A Data-driven Approach for Performance Evaluation for Cache Server Groups in Content Delivery Network

Ziyan Wu, Zhihui Lu*, Wei Zhang, Jie Wu

School of Computer Science, Fudan University, Shanghai 200433, China

Abstract

In industry, Content Delivery Network(CDN) Service providers are increasingly using data-driven mechanisms to build their performance model of their service-providing systems. To build a model to accurately describe the performance of the existing infrastructure is very crucial to make resource management decisions. Conventional approaches that use hand-tuned parameters or linear models have their drawback. Recently, data-driven paradigm have been shown to greatly outperform traditional methods in modeling complex systems. We design a approach that using these techniques to build a performance model for CDN cache server groups. We use deep LSTM auto-encoder to capture the temporal structures from the high-dimensional monitoring data, and use a deep neural network to predict the reach rate, which is a client QoS measurement from the CDN service providers perspective. The experimental results have shown that our model is able to outperform state-of-the-arts models.

Keywords: *edge computing, deep learning, content delivery networks, sequence learning, predictive analysis, high dimensional data, prognosis*

1. Introduction

There is a trend [1] [2] [3] [4] [5] that using data-driven methods to model complex networked systems. Traditional approaches typically use some simple heuristics. These methods have several drawbacks. They cannot accurately reflect the complex systems due to the lack of knowledge of real-word environment. Driven by the opportunity to collect and analyze data (e.g., application quality measurement from end users), many recent proposals have demonstrated the promise of using deep learning to characterize and optimize networked systems. Drawing parallel from the success of deep-learning on pattern recognizaition, instead of using empirical non-linear learning model to describe the complex interaction of different features, we use deep learning methods and treat networked systems as a black-box.

A content delivery network (CDN) is a globally distributed networked system deployed across the Internet. Composed with geographically distributed cache servers, CDNs deliver cached content to customers worldwide based on their geographic locations. Extensively using cache servers, content delivery over CDN has low latency, reliability, supports better quality of experience and security.

The CDN Service providers are increasingly using data-driven mechanisms to build their performance model of their service-providing systems. To build a model to accurately provide an understanding of the performance of the existing infrastructure such as the health of cache groups and network status, is very crucial to make resource management decisions including content placement, network traffic scheduling, load balance of the CDN network. Modeling all available physical resources, we can maximize a resource utilization in terms of service quality, cost, profit, etc.

Generally CDN providers don't have direct measurement from the clients (the logs from video players, web browser that can show the QoE of clients). Instead, they use the indirect measurement reach ratewhich is collected and calculated from the log of the HA proxy of CDN cache groups. The computation of reach rateis done offline . In order to enable themselves make resource management decisions in real time, the CDN providers have to use the metrics that can be collected in the real time to infer the reach rate.

Cache server groups can be characterised as multi-dimensional, highly non-linear, time variant vectors. The metrics that collected from members of the CDN cache server groups are sequence data that are measured every minute, which have hundreds of dimensions. The state-of-art methods are typically using simple heuristics which are oversimplifed and biased due to the human experience, or linear models, which cannot characterize the complex relationship between multiple metrics. Deep learning is a branch of machine learning based on a set of algorithms that attempts to model high-level abstractions in data by using artificial neural network architectures composed of multiple non-linear transformations [6]. They have a lot of suc-

*Corresponding author

cessful applications in sequence modeling. Compared to other machine learning techniques, a lot of work show that it can detect complex relationships among features, can extract hierarchical level of features from high-dimensional data, including monitoring data.

We frame our problem as a sequence learning problem, which consists of stages: (1) feature engineering (2) representation learning by lstm auto-encoder to extract useful (3) a feed foward neural network black-box machine learning algorithm to output the predictions.

Our main contributions are listed below:

- We frame performance evaluation problem as a sequence learning problem.
- We use representation learning by lstm auto-encoder to extract useful features from data.
- We compare our methods with state-of-arts methods and show ours is superior by empirical studies.

The remain organization of this paper is as follows. In Section 2, we first introduce the related concept as our research background. In Section 3, we formulate our performance evaluation problem as a sequence learning problem and then our LSTM-based methods. In Section 4, we introduce our method of feature engineering to reduce the dimensionality for the high-dimensional data and our reach rateprediction model based on lstm auto-encoder. In Section 5, we show our experiment setting and demonstrate performance improvements of our methods over baseline models. Section 6 is discussion and future work. We provide concluding remarks in Section 7.

2. Background

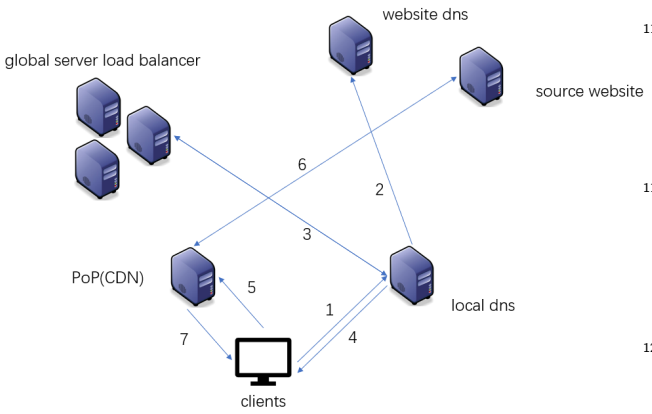

Figure 1: The basic working procedure of CDN

A content delivery network or content distribution network (CDN) (figure 1) is a geographically distributed network of cache servers. CDN helps content provider deliver

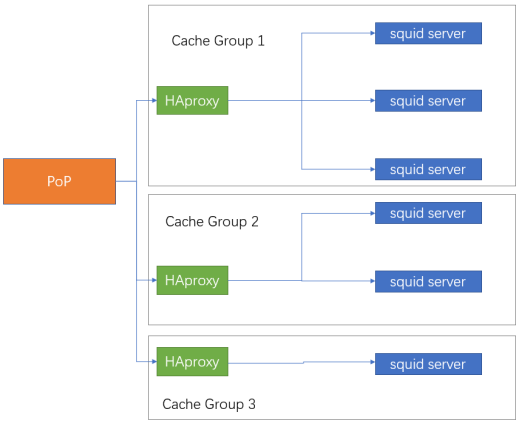

Figure 2: The figure

web pages and other multi-media content to the clients, based on the locations of the clients and cache servers nearby the clients. The basic working procedure is as follows. Step 1client send a request to local dns. Step 2: the DNS finds the CNAME and redirects the request to the gslb (global sever load balance). Step 3: the local DNS server sends request to the glsb and glsb returns the ip address of CDN servers based on the scheduling policy. Step 4: the local DNS return the ip address to the clients. Step 5: clients make a request to fetch content from the selected PoP. Step 6: the cache server groups will pull the content from source website if the content doesn't exsist locally. Step 7: content is sent to the clients.

A CDN cache server group is the basic resource scheduling unit for CDN. A CDN cache group is a load balanced cluster that consists of interconnected cache servers. A typical implementation consists of HAproxy and squid servers. The requests from clients are distributed by HAproxy to cache servers. HAproxy use be set to use different algorithms algorithms to maximize the utilization of every server. Round-robin algorithm distributes the load equally to each server in a homogenours cluster. In heterogeneous cluster, weighted round-robin algorithm is used. A weight was assigned to the server based on its processing capabilities. A heterogeneous structure of a cluster adds complexity to the feature engineering.

As CDN providers don't have direct QoS measurement from the clients (the logs from video players, web browser that can show the QoE), so they use the indirect measurement reach ratewhich is calculated from log of the HA proxy of CDN cache groups with a delay about three minutes.

The number of metrics we collected from the cache servers of cache group are different due to the different configuration, from 64 to 134. The features we have constructed are listed in table. There are about 312 features. They include cpu utilization, network utilization, disk utilization, memory utilization and so on.

feature list
group 1
cpu1.usage
cpu2.usage
...
mem_cached
mem_buffers_cache_free
memory.swap
disk.used.sda1
disk.used.sda2
...
channeltraffic_in
channeltraffic_in
ioutil_util_sda
ioutil_util_sdb
ioutil_util_sdc
ioutil_util_sdd
...
iowait.wait
hitratio.port8101
hitratio.port8102
...
group 2
aggregate.cpu
aggregate.diskused
aggregated.ioutil
aggregate.CPU.max
....
group 3
aggregate.all_machine.cpu
aggregate.all_machine.diskused
aggregated.all_machine.ioutil
aggregate.all_machine.CPU.max

Table 1: list of candidate input features from one cache server. We organize the features into groups. The first group is the raw data we directly collected from the caching servers. The second group are the features we construct based on statistics of the metrics of a single cache server. The third group are the features we construct based on the statistics of the metrics of the whole cache server groups

3. Problem Formulation and Models

We argue that performance evaluation as a sequence learning problem. Since we are able to collect the caching servers performance metrics and network metrics at a certain time interval, we can use a sequence models to describe relationship between metrics collected from cache groups and reach rate

There are three categories of sequence learning problem, which are many to many, one to many and many to one. Our goal is to model the relationship between a sequence collected metrics and reach ratewithin a certain period of time, which is many to one. In general, we can use the following formulation to describe the prediction process.

Giving a sequence of vectors, $\{x_n\} = \{x_\alpha \in R^d | \alpha \in N\}$, , where d is the number of the features, we use $\{x_n\}$ to represent the sequence and x_t to describe a data point at time t with d dimensions.

Given another target sequence, $\{y_n\} = \{y_\alpha \in R | \alpha \in N\}$, our goal is to find the relation between$\{x_n\}$ and $\{y_n\}$, which is

$$y_t = f(x_t, x_{t-1}, ..., x_{t-n+1})$$

where n is the window size and f is the mapping we want to learn from the data.

Many models can be used to approximate f in sequence modeling. The most naive way is to use simple heuristics, which is use exponential moving average to linearly map each metrics to a score in a certain time interval. The paremeters are depending on the experience of the operators. This methods are impractical: the model can hardly generalize well and requires tedious repetitive tuning.

3.1. linear models for sequence learning

There are some conventional approaches which use data to learn: use the multiple linear regression model(MLR). The MLR method builds a model of a sequence that is composed of a linear part and a random noise part. The linear part models the linear relationship between the dependent variables and independent variables, the random noise reflects the unpredictable randomness. Formally, the model for multiple linear regression, given n observations, is

$$y_t = c + \sum_{i=t-n+1}^{t} (\beta_i x_{t_i}) + \epsilon_t$$

Furthermore, the linear part incorporates historical values of the sequence. Here, y denotes the target we want to model, c denotes the constant parameter of the linear decomposition, denotes the parameters to be computed and epsilon reflects the random noise part. The best-fitting line for the observed data is calculated by the least square method.

Linear models are easy to implement and have good interpretation and thus are widely used in many real work. However, linear models are shown not sufficient to describe

some nonlinear behaviors of the complex network systems. In many cases, neural networks tend to outperform linear models [A comparison of artificial neural network and time series models for forecasting commodity prices]. In our experiments, we observe a non-linear relationship between the metrics and reach rate.

3.2. non-linear for sequence learning

Sequence prediction often involves forecasting the next value in a real valued sequence or outputting a class label for an input sequence.

Deep learning (DL) is a branch of machine learning based on a set of algorithms that attempts to model high-level abstractions in data by using artificial neural network (ANN) architectures composed of multiple non-linear transformations. They have a lot of successful applications in sequence modeling. Compared to other machine learning techniques, it can detect complex relationships among features, can extract hierarchical level of features from raw data. So it can produce more accurate result and build a model more accurate with less time. They are advantageous for modeling intricate systems because neural networks do not require the user to predefine the feature interactions in the model, which assumes relationships within the data. Instead, the neural network searches for patterns and interactions between features to automatically generate a best fit model.

One of the more popular kind of deep neural networks is the Recurrent Neural Network (RNN). RNNs are a class of neural networks that depend on the sequential nature of their input. Such inputs could be text, speech, time series, and anything else in which the occurrence of an element in the sequence is dependent on the elements that appeared before it.

3.2.1. Basic feed forward neural network

A generic three layered neural network is illustrated in Figure 2. In this study, the input matrix $I \in R^{Mx(NxT+1)}$ where M is the number of training examples and $NxT+1$ is the number of features (metrics that collected from the cache servers) concatenating the bias term. The input matrix is then multiplied by the model parameters matrix W_1 to produce the hidden state matrix h. The Output of the first layer is transformed by an activation function. We can add more layers to the network. interacts with the second parameters matrix Θ^2 to calculate the output O. The size and number of hidden layers can be varied to model systems of varying complexity. The whole process can be imagined as the information propagating forward. The forward propagation is formalized as follows:

$$\begin{aligned} \mathbf{h}_1 &= \text{sigmoid}(\mathbf{W}_1 * \mathbf{I}) \\ \mathbf{h}_t &= \text{sigmoid}(\mathbf{W}_{t-1} * \mathbf{h}_{t-1}) \\ \mathbf{O} &= \text{sigmoid}(\mathbf{W} * \mathbf{h}_{lastlayer}) \end{aligned} \quad (1)$$

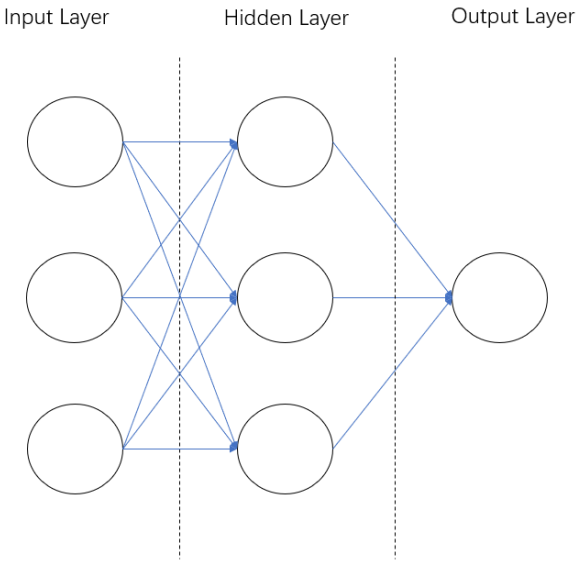

Figure 3: MLP structure

where sigmoid is the activation function:

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}} \quad (2)$$

the loss function J use mean square error: J

The process of training any neural network model can be broken down into four steps: (1) Randomly initialize the model parameters, (2) forward propagation algorithm,(3) Compute the cost function J, (4) back propagate using the chain rule (5) using the gradients calculated, adjust weights to minimize the cost function J.

One major assumption for NNs is the independence of samples. For sequence learning problem, however, this assumption doesn't hold true, for the samples of our problem have a temporal relationship: the status of system of the next timestep not only depends on the status in the current timestep but also on the previous timesteps of indefinite length. One solution is to using a sliding window to capture the sequential relationship between the samples. The performance of this method depends on the window size, which isn't pratical for the dependencies length which isn't a fixed value. RNN eliminates the need to find the size of the window[7].

3.2.2. RNN

The promise of recurrent neural networks is that the temporal dependence and contextual information in the input data can be learned[8] [9].

RNNs 3 process the input sequence one element at a time and maintain a hidden state vector which acts as a memory for past information. They learn to selectively retain relevant information allowing them to capture dependencies across several time steps. This allows them to

utilize both current input and past information while making future predictions. All this is learned by the model automatically without much knowledge of the cycles or time dependencies in data. RNNs obviate the need for a fixed size time window and can also handle variable length sequences. Moreover, the number of states that can be represented by an NN is exponential in the number of nodes.

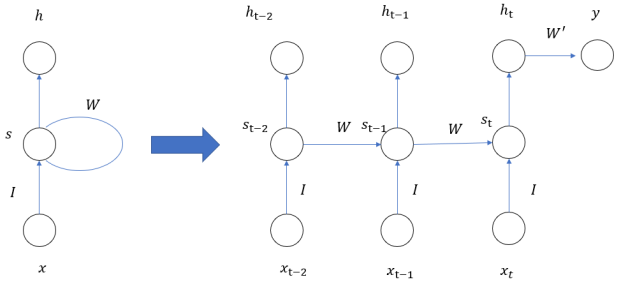

Figure 4: The left shows the backloop structure of RNN and right shows that RNN can be thought as infinite deep layers neural network unfolded along the dimension of time

RNNs maintain a hidden vector $\mathbf{h}$, which is updated at time step t as follows:

$$\mathbf{h}_t = \tanh(\mathbf{W} * \mathbf{h}_{t-1} + \mathbf{I} *_t) \quad (3)$$

where tanh is the hyperbolic tangent function, $\mathbf{W}$ is the recurrent weight matrix and U is a input wight matrix. The hidden state $\mathbf{h}$ is then used to make a prediction

$$\mathbf{y}_t = f(\mathbf{W} * \mathbf{h}_{t-1}) \quad (4)$$

where f can be fully connected layer that linearly maps the hidden state to an output. By using $\mathbf{h}$ as the input to another RNN, we can stack RNNs, creating deeper architectures

$$\mathbf{h}_t^l = \sigma(\mathbf{W} * \mathbf{h}_{t-1}^l + \mathbf{I} * \mathbf{h}_t^{l-1}). \quad (5)$$

The training of RNN use back-propagation through time (BPTT). RNN have several drawbacks: 1. Training vanilla RNNs is known to be particularly difficult, with vanishing and exploding gradients being one possible explanation [8]. 2. RNN aren't capable of learning long-term dependencies. LSTM address these issues by introducing LSTM cell[10]

4. Methods

4.1. Feature Engineering

The feature engineering is the process after data-cleansing, in which we fill the missing data and reformat it. The propose of this stage is two-fold: (1) to find a unified equal-length vector representation of all of the cache groups. The metrics collected are in the granularity of caching servers which have different dimensionality. As showed in graph. To make things more complex, a cache group may have different number caching servers. As we only care about. (2) reduce the dimensionality, for it is very time-consuming to train the models when the number of dimensionality is too high.

Our feature engineering has three steps: 1. data prepossessing which converts raw data into high-dimensional vectors. 2. feature correlation that characterize the linear relationship between every pair of features. 3. cluster analysis that selects the set of features.

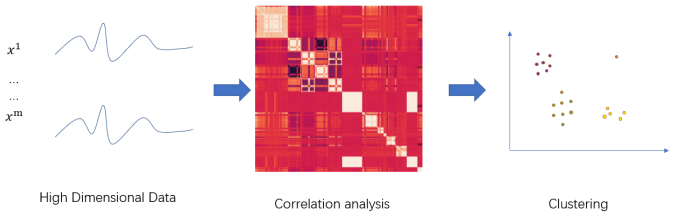

Figure 5: three steps of feature enginnering

As there are hundreds of features, there are many overlaps among the variables. We use correlation in statistics to group highly correlated variables together and create composite features that can represent each group of variables.

Correlation is an analysis of two or more observed or random variables to determine a dependence between the variables. This dependence can be classified as the probability that changes in one variable affect the behaviour of the second variable. The Pearsons correlation defines this dependence in the interval [1.0, 1.0]. Pearsons correlation for two given random variables X and Y is computed by dividing the covariance of both variables with the product of their standard deviations.

Generally, cases of high correlation compute to a value close to 1.0, high anticorrelation is associated with a value close to -1.0 and no correlation is assumed. if the value is around 0.0, the variables appear to be independent.

The correlation matrix we calculate is pictured in figure 6. The white color denotes the correlation of 1.0 and the black denotes -1.0. The red denotes that there is no linear relationship at all. As we can see from the picture that there are overlaps among the features.

After we calculate the correlation matrix for all the features, the correlation of thses features are regarded as distance. Then, we use DBSCAN to cluster these features to eliminate close ones. By defining the neighorhood parameter ϵ, we specify how much extent we regard two features as close. If a feature is highly correlated with another one, we should add these features into the same cluster. When the clustering is done, we choose the representative feature

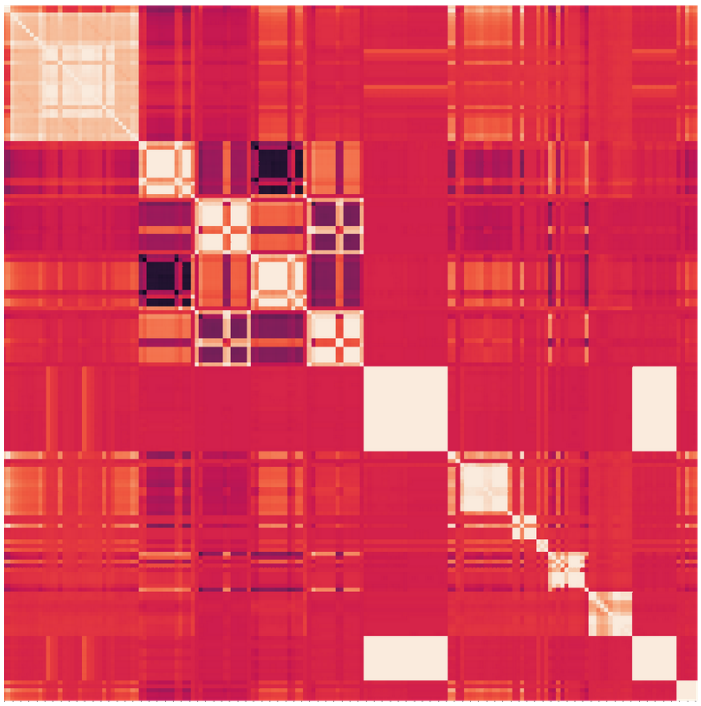

Figure 6: Correlation Matrix of the feature set

in the cluster. We formulate the process in Algorithm 1.

4.2. Prediction Model Design

In this section, we introduce how the architecture we use to predict reach rateof CDN cache group using the data output from the feature engineering stage. The key components of our model are LSTM, LSTM auto-encoder and deep feed foward neurla network.

4.2.1. LSTM

LSTM, introduced in [10], addresses the problem of vanishing gradients by introducing a memory cell. [11] applied in time series. The inner working of LSTM are listed follows:

$$
\begin{aligned}
\mathbf{g}_u &= \sigma(\mathbf{W}_u * \mathbf{h}_{t-1} + \mathbf{I}_u * t) \\
\mathbf{g}_f &= \sigma(\mathbf{W}_f * \mathbf{h}_{t-1} + \mathbf{I}_f * t) \\
\mathbf{g}_o &= \sigma(\mathbf{W}_o * \mathbf{h}_{t-1} + \mathbf{I}_o * t) \\
\mathbf{g}_c &= \tanh(\mathbf{W}_c * \mathbf{h}_{t-1} + \mathbf{I}_c * t) \\
\mathbf{m}_t &= \mathbf{g}_f \odot + \mathbf{g}^u \odot \mathbf{g}^c \\
\mathbf{h}_t &= \tanh(\mathbf{g}_o \odot \mathbf{m}_{t-1})
\end{aligned}
\tag{6}
$$

here σ is the logistic sigmoid function, $\mathbf{W}_u, \mathbf{W}_f, \mathbf{W}_o, \mathbf{W}_c$ are recurrent weight matrices and $\mathbf{I}_u, \mathbf{I}_f, \mathbf{I}_o, \mathbf{I}_c$ are projection matrices.

4.2.2. LSTM auto-encoder

An auto-encoder contains: an encoding function, a decoding function, and a distance function between the amount of information loss between the compressed representation of your data and the decompressed representation (i.e. a "loss" function). The encoder and decoder will be chosen to be deep layered lstm network. So the parameters of the encoding/decoding functions can be optimize to minimize the reconstruction loss, using Stochastic Gradient Descent.

Prior to fitting the data prediction model, we first conduct a pre-training step to fit an encoder that can extract useful and representative embeddings froma time series. The goals are to ensure that (i) the learned embedding provides useful features for prediction and (ii) unusual input can be captured in the embedded space, which will get further propagated to the prediction network in the next step.

As illustrated in figure 8. Given a multi-variate time series of data, the encoder reads the vectors as input and transform them as latent variables. During the pretraining, only the the weights of LSTM auto-encoder are trained. The LSTM auto-encoder is trained to reconstruct the input on the output side. In the second step, a fulley connected feed foward network take the latent variables which are the ouputs of the encoder as input. The targets are the reach rate.

Algorithm 1 Feature Aggregation and Selection

Input: All features list: F; Neighborhood parameter ϵ;
Output: Selected features list F_S;

```
Initialize the set of core object: Ω=∅;
for i in F do
    Confirm the ϵ-neighborhood  N(i,ϵ);
    if |N(i,ϵ)|>1 then
        Add i to the core object set: Ω;
    else
        Add i to the selected features list F_S;
Initialize the number of clusters: k=0;
Initialize the set of features: Γ= Ω;
while Ω ≠ ∅ do
    Check the set of unvisited features: ψ= Γ;
    Random select an core object o ∈ Ω,initilaize the queue Q=⟨o⟩;
    Γ=Γ\ {o};
    while Q ≠ ∅ do
        Fetch an element from Q: q;
        Δ = N(q,ϵ) ⋂ Γ;
        Add the elements in Δ to Q;
        Γ = Γ\Δ;
    k = k + 1;
    Generate the cluster C_k=ψ\Γ;
    Ω = Ω\C_k;
for i=1,2...k do
    Fetch an element in C_k out to the selected features list F_s
return F_s
```

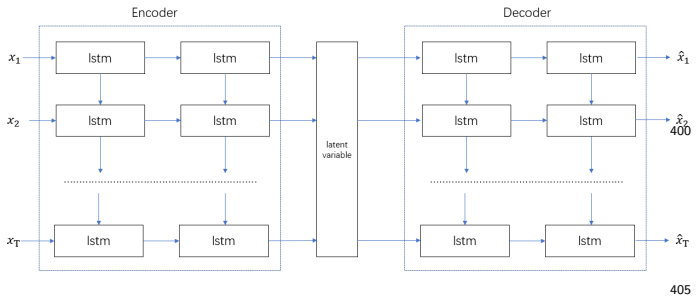

Figure 7: LSTM Auto-encoder Model

5. Evaluation

5.1. Experimental Settings and Dataset

Our implementation uses the Google opensource deep learning library, Tensorflow[12], version 1.2.0. We ran our experiments on a physical machine running an Ubuntu 16.04 operation system, intel i7-6700HQ, 16 GB memory, and GPU gtx1060.

To test the performance, we select two cache groups with average request above 7000 per minutes. The first cache servers groups have 13 cache servers while the other have 10 cache servers. The metrics are raw data collected from the cache servers, including cpu utilization, network utilization, disk utilization, memory utilization of two cache groups servers. The frequency of the metrics and reach rateare both minute-by-minute. This data covers the period from December 20, 2017 to January 26, 2018.

In the experiments, 5-fold cross validation is used to evaluate the accuracy of predictions. This divides the entire data set into 5 equal parts. Prediction is repeated510 times, each time keeping one of the 5 parts as test data and the other 4 parts as training data to build the prediction model. Finally, test results on all the 5 parts are accumulated together to calculate the average prediction accuracy.

In the pre-training procedure We use minibatch stochcasti gradient descent (SGD) together with the Adam optimizer to train the LSTM auto-encoder model. The size of the minibatch is 128. The weights can be learned by standard lstm learning algorithm propagation throught with mean squared error as the objective function. In the training procesure we use a 4 layer feed-foward neural network. We use the batch gradient descent training algorithm to train the neural network.

5.2. Baseline

We compare our model with other baseline model which are listed follow:

1. multiple linear regression model
2. feed-foward neural network
3. vanilla LSTM model
4. LSTM encoder-decoder with a feed-foward neural network

5.3. Experimental Results

We use three different evaluation metrics to compare our models and methods. Among them, root mean squared error (RMSE) and mean absolute error (MAE) are two scale-dependent measures, and mean absolute percentage error (MAPE) is a scale-dependent measure. Specifically, assuming $y^{'}$ is the target at time t and y is the predicted value at time t.

$$RMSE(y^{'}, y) = \sqrt{\frac{1}{n} \times \sum_{n=1}^{N} (y^{'}{}_t - y_t)^2} \tag{7}$$

$$MAE(y^{'}, y) = \frac{1}{N} \times \left| y^{'}{}_t - y_t \right| \tag{8}$$

$$MAPE(y^{'}, y) = \frac{1}{N} \times \left| \frac{y^{'}{}_t - y_t}{y^{'}_t} \right| \tag{9}$$

The prediction results on test set are in figure 10. The performance comparision is listed in Table 2. From the experiment, we observe that that methods using neural networks are superior to the one that using linear model, for linear model cannot characterize the non-linear relationship between the features and targets. The recurrent network based methods perform better than the feed foward neural network, for it contains the hidden state. Our method performs the best, for LSTM auto-encoder can extract temporal structure from the high-dimensional data into a lower dimensional dense representation. For LSTM auto-encoder, we try four different configuration regarding the length of variable, which are 5, 10 , 15 and 20. We observe that although with longer latent variables LSTM auto-encoder performs better when reconstructing output as illustrated in figure , it performs worse when used to predict the reach rate. We believe that it is because when the latent variable size is too large, it isn't forced to learn the representation that is most related to reach rate.

6. Related Work

In recent year, deep learning has gained success in different fields, including image recognization, speech recognization, natural language understanding[6]. RNN[13] outperform the traditional feed-forward neural network in modeling sequence data because its structure can characterize the temporal structure of input data by introducing hidden state. Training RNN is hard because of vanishing gradients problem; LSTM address these issues by introducing LSTM cell[10]. LSTM has been widely proven successful in sequence modeling. In [14], a attention-based lstm outperforms traditional methods like ARIMA. In [15],

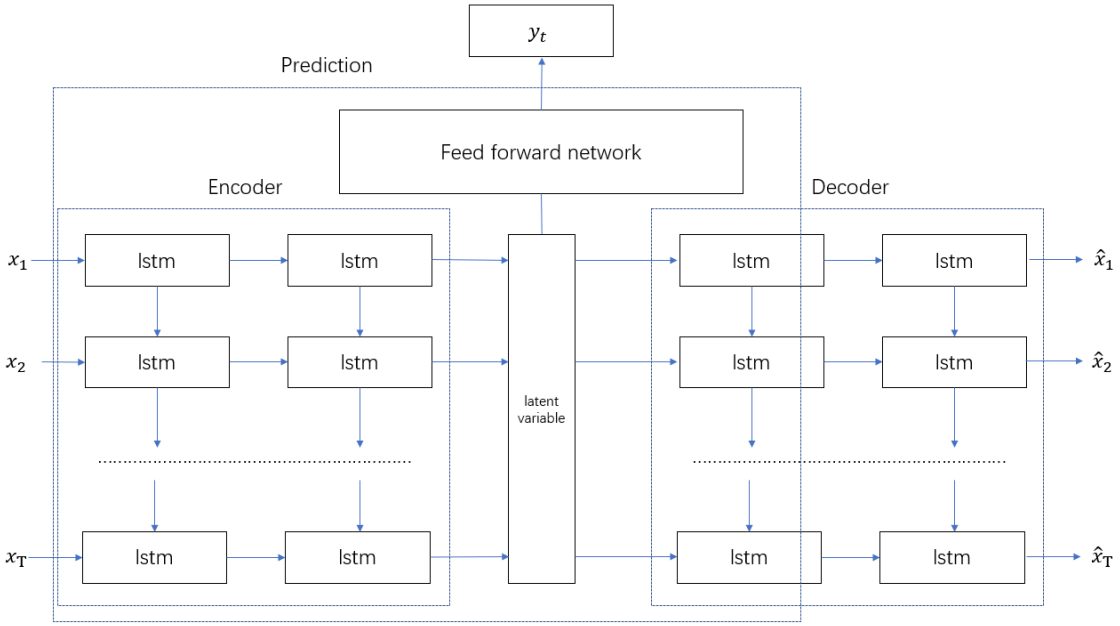

Figure 8: lstm auto-encoder with a deep feadfoward network

Table 2: Performance Comparison

Models	MAE	MAPE	RMSE
multiple linear regression model	0.598	0.604	0.573
feed-foward neural network with sliding windows	0.387	0.481	0.427
vanilla LSTM model	0.188	0.236	0.164
LSTM auto-encoder with feed-foward neural network (latent size = 20)	0.142	0.211	0.133
LSTM auto-encoder with feed-foward neural network (latent size = 15)	0.178	0.146	0.123
LSTM auto-encoder with feed-foward neural network (latent size = 10)	0.138	0.124	0.097
LSTM auto-encoder with feed-foward neural network (latent size = 5)	0.168	0.141	0.127

a robust model was used to predict number of trips and do anomaly detection for Uber.

When evaluating the complex system, the evaluation method can fall into two catagory: model-driven method, data-driven method. In model-driven method, the mathematical model characterizing the inner components of a system has to be explained explicitly. In [16], a model-driven method is used to solve optimal cache-deployment problem. However, in data-driven method, the data characterizing the behaviour of system instead of the analytical model is necessary. In [1], CDN selection decisions of QoE optimization decision is based on the past experience of decisions and decisions outcome. In [2], a ABR decision framework using deep reinforcement learning treats network as a black box. In [17], a convolutional bi-directional LSTM network is used to predict the machine health by sensor data. in [18], the VM selection mechanism allows users to balance performance gains with cost reductions. From the characteristics of the above methods, the data driven method, which takes the gathered data as basis and is independent of the objects prior knowledge, is a more useful approach for performance evaluation, fault detection and reliability evaluation.

There are extensive research regarding CDN, for a large portion of internet traffic was boosted by service provided by CDN. There are three category: (1) long-term network planning, including optimized CDN design that relates to PoP selection and cache deployment[19][20][16], and (2) short-term, run-time cache management, including content replacement and prefetching strategies in the CDN network[21][22][23]. (3) CDN selection to optimize QoE for the client[1][24]. Our research falls in the second categories.

7. Conclusion and Future Work

In this paper we present a data-driven approach to evaluate the performance of cache server groups. The lstm auto-encoder can capture the long-term temporal information in the sequences. This paper shows that it is feasible to apply state-of-the-art deep learning techniques to model networked systems that provides estimation for its performance. The empirical studies shows that ours has outperformed the conventional methods.

As there are hundreds cache group in China. Although our method can be used to train on all kinds of cache

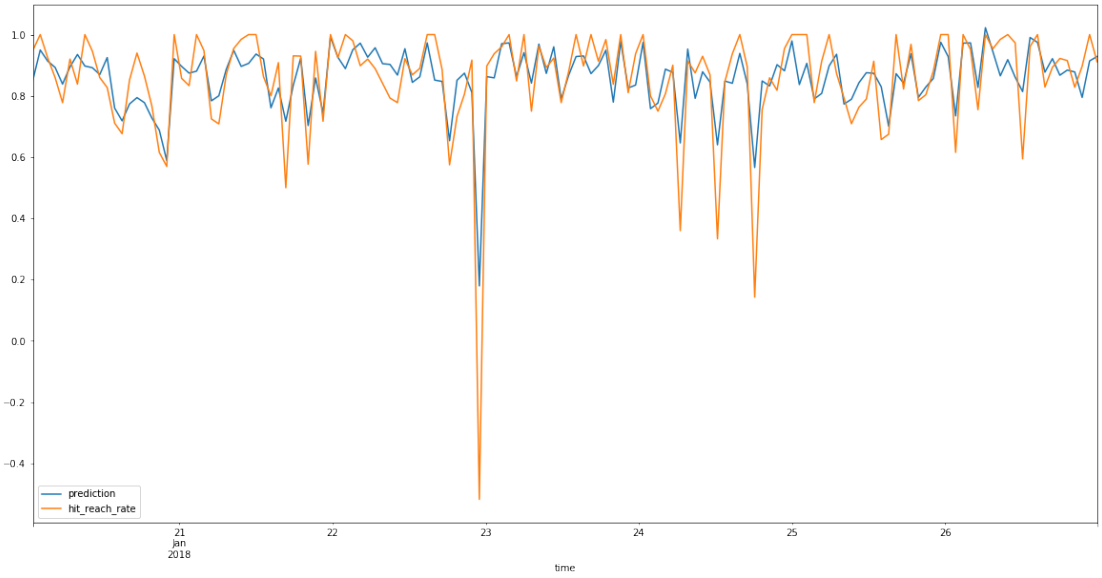

Figure 9: Prediction Results of LSTM auto-encoder

group of different architecture, the model for one specific cache group, a unified models for all kinds of cache groups is required. As qualified rate is an indirect measurement of QoS of clients, collecting data from clients ends will provide useful insights. We also want to develop the online training methods for our models because we observe that the relationship changes over time.

8. Acknowledgment

The work of this paper is supported by National Natural Science Foundation of China under Grant No.61728202-Research on Internet of Things Big Data Transmission and Processing Architecture based on Cloud-Fog Hybrid Computing Model Grant No. 61572137-Multiple Clouds based CDN as a Service Key Technology Research, and Shanghai 2016 Innovation Action Project under Grant 16DZ1100200-Data-trade-supporting Big Data Testbed.

References

[1] J. Jiang, S. Sun, V. Sekar, H. Zhang, Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation, 14th USENAIX Symposium on NSDI.

[2] H. Mao, R. Netravali, M. Alizadeh, Neural Adaptive Video Streaming with Pensieve, Mohammad Alizadeh MIT Computer Science and Artificial Intelligence Laboratory (2017) 197–210doi:10.1145/3098822.3098843.
URL http://dx.doi.org/10.1145/3098822.3098843

[3] T. A. O. Li, Data-Driven Techniques in Computing System Management 50 (3).

[4] J. Jiang, R. Das, G. Ananthanarayanan, P. A. Chou, V. N. Padmanabhan, V. Sekar, E. Dominique, M. Goliszewski, D. Kukoleca, R. Vafin, H. Zhang, VIA: Improving Internet Telephony Call Quality Using Predictive Relay Selectiondoi:10.1145/2934872.2934907.
URL https://www.cs.cmu.edu/~junchenj/via.pdf

[5] J. Gao, Machine Learning Applications for Data Center Optimization.
URL https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42542.pdf

[6] Y. Lecun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444. arXiv:arXiv:1312.6184v5, doi:10.1038/nature14539.

[7] M. Hermans, B. Schrauwen, Training and Analyzing Deep Recurrent Neural Networks 1–9.

[8] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Transactions on Neural Networks 5 (2) (1994) 157–166. doi:10.1109/72.279181.
URL http://www.ncbi.nlm.nih.gov/pubmed/18267787http://ieeexplore.ieee.org/document/279181/

[9] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning Phrase Representations using RNN EncoderDecoder for Statistical Machine Translation.
URL http://www.statnlp.org/wp-content/uploads/2016/02/rnn.pdf

[10] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory, Neural Computation 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[11] P. Malhotra, L. Vig, G. Shroff, P. Agarwal, Long Short Term Memory Networks for Anomaly Detection in Time Series.
URL https://www.researchgate.net/profile/Pankaj_Malhotra3/publication/304782562_Long_Short_Term_Memory_Networks_for_Anomaly_Detection_in_Time_Series/links/5880506308ae71eb5dbfbd10/Long-Short-Term-Memory-Networks-for-Anomaly-Detection-in-Time-S.pdf

[12] TensorFlow.
URL https://www.tensorflow.org/

[13] J. Schmidhuber, A local learning algorithm for dynamic feedforward and recurrent networks, Connection Science 1 (4) (1989) 403–412. doi:10.1080/09540098908915650.
URL http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=B1927E88DDAA6F03912D61DD23B7DABE?doi=10.1.1.51.8813{&}rep=rep1{&}type=pdf{%}0Ahttp://www.tandfonline.com/doi/abs/10.1080/09540098908915650{%}5Cnhttps://www.tandfonline.com/doi/full/10.1080/09540098908915650{%}0Ah

[14] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, G. W. Cottrell, A Dual-Stage Attention-Based Recurrent Neural Network for

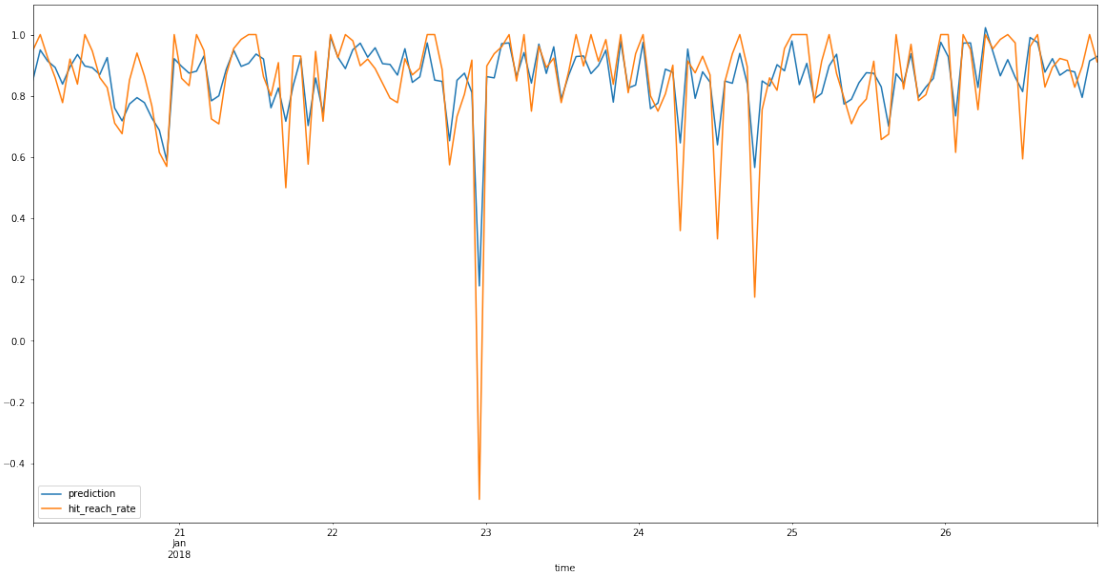

Figure 10: Prediction Results of LSTM auto-encoder with feed-forward neural network (downsampled

Time Series Prediction.
URL http://cseweb.ucsd.edu/{~}yaq007/DA-RNN.pdf

[15] L. Zhu, N. Laptev, Deep and Confident Prediction for Time Series at Uberdoi:10.1109/ICDMW.2017.19.

[16] G. Tang, K. Wu, R. Brunner, Rethinking CDN design with distributee time-varying traffic demands, in: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, IEEE, 2017, pp. 1–9. doi:10.1109/INFOCOM.2017.8057028.
URL http://ieeexplore.ieee.org/document/8057028/

[17] R. Zhao, R. Yan, J. Wang, K. Mao, Learning to monitor machine health with convolutional Bi-directional LSTM networks, Sensors (Switzerland) 17 (2) (2017) 1–18. doi:10.3390/s17020273.

[18] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, R. H. Katz, Selecting the ¡i¿best¡/i¿ VM across multiple public clouds, in: Proceedings of the 2017 Symposium on Cloud Computing - SoCC '17, ACM Press, New York, New York, USA, 2017, pp. 452–465. doi:10.1145/3127479.3131614.
URL http://dl.acm.org/citation.cfm?doid=3127479.3131614

[19] P. Krishnan, D. Raz, Y. Shavitt, The cache location problem, IEEE/ACM Transactions on Networking 8 (5) (2000) 568–582. doi:10.1109/90.879344.

[20] S. Hasan, S. Gorinsky, C. Dovrolis, R. K. Sitaraman, Trade-offs in optimizing the cache deployments of CDNs, Proceedings - IEEE INFOCOM (2014) 460–468doi:10.1109/INFOCOM.2014.6847969.

[21] S. Borst, V. Gupta, A. Walid, Distributed caching algorithms for content distribution networks, Proceedings - IEEE INFOCOMdoi:10.1109/INFCOM.2010.5461964.

[22] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, S. Chouvardas, Placing dynamic content in caches with small population, Proceedings - IEEE INFOCOM 2016-July. arXiv:1601.03926, doi:10.1109/INFOCOM.2016.7524380.

[23] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, K. K. Ramakrishnan, Optimal Content Placement for a Large-Scale VoD System, IEEE/ACM Transactions on Networking 24 (4) (2016) 2114–2127. doi:10.1109/TNET.2015.2461599.

[24] J. Jiang, V. Sekar, H. Milner, D. Shepherd, I. Stoica, H. Zhang, CFA: A Practical Prediction System for Video QoE Optimization.
URL https://www.cs.cmu.edu/~junchenj/cfa.pdf