Automatic Feature Engineering From Very High Dimensional Event Logs Using Deep Neural Networks

Kai Hu
Yahoo Research
kaih@verizonmedia.com

Joey Wang
Yahoo Research
joeywang@verizonmedia.com

Yong Liu
Yahoo Research
yongl@verizonmedia.com

Datong Chen
Yahoo Research
datong@verizonmedia.com

ABSTRACT

As communication networks have grown, event logs have increased in both size and complexity at a very fast rate. Thus, mining event logs has become very challenging due to their high variety and volume. The traditional solution to model raw event logs is to transform the raw logs into features with fewer dimensions through manual feature engineering. However, feature engineering is very time-consuming, and its quality is highly dependent on data scientists' domain knowledge. Furthermore, repeatedly preprocessing event logs significantly delays the scoring process, which must scan all items in the logs.

In this paper, we present our recent study on mining high-dimensional event logs using deep neural networks. We propose a Midway Neural Network (MNN) to avoid both manual feature engineering and the re-preprocessing of event logs. MNN embeds an input feature vector from a particular time window into a dense representation and memorizes these midway representations for incremental training and prediction. The experimental results demonstrated that the proposed method minimized human intervention, decreased the time for training and scoring, and decreased the memory and storage costs while maintaining a similar modeling performance compared to traditional solutions. We hope that our insights and knowledge can inspire colleagues who are working on similar problems.

KEYWORDS

Automatic Feature Engineering, Mining Event Logs, Deep Neural Networks, Incremental Scoring

1 INTRODUCTION

The most commonly used raw event logs in industry are appended lists that contain an event ID, event attributes and a time-stamp. For example, in online advertising, billions of users trigger millions of types of events every millionth of a second in online services every day. These events may include a user's activities on web pages, mobile Apps, search engines, communication tools, smart devices, videos, ad creatives, etc. Each event may only have a few attributes, but the cardinality of some attributes can be very high. For example, the cardinally of web page URLs and search queries can be in the hundreds of millions. Moreover, the number of unique user IDs in the event logs can be in the billions, and the number of events grows by millions every second. The high variety and volume in these event logs results in significant challenges for mining them.

The situation in Yahoo! is even more challenging. Yahoo! has large-scale event log systems that record user's online behaviors in various products, including Yahoo! Finance, Yahoo! Sports, Yahoo! Mail, Yahoo! Search, AOL.com, Huffpost, Tumblr, Flurry, Gemini, BrightRoll, etc. Though data is anonymized by removing identifiable information, such rich event logs can provide information on users' demographics, personal interests, recent purchase, and travel plans from different aspects, and therefore are very valuable for Yahoo!'s advertising business. However, analyzing and modeling these event logs is a nightmare for Yahoo!'s data scientists and machine learning engineers because the event logs are not standardized. Different event logs are recorded in different formats, stored in different datasets, and maintained by different pipelines.

The traditional technique for modeling raw event logs is to transform the raw logs into features with fewer dimensions through manual feature engineering. Popular feature engineering techniques include clustering sparse attributes into categories, accumulating events over time with certain time-decay algorithms, regression-based trending, periodical signal extractions, and others [7, 17, 19]. Feature engineering dramatically reduces the dimensionality of the event streams, resulting in a feasibly solvable mining problem for most of statistics and machine learning models.

However, manual feature engineering cannot work efficiently for the large-scale data from event logs found in industry. It is challenging to apply manual feature engineering because it relies too much on the scientists' domain knowledge. The extremely steep learning curve to understand the complicated raw data significantly impairs scientists' productivity, thereby creating a bottleneck as both the volume and the dimensionality of data grows exponentially in industry. Moreover, it is difficult to prove that the manually engineered features are optimized for a complicated application because the process is usually too time-consuming and too costly to operate over thousands of iterations.

DLP-KDD'19 , August 5, 2019, Anchorage, AK, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6783-7/19/08…$15.00
https://doi.org/10.1145/3326937.3341262

An alternative solution is Deep Neural Networks (DNNs), which have been demonstrated to be powerful tools for automatically exploring feature interactions in many types of data including images [10, 15], video/audio [1, 11], nature language text [5, 18]. A typical DNN organizes its neurons into layers, and each layer consists of one or more neurons. A neuron transforms an input feature vector or tensor with a weight vector, which is usually the same size of the input vector, into a single output, and passes the output to neurons in the next layer. The weight vectors in each neuron can be optimized using training examples through a back-propagation process [8].

Unfortunately, DNN-based feature interaction exploration is also inefficient or even infeasible due to the high dimensionality of the event logs. For example, let us assume that the daily event data has over 1 million dimensions after one-hot encoding for categorical attributes. Since feature interactions can be measured automatically, input feature vectors are simply concatenated across the timeline without manual feature engineering. However, directly concatenating event data in a 30-day window with daily granularity results in over 30 million dimensions. Assuming there are 1000 neurons in the first layer, which is quite a reasonable size for such a high-dimensional input, the DNN would need at least 30 billion parameters. These parameters would usually be stored in memory, resulting in a very high hardware cost in the present day. Note that a finer time granularity, e.g. hourly, demands even more number of total parameters in magnitudes.

Another main challenge of DNN-based feature interaction exploration is the cost of the scoring process. Unlike the training process, which can be scaled down by sampling the data, the scoring process must scan all input vectors created from event logs. If we assume that we have one billion IDs and each ID carries 30 million features, the total data size is more than $30 \times 10^6 \times 10^9 = 3.75$ PB. The processing and storage of such a large data place a huge engineering burden on the data infrastructure, and therefore limit the scoring frequency.

In this paper, we would like to share our experience and insights on mining high dimensional event logs with the hope that it may be useful to colleagues facing similar problems. We propose a large scale machine learning architecture, called a Midway Neural Network (MNN), to address the above challenges in automatic feature engineering from raw event logs. It takes a raw event stream as the input and returns relatively small features vectors that can be used to build a specific type of prediction model for which traditional feature operations—such as feature clustering, aggregation, etc.—are fully automated, thereby minimizing human intervention. In specific, an MNN consists of several encoding DNNs, one prediction Network (NN) and one Midway layer. An event log is first bucketized into a sequence of unit time windows. In each unit window, events are aggregated into a raw feature vector (with 10^6 dimensions in the above example). The encoding DNN is used to encode the raw feature vector of the latest time window into a Midway embeddingr vector (with, e.g., 200 dimensions). The Midway layer is used to remember the entire sequence of Midway vectors and concatenates them into a longer output vector. The prediction NN is then used to predict the label of the Midway output. The advantages of MNN in automatic feature engineering using event log data are listed below:

- The MNN structure enables the exploration of feature correlations both within and across time windows without human intervention;
- It dramatically reduces memory and storage usage in the scoring process by encoding features from previous time windows into low-dimensional midway vectors;
- It dramatically accelerates the scoring computation by preventing prepossessing the raw event logs a second time;
- Both the training and scoring processes of MNN can be performed incrementally.

The remainder of this paper is organized as follows. Section 2 describes the related prior work. The details of the proposed midway neural networks are presented in Section 3. Experimental results on five real event log mining tasks are presented in Section 4. Conclusions are drawn in Section 5.

2 RELATED WORK

Recently, deep neural networks have attracted the attention of researchers interested in the implicit measurement of feature interactions due to their high capacity and end-to-end training scheme. Various neural network structures have been proposed. [29] and [20] discovered an interaction between multi-field categorical data using two different DNN structures, called FNN and PNN. Deep&Cross [28] and xDeepFM [16] utilized the outer product of the features at the bit- and vector-wise levels respectively. Deep crossing [24], Wide&Deep NN [4], and DeepFM [9] applied a fully connected feed-forward neural network to field-embedding vectors to model high-order feature interactions. However, none of the listed investigations can achieve incremental scoring, leading to extremely expensive scoring processes in terms of computation and memory costs, especially for the large-scale data from event logs found in industry.

To the best of our knowledge, there is no existing research that has achieved practical automatic feature engineering using large-scale event logs. Due to the sparsity and scability problems in both representation and optimization, most of the existing research on event-log mining has focused on the traditional technique of manual feature engineering. For example, [2, 23] explored the feasibility of identifying signature patterns directly from event logs. However, the framework proposed in that work relied extensively on the use of domain knowledge to assist with feature extraction and selection. Massimiliano de Leoni, et al. employed a data-clustering algorithm to analyze large-scale textual event logs based on word frequency [27]. Though their solution is novel, its application is limited because it can only handle several specific data types.

3 METHODOLOGY

In this section, we propose an effective approach, called Midway Neural Networks (MNNs), for automatic feature engineering from very high-dimensional event logs. By breaking down a event sequence into chunks with small, fixed-width time windows, which are called unit time windows, MNNs apply preprocessing and automated feature generation on the incoming data stream in real time or in a mini batch fashion. Fig. 1 shows the structure of a typical MNN. An MNN consists of several encoding DNNs, one prediction NN, and one Midway layer. The information in each

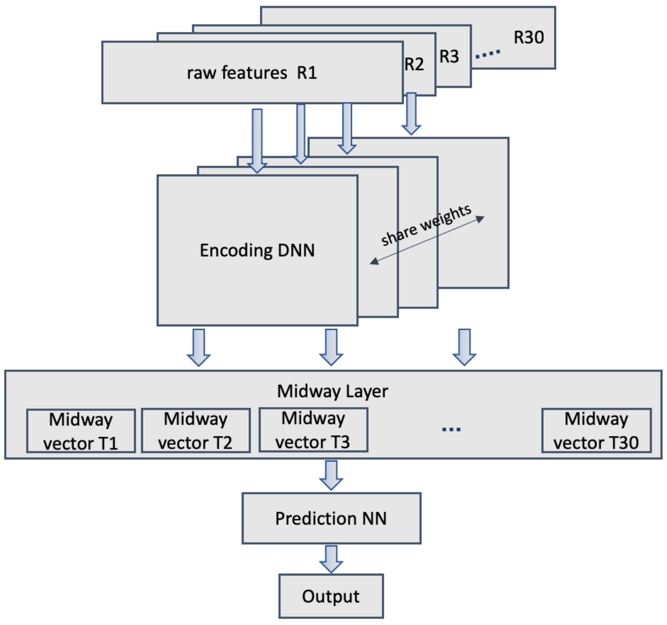


Figure 1: Architecture of a typical midway neuron network.

unit time window is collected and formatted into a unit feature vector. The encoding DNNs are used to encode the unit feature vectors into midway vectors with a lower dimension (e.g., 200), and the prediction NN is trained to predict the given labels using the concatenated midway vectors. The Midway vectors are stored in a distributed file system or index engine so that they can be used by any application in the future.

3.1 Preprocessing of Event Logs

Before preprocessing can occur, it is necessary to define the unit time window and the lookback time window. During the preprocessing of the event logs, the raw events in each unit time window are formatted into a so-called "unit raw feature vector." Events outside of the lookback time window are neglected. All numerical features are simply accumulated in each unit time window. All categorical attributes are transformed into sparse representations using a one-hot encoding scheme—which could be a multi-hot scheme aggregated from multiple events—and then concatenated with other numerical and keyword-embedding features. All individual features are normalized using the min-max method [13] to remove the bias, to scale the value differences of the different features, and to keep the sparsity of the original features.

The unit raw feature vector is expected to be sparse and high-dimensional, especially when there are categorical attributes that have high cardinalities, such as zip codes, keywords in queries, interest tags, etc. These sorts of high-cardinality categorical attributes are common in industry. Concatenating the unit raw feature vectors from a event sequence into a primary input results in an even larger vector, which is either computationally inefficient or unfeasible to build a model from. More importantly, modeling algorithms tend to neglect sparse features due to their low coverage. Without appropriate feature engineering designs, sparse features can have significant negative impacts on the model performance if sparse features are dominant in the feature vectors.

Preprocessing is a very expensive step in terms of the computation and storage of the resulting feature vectors. It is one of the most painful bottlenecks when mining raw event logs. Preprocessing includes many complicated string multiplication operations, which are computationally inefficient. Furthermore, the event logs in industry are very large (e.g., tens or hundreds of terabytes) and are usually stored in different databases or distributed file systems. Hence, both the data transfer and disc-loading of event logs are very time-consuming, which results in a heavy engineering burden on the data infrastructures, especially in the scoring stage. One option to avoid preprocessing the raw event logs multiple times during scoring is to store all the unit raw feature vectors from previous windows. However, this is a very space-consuming solution given the large size of each unit raw feature vector.

3.2 Midway Neural Network (MNN)

Although feature engineering is time-consuming and error-prone, it is commonly used in conventional modeling processes because it effectively reduces the feature dimension, lowering memory and computation cost and avoiding overfitting;

By using a specific deep neuron network structure, our proposed method, called a Midway Neural Network (MNN), allows automated feature engineering (thereby avoiding manual feature engineering operations) with comparable modeling performance to manual feature engineering (Fig. 1). In the MNN system, the unit raw feature vectors of each unit time window are concatenated into a long vector ordered by their time sequence. An empty unit raw feature vector is allowed here if no events are observed in a unit window. The concatenated vector contains all information recorded in the event logs in the lookback time window (R1 - R30). The vector count in the merged vector is equal to the number of unit time windows in the lookback time window. The dimension of merged feature vectors varies with the time granularity of the unit time windows and the size of the lookback window. An MNN takes the merged vector (ordered by time sequence) as the primary input, which avoids any information loss in the training stage. After that, feature engineering operations—such as feature selection, feature clustering, feature aggregation, feature bucketization, etc.—can be implicitly executed and automatically optimized using MNNs.

3.3 Encoding DNNs

The encoding DNNs are used to generate the low-dimensional Midway vectors, which serve as encoded representations of raw inputs. Each encoding DNN takes one unit raw feature vector from the merged feature vectors as the input. To reduce the model complexity, the encoding DNNs corresponding to all of the unit windows in an MNN share the same structure and parameters. Fig. 2 illustrates the structure of the encoding DNNs. Because a simple conversion of the categorical features with high cardinality into a one-hot vector significantly increases the size of the model, we encode the categorical features separately before merging them with other numerical features. This inner structure is expected to dramatically reduce the number of parameters in the encoding DNNs if one-hot features are dominant in the raw feature vector, which is common in event

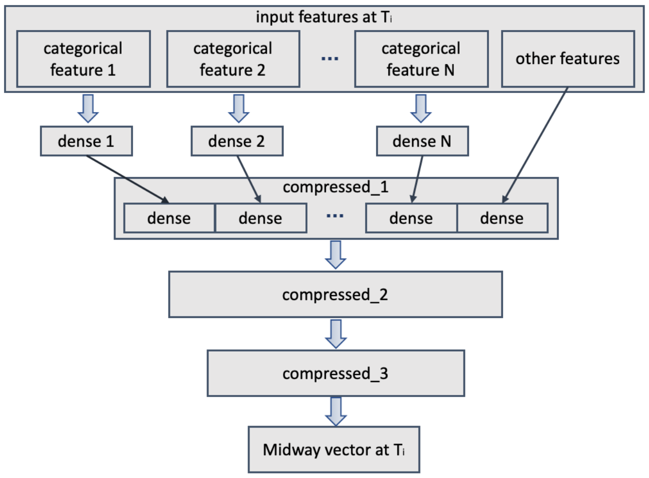


Figure 2: Architecture of a typical encoding DNN.

logs. Note that the interaction between the categorical features and other numerical features can still be detected and optimized in the following fully connected layers.

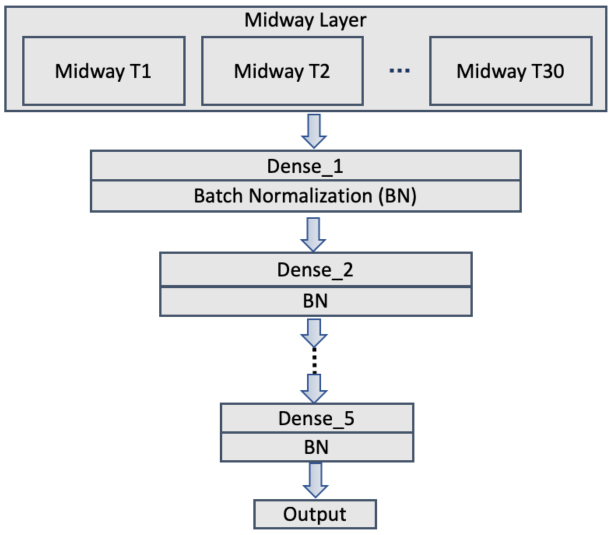
One interesting question is how to determine the embedding dimension of a categorical feature. In one-hot encoding, a categorical feature is transformed to multiple binary features, but only one of them has a value of 1. Therefore, we can always perfectly encode categorical features using a binary vector with length $\log_2 k$, where k is the number of categorical cardinalities. The embedding dimension can be further reduced because the float-encoding embedding vectors actually have much more information capacity than binary vectors. Hence, $\log_2 k$ serves as the upper limit of the embedding dimension of a one-hot encoded categorical feature with k cardinalities. For multi-hot features, the estimation is much more complicated. However, considering the additional information volume provided by float-encoding vectors, we simply take $\log_2 k$ as the embedding dimension for the categorical features in our system.

3.4 Midway Layer

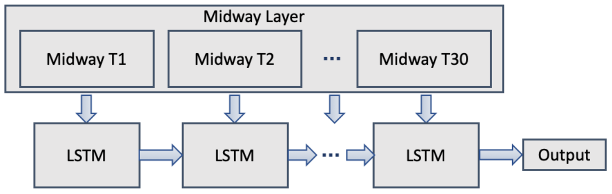
In an MNN, the midway layer is a relay layer between the encoding DNNs and the prediction NN. The midway layer consists of a sequence of low-dimensional midway vectors, which are generated by the encoding DNN and ordered by their time sequence. All midway vectors have the same size. The number of midway vectors is equal to the number of unit time windows in the lookback time window.

Though midway vectors feature information loss from the original unit raw features, they are necessary for the following reasons:

(1) The raw input vectors of the MNN are very high dimensional, especially when the time granularity is fine or the total lookback window is long. Hence, the parameter count may become unacceptably large if the NN is applied directly to the raw input.
(2) The raw feature vectors are very sparse. An event may occur only once or twice in the total lookback time window. Modeling of high-dimensional sparse feature vectors has a high risk of overfitting. Common solutions such as feature selection and regularization also feature additional information loss.
(3) A Midway vector has a much smaller size than a unit raw feature vector. We can preserve the midway vectors from all previous unit time windows to avoid repeatedly preprocessing the raw event logs during scoring.



(a) Prediction DNN



(b) Predition RNN

Figure 3: Illustrations of two prediction NN strutures. (a) DNN; (b) RNN.

3.5 Prediction NN

The prediction NN achieves supervised learning using the midway layers as inputs. Note that different prediction NNs can be structured or trained separately for different objective. The Midway vectors in the midway layer are aligned according to their time sequence. Fig. 3 illustrates two implementations of prediction NN. The first one (Fig. 3a) is based on DNN structures. It consists of multiple fully connected layers that are stacked together. A batch-normalization layer is inserted after each layer to normalize their inputs and to fight the internal covariate shift problem [12]. The inner structure is adjustable according to the prediction tasks. An alternative implementation of the prediction NN is based on the recurrent neural network (RNN), whose structure is illustrated in Fig. 3b. The prediction RNN consists of T long short-term memory (LSTM) units [22], where T is the number of midway vectors in the midway layer. Each LSTM unit takes one midway vector as a input. All LSTM units are chained together, and the output is observed at the last unit connected to the latest midway vector. There are two concerns regarding the RNN solution:

(1) Our experiment results report that the performance of DNN structures is better (Section 4.4).
(2) The RNN structure has longer training and scoring time.
(3) In prediction RNN, the data flow must pass through all unit time windows sequentially, whereas in the proposed prediction DNN, the midway vectors are loaded in parallel and the scoring time can be easily adjusted by changing the number of stacked fully connect layers.

Hence, we prefer to use the stack of fully connected layers (Fig. 3a) as the prediction NN because this structure provides better performance, more flexibility, and shorter training and scoring times.

3.6 Training of MNN

Before training, the raw event logs are formatted as a sequence of unit feature vectors. All feature vectors are normalized and then concatenated together. Though the merged vectors are very high-dimensional, their storage size is still acceptable as long as they are stored in a sparse-vector format. The MNN is trained using mini-batch gradient descent. In each iteration, a batch of merged vectors are expanded to the dense format, and they are the primary inputs for the MNN training. Back-propagation is executed to update the parameters in the whole system. Because the encoding DNNs share parameters and have the same structure, we can replace fully connected layers in the encoding DNNs with convolution layers to simplify the model structure.

To accelerate the training, the parameters in the MNNs are initialized using several pre-trained neural networks. First, we establish a symmetric auto-encoder whose top and bottom halves have the same structure as the encoding DNNs. We sample a number of unit raw feature vectors from different unit time windows and use them as inputs to train the auto-encoder. After that, a DNN/RNN whose structure is the same as the prediction NN is established and trained using the compressed vectors generated by the auto-encoder. The parameters in the auto-encoder and the second NN are used to initialize the encoding DNNs and the prediction NN, respectively. We found that this initialization methodology can result in 2- to 3-fold decrease in the training time.

3.7 Scoring of MNN

Unlike the training process, which can be scaled down by sampling the data, the scoring process must scan all of the event logs to create the raw feature vectors. Thus, a large amount of time is required to preprocess the raw event logs, resulting in a heavy burden for the data infrastructure system. Since the scoring process is run much more frequently than the training process, it is the main bottleneck when mining event logs outside of feature engineering.

Fig. 4 illustrates the diagram of the MNN scoring process. An MNN reduces the scoring time by incrementally scoring the latest unit feature vector together with the midway vectors from the previous unit time windows. Let us assume that there are t unit time windows in a lookback window. The input for an MNN scoring process consists of two parts: $T_{2:t}$ midway vectors that encode the

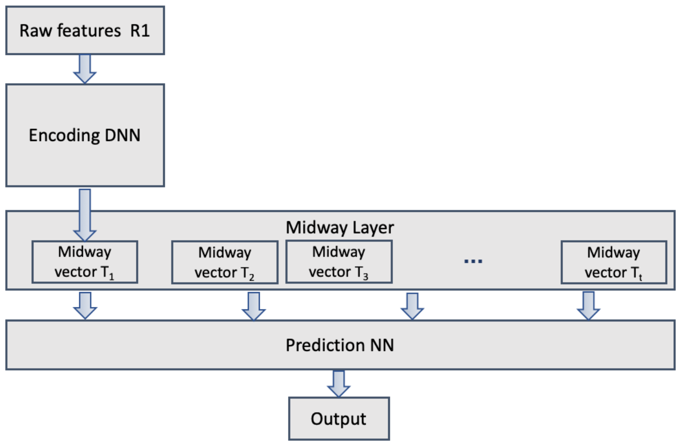


Figure 4: MNN scoring process.

events in previous $t - 1$ unit time windows and the unit raw feature vector R_1 that records the most recent events. The MNN first encodes the R_1 into a new Midway vector T_1 and then concatenates the $T_{1:t}$ vectors into a single long vector in order to predict the final score. Each new midway vector is stored once it is created. Because of their low dimension, the storage cost of the midway vectors is acceptable. Furthermore, they can be reused in the subsequent incremental scoring process, which can prevents prepossessing the raw event logs a second time. The reuse of the midway vectors dramatically reduces the costs of preprocessing, encoding, and storage as well as the time required to load the data. The cost reduction becomes even more significant if we have a longer lookback time window and a finer unit time granularity. This incremental scoring scheme enables the real-time scoring/prediction of new events.

3.8 Advantages of MNNs

In this subsection, we summarize the five main advantages of MNNs:

- **Quick training and deployment**. The automatic feature engineering and the standard structure of MNNs can dramatically improve the productivity of data scientists and, therefore, reduce the time required for model training and deployment.
- **Efficient scoring**. The MNN achieves efficient scoring by reusing the midway vectors and avoiding the re-preprocessing of the raw event logs.
- **Incremental learning**. The proposed MNN structure allows incremental learning, which is significant in industry. Let us assume that the encoding DNNs are invariant. We only need to re-train the prediction NN if a new event occurs or if updating is required.
- **Modest resource requirements**. By using low-dimensional midway vectors, MNNs avoid direct operations on the raw event logs, thereby reducing the memory and storage cost of the whole system.
- **Time-sensitive learning**. The proposed MNN can identify time-related patterns such as periodicity and time decay in the high-dimensional log data.

4 EXPERIMENTS AND EVALUATION

Estimating a user's click-through rate (CTR) is a fundamental task in the personalized advertising and recommendation industry [21]; thus, we use CTR estimation as the working example to evaluate the effectiveness of the proposed MNNs. We investigate five prediction tasks related to CTR estimation using large-scale real-word event log data from Yahoo. The proposed approaches are compared with a baseline solution involving a set of complicated feature engineering operations manually optimized by our data scientists. Two types of prediction NN structure, DNN and RNN, are implemented and compared. The proposed MNNs successfully achieve a comparable or even better prediction performance in terms of AUC in all cases. Considering the significant advantages in the areas of automatic feature engineering and incremental scoring, the proposed MNN is demonstrated to be a successful mining solution for high-dimensional event logs.

4.1 Experiment Data

To verify the performance of the proposed MNNs, we conducted experiments on a variety of event logs that covered almost all important products of Yahoo!. These event logs record information on user behavior (e.g., page view, click, conversion, log-in time, etc.) and profile (age, gender, location, etc.) in these products. The total size of the event logs was more than 100 TB and included information from 1.1 billion Yahoo! active device/browser IDs. Data is anonymized by removing all identifiable information. Since all products operate independently, all event logs are stored in different formats in different databases. Though these precious data resources equip Yahoo with a broad vision of users interests and hobbies, feature engineering is extremely difficult because nobody has sufficient knowledge of the data on so many platforms. Hence, it is essential to design a standard methodology to realize automatic feature engineering from raw event logs.

4.2 Experiment Setup and Preprocessing

The event logs used in the experiment contained various information in different data types. Our features included a series of numeric and binary attributes, such as number of impressions, duration time of a page view, revenue, gender, etc., as well as a series of categorical attributes, such as segment IDs, ZIP codes, search keywords, app category, URL domains, etc. In preprocessing, categorical attributes were one-hot encoded. Event logs were bucketized into a sequence of unit time windows. Events in each unit time window were aggregated in the same manner. The entire preprocessing pipeline was implemented in the Hadoop Framework [6].

Due to the high cardinalities of the categorical features, each unit raw feature vector contained over 1 million features, but only one thousand of them were nonzeros, i.e., approximately 99.9% of the features in the raw feature vector had a value of '0'. In the experiment, the lookback time window was set to one month (30 days) and the unit time window was set to one day. Hence, there were 30 unit raw feature vectors that were concatenated into the merged vector. After concatenating the 30 unit feature vectors, the merged feature vectors had $30 \times 1M = 30$ million features. By storing the merged feature vectors in the sparse vector format, we reduced the total size of the stored data to approximately 4.5 TB.

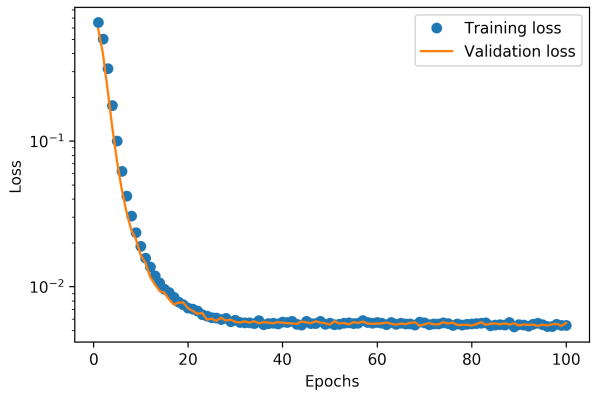


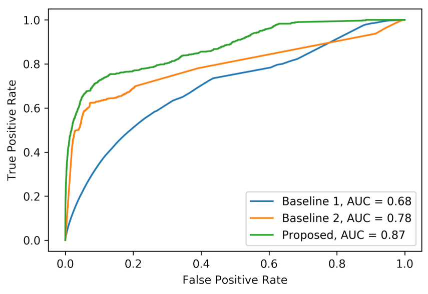
Figure 5: A typical learning curve of the proposed MNNs.

The experiments are designed to use these event logs to predict users' CTR in five different categories of ad contents. A set of binary 1/0 true labels is given to indicate the click/non-click of ad impressions in each category. We use mean square error between true labels and predicted CTRs as the loss function for the MNN training. The category names are not given due to business confidentiality. CTR prediction is important to the online advertising industry because CTR is one of the key metrics for evaluating the marketing effects of advertising. The following experiments were carried out in a virtual machine with 64 vCPUs, 8 GPUs, and 448 GB memory.

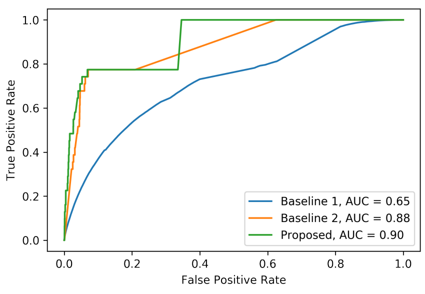
4.3 Model Training

Once a merged feature vector has been transformed to the dense form for the training and scoring stages, it is too large to be used for direct training, even with present-day computing power. Feature engineering is a common solution that reduces feature dimensions via a number of aggregation and clustering operations. However, feature engineering is very time-consuming and its quality is highly dependent on data scientists' data intuition and their understanding of the raw data. Thus, it has become one of the main bottlenecks of large-scale machine learning in industry. One motivation of MNNs is to release data scientists from such tedious work.

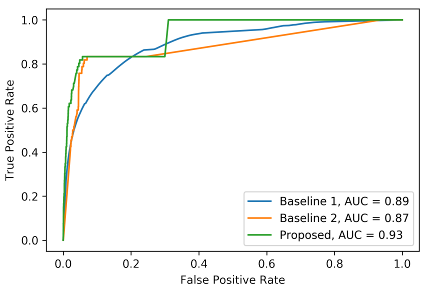
In the experiment, the total 1.1 billion browser/device IDs in the dataset were split into random training, validation, and test datasets with respective the proportions of 0.4, 0.1, and 0.5. The training dataset was used as the primary inputs to the MNNs during training. We first used the encoding DNNs to generate low dimensional midway vectors that represented the raw feature vectors. The structure of the encoding DNNs is shown in Fig. 2. All encoding DNNs share the same structure and parameters. Because the one-hot features were dominant in the raw feature vectors, we encoded the categorical features separately before merging them to other numerical features. We took the upper limit, i.e., $\log_2 k$, as the embedding dimension of each categorical feature, where k was the categorical cardinality. Hence, over 1 million one-hot sparse features were finally compressed to just 50 dense features. After merging with other numerical features, we had 10893 features at Layer "compressed_1". The remainder of the encoding DNNs was



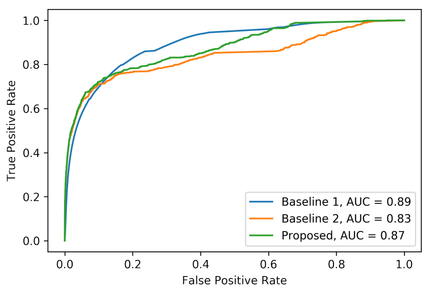
(a)



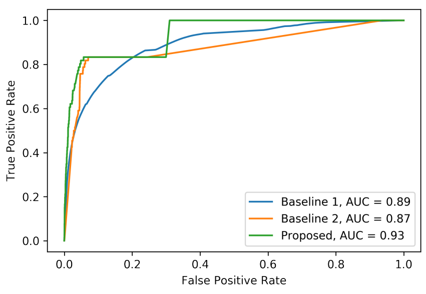
(b)



(c)



(d)



(e)

Figure 6: Performance comparison between Baseline 1, Baseline 2, and the proposed MNNs in terms of ROC curves and AUCs.

two sequential, fully connected layers. The dimensions of these two layers were 2000 and 1000, separately. All of the intermediate layers used rectifier activation functions. The last layer, which was the midway layer, used a linear activation function. In our experiment, the dimension of the midway vectors was 200. The parameters in the encoding DNNs were initialized with a pre-trained auto-encoder with the same structure to accelerate the training.

The prediction NN had two different implementation solutions: DNN and RNN. The DNN solution consisted of five fully connected layers (Fig. 3a). Batch normalization layers were inserted after each fully connected layer. The RNN solution consisted of a chain of 30 LSTM units (Fig. 3b). Both solutions took the midway layer as the primary input, which is a time-wise concatenation of the midway vectors. The usage of the midway vectors reduced the feature volume by 80%. The parameters in both the DNN and RNN implementations can be initialized with a pre-trained DNN/RNN with the same structure.

The MNN structure is implemented using Tensorflow. We choose Adam [14] and mean squared error (MSE) as the optimizer and loss, respectively. Because clickers are a minority in the entire population of the training set, we balanced the weight of clickers and non-clickers with class weights that were inversely proportional to their frequencies. If no loss improvement was observed for at least three consecutive epochs, the learning rate was reduced by half (adaptive learning-rate adjustment). The training process was terminated when no loss improvement was observed after 5 consecutive epochs (EarlyStopping). Otherwise, the training was terminated after 100 epochs. It took approximately 7.3 minutes to finish a single epoch, where each epoch scanned two million users in the training data set. Thus, the total training time was approximately 12 hours.

4.4 Performance Evaluations

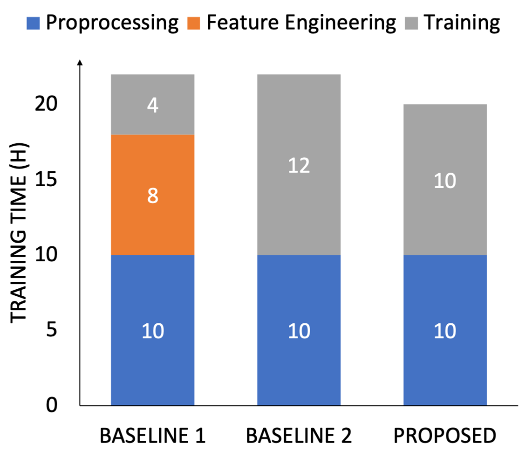
The performance of the proposed MNNs is evaluated using five real CTR-prediction tasks from Yahoo!. Note that We propose the use of a stack of fully connected layers (Fig. 3a) for the implementation of the prediction NN. The results are compared to two baseline methods.

The first baseline method is the modeling methodology that is running in the current production line. Raw event logs are transformed into features with relatively lower dimensions through a set of complicated manual feature engineering operations. It takes three data scientists more than three months to optimize the feature engineering strategy. A distributed logistic regression model is built using SPARK [25] with 500 work nodes, 5 TB of memory and 2500 cores. Random Forest [26], Gradient Boosting Decision Tree [3], or other more complicated algorithms are not used in the production line due to the concerns of computational costs.

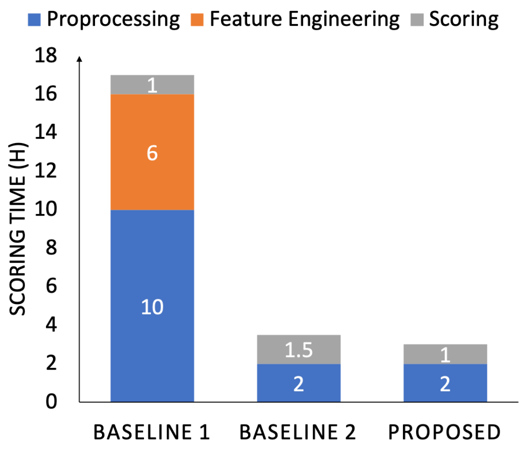
The second baseline method is to replace the stacked fully connected layers in the prediction NN with a LSTM chain with length T, where T is the number of midway vectors in the midway layer.

Fig. 5 shows the learning curve of the training process. No overfitting is observed in the training stage because the training and validation loss decrease monotonically at the same pace.

Fig. 6 illustrates the ROC curves and AUCs of the five prediction tasks. In Experiment (a) and Experiment (b), the performance of the proposed method is significantly better than that of Baseline 1. A better design of feature engineering strategy may reduce this



(a)



(b)

Figure 7: Comparison of average training and scoring time of the five experiments (unit: hour). (a) training time; (b) test time. The proposed approach provides the shortest training and test time in all cases.

gap. In the rest cases, the proposed MNNs have similar ROC curves and comparable AUCs, compared with other baseline methods. Another observation is that compared to the RNN-based prediction NN (Baseline 2), the DNN-based solution (Proposed) reports better AUC in all cases.

The proposed MNN structure has the shortest training and scoring time, which is indicated in Fig. 7. Compared to Baseline 1, the scoring time of the proposed method is reduced by a factor of 6. Efficient scoring is the second main contribution of MNNs, other than automatic feature engineering.

5 CONCLUSIONS

Industry commonly uses raw event logs in their applications, which usually consist of an ever-growing list of records that include an event ID, a time-stamp and a large number of other attributes. Most of these other attributes are sparse categorical data, which have very high cardinalities. The traditional method for modeling raw event logs is to use manual feature engineering to transform the raw logs into features that have lower dimensions. However, manual feature engineering depends significantly on human input, which makes it a bottleneck because both the volume and the dimensionality of data is growing exponentially in industry. Recently, DNN-based feature interaction exploration solutions are proposed to avoid manually build up high-order cross features. However, However, it is not practical to employ them for mining event logs because all of the event logs must be preprocessed for feature extractio during every scoring process.

In this paper, we present our recent study on CTR prediction using high-dimensional event logs from Yahoo!. We propose a novel approach, which is called a Midway Neural Network (MNN), performs automatic feature engineering—including the identification of time-series features from growing raw event logs with high dimensional categorical and numerical attributes—with practical costs in terms of memory/storage/computation usage. An MNN consists of several encoding DNNs, one prediction NN, and one Midway layer. Each encoding DNN is used to encode the raw features in a narrow time window in millions of dimensions into a much shorter embedding vector, called the Midway vector. The Midway layer is used to remember the sequence of Midway vectors from each of the narrow time windows in the event logs and then to concatenate them into a longer vector. The prediction NN is then used to model the concatenated Midway vector without any manual feature engineering operations. Additionally, an MNN also supports incremental training and scoring. The memory/storage/computation costs are limited, namely, proportional to the dimension of midway layer, which is usually one or two magnitudes less than the time-aligned raw features across all event logs. Our experiments on user online behavior modeling show that the proposed MNNs have comparable prediction accuracy with much lower computational costs (in terms of memory, storage, and CPU time) compared with modeling a set of complicated feature engineering operations.

ACKNOWLEDGEMENT

The authors kindly thank Haiying Huang, Gil Barretto, Jack Wang, Jingchao Sun from the Yahoo! ads targeting team for the data preprocessing, and Heting Yan from Florida State University for helpful discussions.

REFERENCES

[1] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. 2016. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*. 173–182.
[2] RP Jagadeesh Chandra Bose and Wil MP van der Aalst. 2013. Discovering signature patterns from event logs. In *Computational Intelligence and Data Mining (CIDM), 2013 IEEE Symposium on*. IEEE, 111–118.
[3] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 785–794.
[4] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 7–10.
[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase

representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[6] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[7] Tak-chung Fu. 2011. A review on time series data mining. *Engineering Applications of Artificial Intelligence* 24, 1 (2011), 164–181.

[8] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[9] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247* (2017).

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[11] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine* 29, 6 (2012), 82–97.

[12] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[13] T Jayalakshmi and A Santhakumaran. 2011. Statistical normalization and back propagation for classification. *International Journal of Computer Theory and Engineering* 3, 1 (2011), 1793–8201.

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[16] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. *arXiv preprint arXiv:1803.05170* (2018).

[17] T Warren Liao. 2005. Clustering of time series dataâĂŤa survey. *Pattern recognition* 38, 11 (2005), 1857–1874.

[18] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

[19] Alex Nanopoulos, Rob Alcock, and Yannis Manolopoulos. 2001. Feature-based classification of time-series data. *International Journal of Computer Research* 10, 3 (2001), 49–61.

[20] Yanru Qu, Bohui Fang, Weinan Zhang, Ruiming Tang, Minzhe Niu, Huifeng Guo, Yong Yu, and Xiuqiang He. 2018. Product-Based Neural Networks for User Response Prediction over Multi-Field Categorical Data. *ACM Transactions on Information Systems (TOIS)* 37, 1 (2018), 5.

[21] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 521–530.

[22] Haşim Sak, Andrew Senior, and Françoise Beaufays. 2014. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*.

[23] Lars Ropeid Selsaas, Bikash Agrawal, Chumming Rong, and Thomasz Wiktorski. 2015. AFFM: Auto feature engineering in field-aware factorization machines for predictive analytics. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*. IEEE, 1705–1709.

[24] Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. 2016. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 255–262.

[25] SPARK. 2019. Overview - Spark Documentation. (2019). https://spark.apache.org/docs/latest/

[26] Vladimir Svetnik, Andy Liaw, Christopher Tong, J Christopher Culberson, Robert P Sheridan, and Bradley P Feuston. 2003. Random forest: a classification and regression tool for compound classification and QSAR modeling. *Journal of chemical information and computer sciences* 43, 6 (2003), 1947–1958.

[27] Risto Vaarandi. 2008. Mining event logs with slct and loghound. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*. IEEE, 1071–1074.

[28] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. ACM, 12.

[29] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep learning over multi-field categorical data. In *European conference on information retrieval*. Springer, 45–57.