the time-unit-level feature vector for future processing. The time-unit-level feature vector should be able to represent the user's action pattern of the specific time unit. And because of convolutional layer's strong ability of pattern recognition, we choose convolutional layers to complete this task. To better represent each feature, we apply three different convolutional kernels to accomplish this job, and by using appropriate paddings, to make their outputs all in the same size. We simply concatenate their outputs together. In this step, we use group convolution and set the number of groups equals to the number of input channels, which should be the number of time units. Otherwise, information of different time units will be mixed together, and data leakage will be caused. The data leakage here means the features used contain something that the model should not have known, which is the information of the time units to predict. The group convolution can handle this problem properly since every filter only operates on a specific channel of the input [23], which is the information of a specific time unit in the problem we're talking about.

For our fourth step, to grasp the potential sequential feature of different time units, we apply another self-attention layer on a time-unit-level representation, which is the output of the convolutional layer.

With the representation for each time unit, we can now think about how to use the sequential features between time units. Since attention mechanism is good at finding the relevance relationship among sequence units, we choose to apply another self-attention layer. To prevent the model from using the information of the time units we are going to predict, we use a trick here. Attention mask is being used to mask the information of the time units which the model is going to predict, which should be like the matrix below:

$$AttentionMask = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5}$$

ATTENTION FORMULATION To explain why the use of attention mask can prevent the model from using the information of the time units to predict, we need to discuss the attention functions. The principle of self-attention is described in the following formulas:

$$Value = Query = Key = InputFeature \tag{6}$$

where in this case, InputFeatures is described as follows:

$$InputFeature = \begin{bmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,83} & x_{0,84} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,83} & x_{1,84} \\ x_{2,0} & x_{2,1} & \cdots & x_{2,83} & x_{2,84} \\ x_{3,0} & x_{3,1} & \cdots & x_{3,83} & x_{3,84} \end{bmatrix} \tag{7}$$

then, weight of different time units influencing each other is calculated as follows:

$$Weight = \sigma(QueryA \cdot Key^T) \tag{8}$$

here, σ refers to apply the following operation to the matrix element-wise:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{9}$$

now, the output of self-attention is calculated as follows:

$$Attention(Query, Key, Value) = Weight \cdot Value \tag{10}$$

So, with all the knowledge how it works, below is the explanation of how we mask the information from the future. We just add the mask to the Weight:

$$MaskedWeight = Weight + AttentionMask \tag{11}$$

And the previous weight is like:

$$Weight = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} \\ w_{1,0} & w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,0} & w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,0} & w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix} \tag{12}$$

where $w_{i,j}$ refers to the weight score of time unit j influencing time unit i

The weight is shown as follows:

$$MaskedWeight = Weight + AttentionMask \tag{13}$$

$$MaskedWeight = \begin{bmatrix} w_{0,0} & -\infty & -\infty & -\infty \\ w_{1,0} & w_{1,1} & -\infty & -\infty \\ w_{2,0} & w_{2,1} & w_{2,2} & -\infty \\ w_{3,0} & w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix} \tag{14}$$

As we can see, the weights of future time units on present time unit becomes negative infinity, which means they no longer contribute information to our present output.

In our fifth step, we use linear transformation, normalization and softmax layers to get the label vector.

Now we explain why these layers are adopted. With the output of the masked self-attention layer, we need to convert it to label vector. The straightforward way to obtain this vector is to apply a linear transformation. Then, to let it be possibility score like, which should be in the range of 0 and 1, a softmax layer is applied. However, when in practice, softmax layer can easily cause gradient vanishing of the optimization course. To handle this problem, an additional normalization layer is applied between the linear layer and the softmax layer [28],[29].

Lastly, after linear transformation, normalization and softmax layer processing, the output is fed to the Conditional Random Field layer, where the labels are predicted sequentially. When the label vector is ready, the CRF layer can be applied.

In the CRF layer, the output of the previous layer is taken as the basis of the prediction, while at the same time, learn various sequence patterns. This enables the prediction of the present time unit not only dependent on the features of the present and previous time units (in the masked self-attention

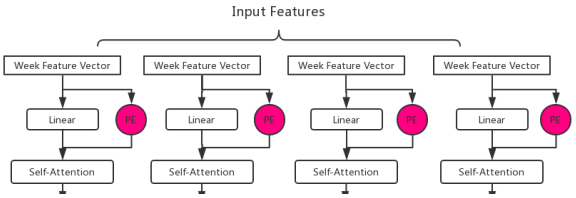

FIGURE 2. Graphical representation of the position embedding layer (*The highlighted "PE" module represents for "position embedding"*).

layer, the features of previous time units are already used), but also to take the predictions of the previous time units into account.

For example, suppose we have the input of CRF layer as the following tuple.

$$input = \begin{bmatrix} l_{0,0} & l_{0,1} & l_{0,2} & l_{0,3} \\ l_{1,0} & l_{1,1} & l_{1,2} & l_{1,3} \\ l_{2,0} & l_{2,1} & l_{2,2} & l_{2,3} \\ l_{3,0} & l_{3,1} & l_{3,2} & l_{3,3} \end{bmatrix} \tag{15}$$

where $l_{i,j}$ denotes the probability score of the ith label being j. In the MOOC dropout prediction problem, there was originally only 2 labels, representing the completion and dropout, respectively. However, in the CRF algorithm, there must be two other labels representing START and STOP flag of the prediction sequence, respectively.

The CRF's idea is to learn another matrix which is called transition matrix as below:

$$transition = \begin{bmatrix} t_{0,0} & t_{0,1} & t_{0,2} & t_{0,3} \\ t_{1,0} & t_{1,1} & t_{1,2} & t_{1,3} \\ t_{2,0} & t_{2,1} & t_{2,2} & t_{2,3} \\ t_{3,0} & t_{3,1} & t_{3,2} & t_{3,3} \end{bmatrix} \tag{16}$$

where $t_{i,j}$ denotes the probability score of label i transiting to label j. If the output is like below:

$$output = \begin{bmatrix} o_0 & o_1 & o_2 & o_3 \end{bmatrix} \tag{17}$$

then the score of the prediction sequence is defined as:

$$score = \sum_{i=0}^{3} l_{i,o_i} + \sum_{i=1}^{3} t_{o_{i-1},o_i} \tag{18}$$

When training, the model will try to make this score as large as possible. Thus, the sequence features of predictions are taken into account.

When predicting, we use Viterbi Algorithm to get the best label sequence, which should be the target label sequence [26].

B. THE MODEL WITH POSITION EMBEDDING

To explore the necessity of position embedding which is always applied with attention architecture, we develop a different architecture. The only difference to the former one is to apply a position embedding technique and the then concatenate the vector embedded from the position of each time unit with the feature vector obtained from the first layer of the model. Such function is performed by a linear transformation layer aiming to obtain the high-dimension representation of each feature.

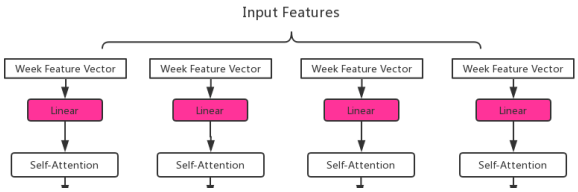

FIGURE 3. Graphical representation of the linear transformation layer (*The highlighted "Linear" module is the layer we're talking about*).

The position embedding is only applied to the 2nd model we design so as to explore whether position embedding is needed in our model. Position Embedding aims to make use of the position feature of the input sequence. In this problem, we add an additional vector to every input feature, which denotes their position in the input sequence. The dimension of the input will thus be bigger than those who don't adopt this operation, but the dimension will be squeezed in the linear layer at the end of the model.

C. THE MODEL WITHOUT LINEAR TRANSFORMATION

With all these models, we still cannot determine whether the application of the first linear transformation layer which maps the original low-level statistical features into a high-dimension space contributes something to the accuracy. Is it just the power of attention? So we remove the first linear transformation layer of the original model, and then train it to see whether the result differs a lot from the original one, thus exploring the effectiveness of this linear layer.

IV. EVALUATION AND ANALYSIS

In this section, we conduct extensive experiments to test the performance of the proposed approaches.

A. EXPERIMENTAL SETTINGS

1) ENVIRONMENT

We implement the algorithms by python using the package scikit-learn, numpy pytorch 1.3.0 and CUDA 10.1. We conduct the experiments on a PC with an Intel Core i5 2.40GHz CPU of 8.00GB memory, a GeForce GTX 1650 GPU of 8.00GB memory and 64bit Windows 10.

2) DATASET AND PROBLEM

The dataset is a subset from the dataset once used in the KDD up 2015. The original dataset contains 39 online courses with 79,186 users participated. The most participated course involves 12,004 users, and we use this part of data for experiments.

The data contains 12,004 users' action information, each pair with a label where 1 denotes dropout, and 0 denotes non-dropout. Each user's information is composed of a series of user actions, details are shown in Table 1.

The time period of this course is roughly divided into 4 weeks. We add each week a label which denotes whether the user has any actions next week. The 4th week's label is the label that the original dataset provides. And our goal is to predict these labels and see whether our model outstands or not.

TABLE 1. The description of user action data.

Time	Source	Event	Object
2014-06-14T09:38:29	server	navigate	Oj6eQgzrdq· · ·
2014-06-14T09:38:39	server	access	3T6XwoiMKg· · ·
2014-06-14T09:38:39	server	access	qxvBNYTfiR· · ·
⋮	⋮	⋮	⋮

3) FEATURE ENGINEERING

As stated above in the Introduction section, we use statistical features as the input of the model we design. For each week, we count the number of different sources, events, objects involved. Operate objects or do events from different sources or may have different meanings, so we count them separately. At last, to somehow grasp some sequential features, we compute a position score for each event type, which is simply the sum of their position indexes. So every week is represented as a vector of $2+7+16+2^*7+22^*16+22^*7=85$ integer numbers.

The models selected as the baseline include a Linear Regression Model, a Support Vector Machine Model, a Logistic Regression Model and a CNN-LSTM model.

The last model mentioned above, the CNN-LSTM model, uses the same form of training data as that used in our originally designed model. The other three uses slightly different data to train, which will be clarified below. For each one of the three machine learning models, it trains 4 independent sub models to predict the 4 labels so that there will not be any data leakage. We need to process the original train data used in our originally designed model. The original input is a matrix of shape [4, 85], each line of which denote the information of one week. We extract the first line, which is the information of the first week, and the first label, to be a training set. Then we extract the first two, then first three, and so on. Finally, we will get 4 training sets, used to train 4 independent machine learning models.

4) MEASUREMENT

Another statement which has been made, is how we define the measure for evaluation. Here we will use the accuracy rate as the only evaluation measurement. In our originally designed models, the output can only be 0 or 1 because of the existence of the CRF layer since CRF takes labels as output. Therefore, the accuracy is the proportion of the correctly predicted labels. For other models, the output is a float number denoting the probability of dropout. The accuracy rate here is the mean of the probability of correct prediction. For example, the output is [0.5, 0.2, 0.7, 0.3], and the label is [1, 0, 1, 0], the accuracy is:

$$(output * label + (pad - label) * output)/4 \tag{19}$$

where

$$pad = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \tag{20}$$

B. COMPARISONS WITH CLASSICAL METHODS

We select classical models including a Linear Regression Model, a Support Vector Machine Model, a Logistic Regression Model and a CNN-LSTM model as he competitor. The experimental results are shown in Table 2.

TABLE 2. The accuracy comparisons.

Model	Epoch	ACC	ACC1	ACC2	ACC3	ACC4
Attn	10	0.8425	0.7338	0.8779	0.9146	0.8438
Attn	1	0.8097	0.7087	0.8629	0.8650	0.8021
CNN+LSTM	60	0.6557	0.6299	0.5809	0.6994	0.7904
CNN+LSTM	100	0.6303	0.5649	0.5790	0.6117	0.7652
CNN+LSTM	10	0.6303	0.5667	0.5830	0.6099	0.7449
svm	-1	0.6348	0.6004	0.5795	0.6020	0.7575
LogR	-1	0.6275	0.6000	0.5750	0.5795	0.7554
LR	-1	0.5501	0.5223	0.5139	0.5106	0.6535

where "Epoch" is the epoch number we choose to train the model. "ACC", "ACC1", "ACC2", "ACC3", "ACC4" are the total accuracy and the separated accuracy of week 1 to 4, "Attn" is the model we designed. "LogR" represents for "Logistic Regression" and "LR" represents for "Linear Regression". When "Epoch" equals to "-1", it means that we continuously train the model until it converges.

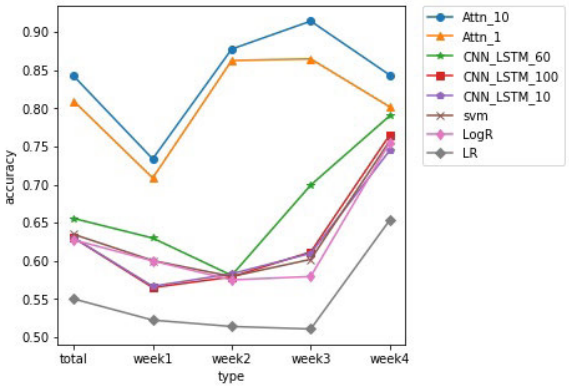

FIGURE 4. Graphical representation of accuracy.

And to make it more clear to see, Figure 4 is the graphical representation of accuracy of different models. The num suffix of the model's name, if any, refers to the number of epochs it trained.

As shown above, we not only evaluate the total accuracy, but also compute the model's accuracy on 4 weeks separately. This make it easier to analyze the performance of different models.

From the result, we can observe that our model reaches the best performance in both total accuracy and weekly separated accuracy, which proves the effectiveness of our designed architecture. This means the proper and reasonable use of attention architecture can significantly improve the performance. And the result also shows us the strong ability of Conditional Random Field to handle sequence problems.

Another observation is that deep learning models usually make less better performance when predicting the first two labels. This because the lack of data. The first two weeks have many all-zero vectors. This is because many dropouts occur in the early period of the course. And many participants may have not participated yet. Contrary to this, the performances of machine learning models are average. This shows that the

TABLE 3. The accuracy of attention models.

Model	Epoch	ACC	ACC1	ACC2	ACC3	ACC4
Attn	10	0.8425	0.7338	0.8779	0.9146	0.8438
Attn	1	0.8097	0.7087	0.8629	0.8650	0.8021
Attn+pos	10	0.8070	0.7192	0.7788	0.9250	0.8050
Attn+pos	1	0.7408	0.7079	0.6796	0.7533	0.8225

where "Epoch" is the epoch number we choose to train the model. "ACC", "ACC1", "ACC2", "ACC3", "ACC4" are the total accuracy and the separated accuracy of week 1 to 4, "Attn" is the model we designed, and "Attn+pos" is its variant with position encoding.

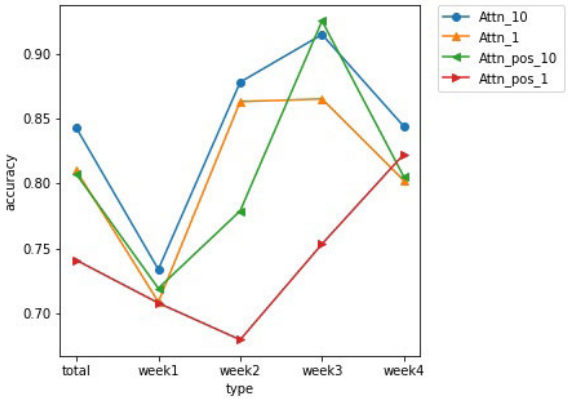

FIGURE 5. Graphical representation of accuracy of attention models.

deep models perform better when the data is richer, while machine learning models may get an average performance. The richness of the data is not only decided by the amount, but also by the quality. Considering MOOC platform will have more and more users, deep models should be the main stream on dropout prediction task. When this problem still exists, it could be solved in a straightforward way. That is to add a weight to the learning rate, so that we can enhance the zero label's impact on the model when training first-week prediction.

C. COMPARISON WITH POSITION EMBEDDING MODEL

Position embedding is proposed with attention technique [23], it adds an additional feature to each input feature vector which denotes their position of the input sequence, so as to make use of the position features. We therefore add a position embedding layer to the model, and then train it to see whether position embedding is needed in this problem. Here are the results.

And to make it more clear to see, Figure 5 is the graphical representation of accuracy of different models. The num suffix of the model's name, if any, refers to the number of epochs it trained.

What impresses us the most is that the model without position embedding, *Attn*, reaches significantly higher performance than model with position embedding, *Attn_pos*. We can find from the table that the performance of *Attn* which has been trained 1 epoch is already better than that *Attn_pos* which has been trained 10 epochs. That means that

TABLE 4. The accuracy of attention models.

Model	Epoch	ACC	ACC1	ACC2	ACC3	ACC4
Attn	10	0.8425	0.7338	0.8779	0.9146	0.8438
Attn	1	0.8097	0.7087	0.8629	0.8650	0.8021
Attn-L	10	0.7479	0.6983	0.6779	0.7854	0.8300
Attn-L	1	0.7434	0.6992	0.6633	0.7867	0.8246

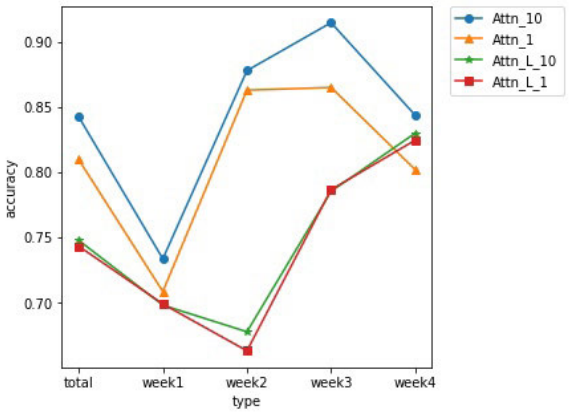

FIGURE 6. Graphical results.

the technique of position embedding may be redundant in this architecture. From our point of view, it is because the position embedding is originally used to denote the position of the word in the sentence, while in our input, the length and the position is always fixed. So the weights are already influenced by the position feature. Or put it another way, the network already memorizes the positions.

D. COMPARISON WITH NO LINEAR TRANSFORMATION MODEL

The first linear transformation layer of our model was inspired by word embedding as we have stated. To further verify its effectiveness, we removed the linear layer and then retrain the model to see the difference.The result can be seen in Tabel 4, where Attn-L is the model of which the first linear layer was removed.

The graphical results are shown in Figure 6.

The results show that, the Attn_L model performs well when the number of epoch of training is 1. However, it converges so fast that its performance rarely varies after another 9 epochs of training. That means it's ability of representation and modeling is far too weaker than those with a linear transformation layer, thus proves the necessity of map the feature into a high-dimension space.

On this issue, we can discuss it with more depth. We think when applying the linear transformation layer to map the original feature into a high-dimension space, we are actually making the model to learn the semantic features of various statistical features. Among the discussion above, we talk about this kind of features using the word relevance, which is exactly a rough definition of semantic relations. In the field of Natural Language Processing, people often use the word co-occurrence to describe or model this feature, but the relevance is obviously more precise. Attention is so good at

maintaining features of relevance, that's why the attention is more powerful in training word embedding models, and why attention can be properly applied into our model.

V. FUTURE WORK

In this article, we are talking about a problem of predicting the dropout label weekly. Here the time granularity is week, but can we predict the dropout label in a smaller granularity? For example, can we predict the daily dropout label? If we can make the prediction result practical, it will bring about great changes in the online education sector.

Another research interest is about feature construction or feature engineering. In the model we proposed, we use artificial designed features, basically a kind of statistical features. There are attempts to use raw log data of the user to extract features using convolutional networks, as we stated in the introduction section. Since attention has strong power of feature extracting, can we use attention architecture to replace it or combine the two together? What's more, all the feature construction method I have mentioned above use either the statistical features like our model, or the sequential semantic features like CNN or LSTM. It's hard for CNN and LSTM to grasp the statistical features reasonably, so we may either make significant changes to the network architecture, or to design and adopt a brand new one. Plus, we can reasonably infer that the network which meets the above requirements can be a step to the Interpretable Deep Neural Network.

VI. CONCLUSION

In this article, we propose a deep learning model for MOOC dropout prediction. The model uses Attention, mainly self-attention, and Conditional Random Field architecture to grasp the features and predict the label. We also apply a method to map a single feature into high-dimension feature space and then use attention and convolution to reconstruct it, so as to obtain better representation of the original feature. We then test the position embedding technique and prove it to be redundant in some kind of sequence problems like MOOC dropout prediction. At last, we test a model without the first linear layer to prove the necessity of the linear transformation. Our model achieves the best performance compared to other four classic models shown by the test result.

The traditional research about dropout rate always predicts one label, i.e., whether students drop out at last. However in this work, we predict a label for each week. This is useful because in real life teachers need to adjust the teaching policy in real time. We believe that finer time granularity prediction should be the future of MOOC dropout prediction and we hope our model can provide some base techniques for this task.

Our study also shows the powerful use of attention. Attention technique is highly useful when dealing with context-sensitive features, and our work provides a strong proof of this. In real life, many problems are context-sensitive, so attention can be applied in many fields. Our work should be an encouragement for people to explore the wide usage of it.

When it comes into details, our work contributes to two aspects. The first is about position embedding. It shows that for fixed-length inputs, position embedding should be unnecessary since the shift of input sequence is not needed. The second is about linear transformation. Experiments proves that it can really enhance the representation ability of the inputs. This is intuitive because a vector can always hold more information than a scalar. We suggest that a linear transformation should always be applied when using statistical features, since such features always hold rich information.

In the future, we plan to combine the statistical features and the sequential semantic features more organically, and try to reach a practical performance when it comes to finer time granularity.

ACKNOWLEDGMENT

This work was partially supported by NSFC grant U1866602, 61772157. This research is based on the dataset provided by KDD Cup 2015, so they would like to acknowledge the organizers of it and XuetangX where the data come from.

REFERENCES

[1] P. Guiney, "Government and sector-level tertiary e-learning initiatives," *Biochimica Et Biophysica Acta*, vol. 1277, nos. 1–2, pp. 93–102, 2014.

[2] M. H. B. N. Azhan, M. Y. B. M. Saman, and M. B. Man, "A framework for collaborative multi-institution MOOC environment," in *Proc. Int. Conf. Internet Things Cloud Comput. - ICC*, 2016, pp. 1–6.

[3] G. Christensen, A. Steinmetz, B. Alcorn, A. Bennett, D. Woods, and E. Emanuel. (2013). *The MOOC Phenomenon: Who Takes Massive Open Online Courses and Why?* [Online]. Available: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2350964

[4] L. Breslow, D. E. Pritchard, J. DeBoer, G. S. Stump, A. D. Ho, and D. T. Seaton, "Studying learning in the worldwide classroom research into edX' s first MOOC," *Res. Pract. Assessment*, vol. 8, pp. 13–25, Jul. 2013.

[5] F. Dalipi, A. S. Imran, and Z. Kastrati, "MOOC dropout prediction using machine learning techniques: Review and research challenges," in *Proc. IEEE Global Eng. Edu. Conf. (EDUCON)*, Apr. 2018, pp. 1007–1014.

[6] J. Liang, C. Li, and L. Zheng, "Machine learning application in MOOCs: Dropout prediction," in *Proc. 11th Int. Conf. Comput. Sci. Edu. (ICCSE)*, Aug. 2016, pp. 52–57.

[7] W. Wang, H. Yu, and C. Miao, "Deep model for dropout prediction in MOOCs," in *Proc. 2nd Int. Conf. Crowd Sci. Eng. - ICCSE*, 2017, pp. 26–32.

[8] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Comput.*, vol. 12, no. 10, pp. 2451–2471, 2000.

[9] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Netw.*, vol. 18, nos. 5–6, pp. 602–610, Jul. 2005.

[10] Y. Wu *et al.*, "Google' s neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*. [Online]. Available: https://arxiv.org/abs/1609.08144

[11] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014, *arXiv:1406.1078*. [Online]. Available: https://arxiv.org/abs/1406.1078

[12] F. Xiong, K. Zou, Z. Liu, and H. Wang, "Predicting learning status in MOOCs using LSTM," in *Proc. ACM Turing Celebration Conf. China - ACM TURC*, 2019, pp. 1–5.

[13] S. Qu, K. Li, B. Wu, S. Zhang, and Y. Wang, "Predicting student achievement based on temporal learning Behavior in MOOCs," *Appl. Sci.*, vol. 9, no. 24, p. 5539, Dec. 2019.

[14] D. Sun, Y. Mao, J. Du, P. Xu, Q. Zheng, and H. Sun, "Deep learning for dropout prediction in MOOCs," in *Proc. 8th Int. Conf. Educ. Innov. Through Technol. (EITT)*, Oct. 2019, pp. 87–90.
[15] B. Jeon, E. Shafran, L. Breitfeller, J. Levin, and C. P. Rosé, "Time-series insights into the process of passing or failing online University courses using neural-induced interpretable student states," in *Proc 12th Int. Conf. Educ. Data Mining*, Jul. 2019, pp. .
[16] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 6645–6649.
[17] T. Mikolov, T. Mikolov, M. Karafiát, and L. Burget, "Recurrent neural network based language model," in *Proc. Interspeech, Conf. Int. Speech Commun. Assoc.*, Makuhari, Japan, Sep. 2015, pp. 1–4.
[18] Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding," in *Proc. IEEE Workshop Autom. Speech Recognit. Understand. (ASRU)*, Dec. 2015, pp. 167–174.
[19] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu, "CNN-RNN: A unified framework for multi-label image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2285–2294.
[20] T. Stafylakis and G. Tzimiropoulos, "Combining residual networks with LSTMs for lipreading," 2017, *arXiv:1703.04105*. [Online]. Available: http://arxiv.org/abs/1703.04105
[21] K. Xu, H. Wang, and P. Tang, "Image captioning with deep LSTM based on sequential residual," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Jul. 2017, pp. 361–366.
[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017, *arXiv:1706.03762*. [Online]. Available: https://arxiv.org/abs/1706.03762
[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*. [Online]. Available: http://arxiv.org/abs/1810.04805
[24] T. Sinha, P. Jermann, N. Li, and P. Dillenbourg, "Your click decides your fate: Inferring information processing and attrition behavior from MOOC video clickstream interactions," 2014, *arXiv:1407.7131*. [Online]. Available: http://arxiv.org/abs/1407.7131
[25] E. J. Kandola, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Stud. Fuzziness Soft Comput.*, vol. 194, pp. 137–186, Feb. 2006.
[26] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, New York, NY, USA: Curran Associates Inc, 2012, pp. 1–9.
[27] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*. [Online]. Available: http://arxiv.org/abs/1502.03167
[28] J. Lei Ba, J. Ryan Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*. [Online]. Available: http://arxiv.org/abs/1607.06450
[29] G. D. J. Forney, "The viterbi algorithm," *Proc. IEEE*, vol. 61, no. 5, pp. 268–278, Mar. 1993.

SHENGJUN YIN was born in 1978. She received the master's degree in software engineering from the Harbin Institute of Technology. She is currently pursuing the Ph.D. degree in computer science and technology. Her research interests include big data of education and big data quality.

LEQI LEI was born in 1999. He is currently pursuing the bachelor's degree in computer science and technology with the Harbin Institute of Technology. His research interests include deep learning algorithm, pattern recognition, and natural language processing.

HONGZHI WANG (Member, IEEE) is currently a Professor and the Ph.D. Supervisor with the Harbin Institute of Technology. His research interests include big data management, data quality, graph data management, and web data management. He was awarded the Microsoft Fellowship, Chinese Excellent Database Engineer, and the IBM Ph.D. Fellowship.

WENTAO CHEN was born in 2000. He is currently pursuing the bachelor's degree in computer science and technology with the Harbin Institute of Technology. His research interests include big data of education and pattern recognition.