Click Count Prediction Utilizing Link Graph Embedding and Semantic Correlation

Dawen ZHOU†, Qian DONG† and Mizuho IWAIHARA‡

Graduate School of Information, Production and Systems, Waseda University
2-7 Hibikino, Wakamatsu-ku, Kitakyushu-shi, Fukuoka, 808-0135 Japan

E-mail: † davinzhou620@fuji.waseda.jp, † dongqian43@outlook.com, ‡ iwaihara@waseda.jp

Abstract Click count prediction is one of important temporal prediction tasks, predicting the future popularity of web pages. In the traditional methods, a limited number of time series are used for prediction of the click counts of a target page. However, in reality, click traffics are influencing each other via inter-page links and similar trending contents. Therefore, the prediction model needs to focus on important and related time series out of a large number of time series, to perform prediction. We firstly observe that web page traffics are influenced by web pages of their link neighbors. With link-graph embedding, we can discover traffic-related pages, which helps improving prediction performance and dealing with missing data. Additionally, semantic similarity of two pages can affect their time-series similarity, which motivates us to utilize pre-trained language models, such as BERT and its variants, for incorporating semantic similarity of web pages. We propose a method utilizing sequence-to-sequence recurrent neural networks trained over combination of graph embeddings and sentence embeddings. Our experiments are conducted on the dataset of Web Traffic Time Series Forecasting, consisting of real click counts on Wikipedia articles.

Keyword Click count prediction, Graph analytics, BERT, Sequence-to-sequence recurrent neural networks

1. Introduction

A time series is a series of data points indexed in time order, which assists us to visually capture the trend of statistical objects. Time series prediction is used in a number of domains: Statistics, web traffic forecasting [26], complex system analysis [14], mathematical finance [23], weather forecasting [3].

Click count series is a form of time series in web traffic forecasting, which represents the number of user clicks generated on a web page over a period of time. Prediction on website traffic helps to tune infrastructure resources and tracing hot topics, which gives guidance on the next step of strategy development. Accurate prediction of traffic helps webmasters to tune these resources.

Real temporal data can stretch over a long period of time. Click Count Data is a typical type of temporal data. In current classical methodologies such as ARIMA [16] , RNN [6] and Seq2Seq model [4], their effectiveness is proved. However, they cannot capture causal relationship between click count series, since the weights are only updated by information from target series. Thus, we expect to take advantage of external features to enhance interpretability and achieve better prediction.

Websites contain pages and links, forming a graph. Users can jump to other pages by clicking on internal links in one page. Graph structure can provide rich information for forecast. Linked web pages form a graph structure, where nodes represent pages, and edges represent hyperlinks between pages. So, we proposed the first method: *Click-count Prediction with Graph Embedding* (CPGE). We firstly take advantage of the graph structure by utilizing graph embedding, which transforms a graph into low-dimensional vectors. To find a traffic-related series, we build a weighted graph where edges are weighted by similarity score between two series by DTW [2][7]. Finally, we use this information to effectively predict future click counts.

Moreover, for a webpage, clicks on a page often start with user searching behavior. In searching scenario, a search engine will return a number of web pages based on relevance to the topic. The relevance is chiefly based on semantic similarity between the search topic and the content of the page, which is often summarized in the page title and its leading text. When two pages are related to a phrase, the search engine will return both pages at the same time. So, when there is a high similarity between two pages content, the probability of them appearing at the same time also increases, and their clicks will be similarly affected. Thus, we expect that other pages with high text similarity to the target page can contribute us to infer future clicks on the target page and propose the second method: *Click-count Prediction with Semantic Embedding* (CPSE).

Evaluating semantic similarity of documents has been rapidly evolving in the NLP field. In CPSE method, we firstly apply BERT and design a finetuning strategy by using the series in downstream task. The text similarity score is combined with the input series of the GRU network to reflect the text similarity influence of neighbor page on the target page series prediction. By finetuning BERT or its variants, we can construct a task-specific neural network to generate representations that are more suitable for our predictions. Then, we select a set of semantic-related series to assist our final prediction.

To this end, the contributions of our paper are as follows:

- We design two method by using the graph structure with internal links and page semantic similarity as the external feature respectively to find the relevant series to assist our prediction.
- Our two methods performed better than baselines on the target dataset, and verified the effectiveness of our finetuning strategy, which provide a possible explanation and enhance the interpretability to a certain extent.

Our method can be used on any time series prediction scenarios, if the series has a link graph structure and textual information.

The rest of this paper is organized as follows. Section 2 shows related work. Section 3 introduces our method on predicting click counts for linked webpages. In Section 4 we describe our CPSE method with textual information. In Section 5, we design evaluation experiments. Section 6 is to address a conclusion and future work.

2. Related work

2.1 Existing Click Count Prediction Methods

Real temporal data can stretch over a long period of time. Traditional methods like ARIMA [16] generally model the problem with predefined temporal features and predict by regressors. Recurrent neural network (RNN) [6] is more advanced to address those problems, by its ability to model long-term historical information of time series. Traditional RNNs, however, suffer from the problem of vanishing gradients and thus have difficulty in capturing long-term dependencies. Compared with traditional RNN, long-short-term memory (LSTM) [10] and gated recurrent unit (GRU) [5], as novel RNN cells can solve long-term dependencies problems and determine the retention of past information and current state through the internal gate structure. The idea of sequence-to-sequence is to split a continuous RNN network into encoders and decoders. Sequence-to-sequence models [4] have overcome these limitations and achieved great success in time series prediction with different pattern [22].

2.2 Graph Embedding

Among various graph embedding methods, DeepWalk [17] uses local information obtained from truncated random walks to learn latent representations by treating walks as the equivalent of sentences. Combining the neural language model Skip-gram [15] and deep learning for graph embedding, this method is a generalization of language modeling to explore the graph through a stream of short random walks. Random walks have been used as a similarity measure for a variety of problems [1][8]. DeepWalk is scalable and trivially parallelizable, which helps us for processing large graphs with millions of nodes and edges. Graph embedding can extract traffic-related neighbors, providing a good estimation of the transition probability from one page to another.

2.3 Pretrained Language Models

Although deep neural models can be impressive in NLP tasks, they would take too much computational resource and time to converge, as well as demanding large labeled datasets to train from scratch [20]. Thus, utilizing pretrained representations and finetuning methods can alleviate the problem. In 2018, Devlin et al. [9] release BERT. Built on multi-layer bidirectional Transformer [21] blocks, BERT is trained on large training corpora based on two tasks: Masked word prediction and next sentence prediction. The first finetuning-based representation model achieves the state-of-the art results on various NLP tasks.

Based on BERT, a number of variants of BERT are proposed, such as Robustly Optimized BERT Pretraining Approach (RoBERTa) [13]. Compared with BERT, the modifications of RoBERTa include: (1) Training the model longer, with larger batches, over more data; (2) removing the next sentence prediction objective; (3) training on longer sequences; and (4) dynamically changing the masking pattern applied to the training data. Additionally, RoBERTa applied the Byte-Pair Encoding (BPE) [19] for text encoding. Using byte-level representations can encode any input text without and not introduce any “unknown” tokens. Through experiments, RoBERTa can match or exceed the performance of all of the post-BERT methods in several situations.

2.4 RNN cells

The standard RNN structure has a ***tanh*** layer for repetitive learning, which can have some drawbacks that cannot handle long dependencies. LSTM is a special type of RNN. LSTM [10] is efficient in learning an over extended time intervals via recurrent backpropagation. Multiplicative gate units learn to open and close access to the constant error. Gated recurrent unit (GRU) [5] is a variant of LSTM, such that GRU simplifies the internal structure and mixes cell states and hidden states, resulting in a simpler model than the standard LSTM model. The differences between the standard LSTM and GRU are not significant, so the choice between the standard LSTM or GRU depends on the specific task.

2.5 Sequence-to-sequence model

The sequence-to-sequence model [4] is a novel structure for sequence generation. It consists of two recurrent neural networks (RNN): The first one acts as an encoder. Because encoder can take the input and maps it to a vector with fixed dimension, we do not need to change the input dimension. Hidden layer passes the hidden state generated by the encoder to the decoder as the initial hidden state. The second one acts as a decoder, taking the vector and maps it to an output sequence. Although initially this method was used for NLP tasks like machine translation, several works [12][27] show it is effective in time series prediction. Compared with the traditional RNN, the Seq2Seq model can handle tasks with different input and output sequence lengths.

In our target datasets, each series is click data on one web page, which have different patterns in magnitude and cycle. Sequence-to-sequence models can overcome such challenge by splitting a continuous RNN network into encoders and decoders.

3. Click-count Prediction with Graph Embedding (CPGE)

The main difference between CPGE and CPSE is extracting other related series by using different external features.

In this section, we firstly describe our proposed method CPGE, utilizing graph embedding, integrated into a sequence-to-sequence model to take advantage of graph structure.

3.1 CPGE Methodology Overview

Figure 1 shows the method overview of the CPGE method. In Figure 1, we aim to discover similar neighbors of the target webpage in terms of the click traffic and link structure. To achieve this, we build a graph where edges are weighted by click sequence similarities. Then, through graph embedding, similar neighbor series can be found. Finally, a group of neighbor series are used to train a sequence-to-sequence model for future series prediction.

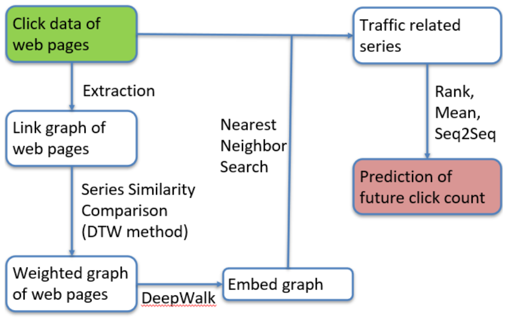

Figure 1. CPGE Overview

3.2 Building Weighted Graph from Webpages

We consider a graph $G = (V, E)$ where nodes V correspond to articles and edges E correspond to links. Each edge is weighted with a certain similarity measured by combing temporal similarity. Regarding time series similarity, Dynamic Time Warping (DTW) [2] can be considered, which computes time series similarity by dynamic programming, obtaining the best alignment for two series. The soft-DTW [7] method uses a differentiable loss function to solve the gradient vanishing problem caused by different alignment strategies.

3.3 Graph Embedding by DeepWalk

In our CPGE, we adopt DeepWalk [17] for learning graph embeddings. DeepWalk consists of two phases. Firstly, a random surfer walks through the graph and generates paths of node sequence within the graph. The random walk sequences are regarded as sentences and the word embedding method Skip-Gram can be applied on them, which produces a set of d dimensions (d=128) vectors where a vector represents a graph node, and the cosine distance between two nodes reflects how close they are in the similarity measure.

3.4 Nearest Neighbor Search and Rank

Since we can tolerate inaccurate results, the approximate nearest neighbor search (NNS) method [25] which does not guarantee to return the actual nearest neighbor in every case can be used. This method takes advantage of random projection and random forest to do NNS for a large number of vectors. With the collected neighbor set, a series similarity value to query series is obtained from Euclidean distance of their embedding vectors. Then, we rank them with respect to the similarity score and select N neighbor series to obtain the average series of them to reduce the input vector size.

3.5 Prediction by Sequence-to-Sequence Model

The Seq2Seq model consists of two RNNs, encoder and decoder. The sequence-to-sequence enables the RNNs to learn from and predict many series with a different pattern. In this paper, GRU and LSTM cells are both tested.

For the RNN cell in encoder and decoder, the formula can be written as formular (1) with the input vector x_t at time t and the hidden state h_{t-1} produced by the previous RNN cell.

$$h_t = f(W^h h_{t-1} + W^x x_t) \tag{1}$$

Here, the function $f(\cdot)$ denotes the operation in each RNN cell, which will change with the different RNN cell type. W^h and W^x denote the weights to learn.

What needs to be mentioned is that, after encoding the input series, the final hidden state of encoders c is logged. Between the encoder and decoder, there is a hidden layer to pass the hidden state generated by the encoder to the decoder as the initial hidden state. Decoders will start by hidden state $h_0 = c$. Decoders update their hidden states by training series. When a RNN cell in decoder produce its output, (2) is used to transform hidden state to the prediction value y_t:

$$y_t = e^{h_t} - 1 \tag{2}$$

4. Click-count Prediction with Semantic Embedding (CPSE)

In addition to CPGE, we design another method called Click-count Prediction with Semantic Embedding (CPSE), which utilizes semantic similarity of neighboring web pages for click count prediction, assuming that pages of the same or similar topics attract similar click traffics. The overview of CPSE is shown in Figure 2.

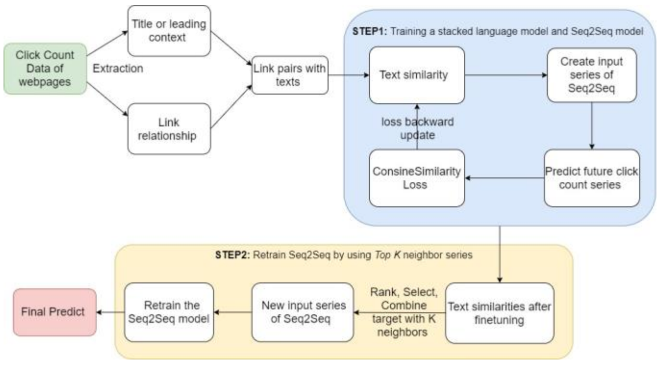
Click Count Data of webpages
Extraction
Title or leading context
Link relationship
Link pairs with texts
STEP1: Training a stacked language model and Seq2Seq model
Text similarity
Create input series of Seq2Seq
loss backward update
ConsineSimilarity Loss
Predict future click count series
STEP2: Retrain Seq2Seq by using Top K neighbor series
Text similarities after finetuning
Rank, Select, Combine target with K neighbors
New input series of Seq2Seq
Retrain the Seq2Seq model
Final Predict

Figure 2. CPSE Overview

As shown in Figure 3, we firstly evaluate the text similarity of each link pair (Target, Neighbor), so that we simultaneously finetune a staked network of BERT (or RoBERTa) and Seq2Seq models in STEP 1. In STEP 2, we utilize the finetuned BERT model to select top-K similar neighbor time series. Using the selected K series, we retrain the Seq2Seq model for final prediction. The sequence-to-sequence model and final prediction of CPSE is identical to CPGE.

Our CPSE Method is a 2-step model: 1) Finetuning a stack of a pretrained language model and Seq2Seq model, and 2) Retraining the Seq2Seq model by using top-K neighbor series selected by the pretrained language model of 1).

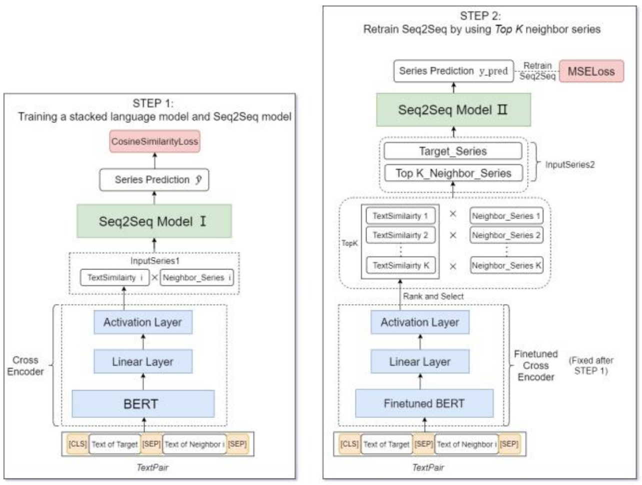

Figure 3. Top-view structure of CPSE

4.1 Text Truncation

We extract the text information such as the title, abstract and leading text of the body of each web page, to capture the topics of the page. We call the text other than the title as the *context* of the web page. In contrast to methods such as word2vec [15], BERT can generate context-aware representation of sentences. For each link of the training dataset, we use the text information of the adjacent two nodes the link to finetune a BERT model or its improved version RoBERTa, to learn sematic similarity the links represent.

BERT has a length limit of 512 on the input token sequence. To train on text pairs of links, we utilize cross encoder [11]. To construct an input sequence for the cross encoder, we concatenate the pair of texts of a target page and its neighboring candidate page into a single sequence *TextPair*. Here, we consider three types of input text sequences, as shown in Figure 4, where each segment separated by the middle [*SEP*] token is truncated into an equal length.

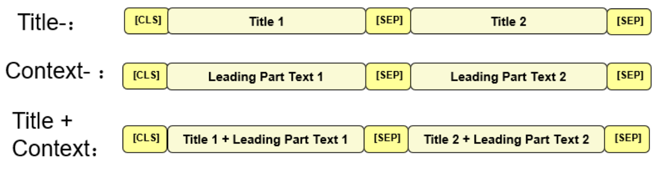

Figure 4. Text truncation for BERT input sequence

4.2 Training a stacked language model and Seq2Seq model

Figure 3 left shows STEP 1, in which finetuning of a stacked network of a pretrained language model and Seq2Seq model is carried out. Our finetuning strategy for STEP 1 is to capture semantic similarity of the texts of each linked node, and the semantic similarity is combined with the corresponding click count sequence. The Seq2Seq is trained over the combined sequences to minimize sequence prediction error, which is back propagated to the pretrained language model, so that the semantic similarity is emphasizing text segments that are important for time series prediction.

The input sequence for the Seq2Seq model in STEP 1 consists of the text token sequence and click count series of a link pair. Firstly, as shown in Eq. (3)-(4), we use the cross-encoder of the BERT encoder layer to obtain the text similarity of the linked pair nodes. We enter a $TextPair$ to the BERT encoder layer. The BERT model produces an output vector, which is passed to the linear layer and the sigmoid function to obtain a value between 0 and 1 to indicate the similarity of the input sentence pair.

$$Emb_{TextPair} = BERT(TextPair) \tag{3}$$

$$TextSimi = \sigma(W \cdot Emb_{TextPair}) \tag{4}$$

Here, $BERT(\cdot)$ indicates the output text embedding by the pretrained BERT model, which is captured at the input of the $[CLS]$ token. W and $\sigma(\cdot)$ are trainable weights of the linear layer and sigmoid activation layer, respectively.

Given a set of click count series $T = \{T_{1,*}, T_{2,*}, \dots, T_{N,*}\}$ of N web pages, we assume that each click series $T_{i,*}$ consists of D days click count value. Let $T_{i,j}$ denote the value of the j^{th} (j =1,2,...,D) day of the i^{th} (i =1,2,...,N) time series $T_{i,*}$. In our method, the click counts series of the target and neighbor series are firstly standardized at each day:

$$Standardize(T_{i,*}) = [\text{std}(T_{i,1}), \dots, \text{std}(T_{i,D})] \tag{5}$$

$$std(T_{i,j}) = log(T_{i,j}) - \frac{1}{D}\sum_{k=1}^{D} log(T_{i,k}) \tag{6}$$

Here, the logarithm is applied on each point $T_{i,j}$ of time series $T_{i,*}$ to reduce the impact of magnitude. The mean value of the standardized points is adjusted to zero.

Let us assume that the target node has time series $T_{i,*} \in T$, and has C link neighbors. We also assume that the time series set of these link neighbors is denoted as $Nei = \{Nei_{1,*}, Nei_{2,*}, \dots, Nei_{C,*}\}, where\ Nei \subseteq T$. We compute the text similarity $TextSimi$ of the target page and the M-th neighbor page (M =1,2,...,C). Each point of the neighbor series $Nei_{M,*}$ is standardized, and multiplied with $TextSimi$, before entered into the Seq2Seq model for generating the predicted click count series $\hat{y}$. The standardized target series y is used as the reference series. The formulars (7)-(9) show details:

$$InputSeries1 = TextSimi \cdot Standardize(Nei_{M,*}) \tag{7}$$

$$\hat{y} = Seq2Seq(InputSeries1) \tag{8}$$

$$y = Standardize(T_{i,*}) \tag{9}$$

Here, $Seq2Seq(\cdot)$ denotes the output vector of the Seq2Seq model, which is explained in the Section 3.6.

In STEP 1, we use cosine similarity loss function between $\hat{y}$ and y as below for the Seq2Seq model:

$$Loss(\hat{y}, y) = 1 - cos(\hat{y}, y) \tag{10}$$

This loss function can reflect the gap between the two series and reduce outliers' influence. STEP 1 is a stacked model with the Seq2Seq model over the cross-encoder output. In the back propagation, after the loss is derived from the Seq2Seq input, the obtained value is used to calculate the partial derivative value of the parameters of the cross-encoder. Thus, the training process in STEP 1 is summarized as:

(1) Calculating the cosine loss on the pair of time series multiplied with the semantic similarity between the texts of the corresponding node pair.

(2) Calculating the gradient of all the parameters in the Seq2Seq model and cross-encoder.

(3) Updating all the parameters in STEP 1.

4.3 Retraining the Seq2Seq model by using Top K neighbor series

After finetuning the stacked model in STEP 1, in STEP 2 we use the language model for ranking neighbor pages by the semantic similarity, and choose top-K neighbor pages. The time series of these pages are coupled with the target time series, and used for retraining the Seq2Seq model.

We utilize the finetuned BERT model of STEP 1 to obtain the text similarity of target with its each neighbor. Since a target page will have multiple link neighbors, we rank all the neighbors by the text similarity and select top-K neighbors to be used in prediction. We expect that prediction over multiple series consisting of the target page and semantically similar neighboring pages can better capture hidden trends of click traffics.

We denote by $nb(j)$ the index of the j-th page in the selected top-K neighbor pages, j=1,...K.

We normalize the similarity values $sim_1, \dots, sim_K$ of the top-K neighbor pages with the target page by the SoftMax function. The neighboring K series $T_{nb(j),*}$, $j = 1, \dots K,$ are weighted by $sim_1, \dots, sim_K$ and summed into one series $Nei\_SumSeries$. The concatenation of $Nei\_SumSeries$ and the target series $T_{i,*}$ becomes the input sequence $InputSeires2$ for the Seq2Seq model in STEP 2. Below we describe details.

$$topK\_sim_i = \frac{exp(sim_i)}{\sum_{j=1}^{K} exp(sim_j)}, \quad i = 1.., K \qquad (11)$$

$$Nei_{SumSeries} = [ne_1, \ldots, ne_D], \; ne_i = \sum_{j=1}^{K} topK_{sim_j} T_{nb(j),i} \; (12)$$

$$InputSeires2 = Concatenate(T_{i,*}, Neigh\_SumSeries) \quad (13)$$

Here, $topK\_sim_i$ is the semantic similarity of the node pair of the target and the i-th page in the top-K neighbor pages.

Since we need to predict the future click count of the target page, we utilize the mean square error loss function for retraining the Seq2Seq model in STEP2. The definition of the MSELoss function is as follows:

$$MSELoss(y\_pred, y) = (y\_pred - y)^2 \qquad (14)$$

Here, y_pred and y are the predicted and true click count series, respectively.

5. Experiments

5.1 Dataset

Our experiments are conducted on the dataset of Web Traffic Time Series Forecasting [18], consisting of real click counts on Wikipedia articles. We utilize 9414 series of 803 days with type "All access_all agent" to do the experiment on the baselines and our proposed methods.

The train-test split is time-based split. As shown in Figure 5, for training a model, we use the click count series from Day 1 to Day 743, and then evaluate the model by using data from Day 744 to Day 773. For testing, we use the data from Day 31 to Day 773 as input to predict the results of the last 30 days. This split method evaluates each model on unseen time period.

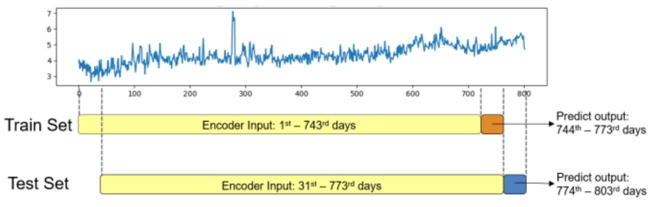

Figure 5: Time-based train-test split

5.2 Baseline methods

To compare with our method, the following baselines are considered:

(1) **LD-Baseline** and **mean baseline** [24]. These simple methods assume time series is stable at one value for the future. LD-Baseline chooses the last value in training data as a prediction. The mean baseline chooses the mean value of the selected period for prediction.

(2) **Autoregressive Integrated Moving Average Model (ARIMA)** [16]. Different from RNN models that train weights by fitting the whole dataset, this model assumes a small number of hand-made features that can fit time-series individually to make predictions. As a result, RNN models learn the global pattern for time series, but the ARIMA model is localized to each series.

(3) **LSTM seq2seq** and **GRU seq2seq** predict by the target time series only. Current results on webpage traffic prediction generated by a GRU sequence to sequence model are the state-of-the-art [28].

5.3 Experimental Settings

In our two methods, a Seq2Seq model is used for final prediction. We choose LSTM and GRU for the seq2seq model. We use Adam as the optimizer with learning rate = 1e-3 and dropout =0.1. During training, we train the model in 100 epochs. In CPSE, the pretrained language model in STEP 1 is either the 'bert-base-uncased' or 'roberta-base'. Due to the different input types, we set the epoch number to be 128, 12 and 12, respectively for 'Title', 'Context' and 'Title+Context'. As for the optimizer, we use the AdamW optimizer with dropout probability 0.1 and base learning rate 2e-5.

5.4 Evaluation

To measure the performance of time series prediction, we consider multiple different evaluation metrics. Root mean squared error (RMSE) and mean absolute error (MAE) are scale-dependent measures. Mean absolute percentage error (MAPE) and Symmetric mean absolute percentage error (SMAPE) are scale-independent measures. In contrast with predicting just one long series, our prediction task is on multiple time series with different magnitudes. Using absolute error which takes the sum of absolute or square value as metrics is not suitable for our task, since they are influenced by such magnitudes. Considering the above, we choose SMAPE as our evaluation metrics, which has stronger tolerance to 0 values and less influenced by magnitude, compared with other evaluation metrics.

$$SMAPE = \frac{100\%}{n} \sum_{i=1}^{N} \frac{|Xi - Yi|}{\frac{|Xi| + |Yi|}{2}} \qquad (15)$$

Here, Xi and Yi are predicted and true click counts of data i, respectively.

5.5 Performance Comparison

We show the results of time series prediction in Table 1. The lower value of SMAPE indicates better performance.

Table 1: Results of click count prediction

Method	SMAPE

Baselines	LD-Baseline		0.3941
	Mean-30		0.3796
	ARIMA		0.2819
	LSTM Seq2Seq		0.2468
	GRU Seq2Seq		0.2373
Proposed CPGE Method	LSTM Seq2Seq + graph		0.2480
	LSTM Seq2Seq + graph + DTW		0.2393
	GRU Seq2Seq + graph		0.2331
	GRU Seq2Seq + graph + DTW		**0.2099**
Proposed CPSE Method (top-K, K=10)	Title-	BERT + LSTM	0.2131
		RoBERTa + LSTM	0.2116
		BERT + GRU	0.1984
		RoBERTa + GRU	0.1960
	Context-	BERT + LSTM	0.2109
		RoBERTa + LSTM	0.2146
		BERT + GRU	0.1924
		RoBERTa + GRU	0.1983
	Title+ Context-	BERT + LSTM	0.2116
		RoBERTa + LSTM	0.2145
		BERT + GRU	**0.1920**
		RoBERTa + GRU	0.1979

(1) CPGE, which is based on graph neighbor series and edge weights are given by DTW, outperforms the existing methods LSTM Seq2seq and GRU Seq2seq, which predict only using the target time series. Especially, GRU Seq2Seq + graph + DTW achieved a significant improvement over all the baselines.

(2) Our CPSE method also significantly outperforms the compared models. CPSE's graph embedding method which reflects semantic similarity between linked articles is also showing effectiveness on both LSTM and GRU Seq2seq models. Among them, CPSE-Title+Context with BERT and GRU combination achieved the state-of-the-art.

(3) Regarding CPGE and CPSE, CPSE appears to have superior results over CPGE. CPGE utilizes temporal similarity of link neighbor series only, while CPSE also considers text similarity, where the text similarity is finetuned to promote text contexts that are sensitive to temporal changes of click counts. CPGE with Title+Context which incorporates the largest number of texts between link pairs appears to be at most exploiting the semantic similarity.

5.6 Effect of Finetuning Strategy on Pretrained Language Models

To clarify the effect of the finetuning strategy of pretrained language models, we perform an ablation study on the text similarity stage. In this experiment, we use 'Title' as the input text type and set K=10 for selection of top-K neighbor series.

Firstly, for pretrained language models without any finetuning, we directly use 'bert-base-uncased' and 'roberta-base' to produce the text similarity of each link pair, on text input Title. Then, they are compared with our models which finetune BERT or RoBERTa by the back propagation from the stacked Seq2Seq model, which learns from prediction error on click count sequences. Table 2 shows the SMAPE scores of the models with or without finetuning, stacked with LSTM or GRU for the Seq2Seq model.

Table 2: Results on finetuning strategy

Method	BERT-Pretrained	BERT-Finetune	RoBERTa-Pretrained	RoBERTa-Finetune
LSTM	0.2216	**0.2131**	0.2198	**0.2116**
GRU	0.2139	**0.1984**	0.2108	**0.1960**

In Table 2, we can see that our finetuning strategy is showing effectiveness on both LSTM and GRU, achieving 5.5% improvement in average over the pretrained language models without finetuning.

6. Conclusion and future work

In this paper, for predicting future page click counts, we proposed two methods CPGE and CPSE, which utilize click count sequences of articles link neighbors and/or semantic neighbors. CPGE builds graph embeddings from the links of web pages to perform click count prediction over link neighbors, where temporal similarity between click count series of two pages is reflected in training graph embeddings. In CPSE, pretrained language models BERT and RoBERTa are adopted to evaluate semantic similarity of pages, which promote pages of similar topics. The pretrained language models are finetuned by the backpropagation form click count prediction, so that text similarities that have high correlation with time series are promoted. Our results show our two methods both improve the performance and CPSE achieved the state-of-the-art.

In the future, we will pay more attention on finetuning language models and improve layers of the Seq2Seq model for final prediction, by adding attention layers.

References

[1] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using page rank vectors[C]. 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06). IEEE, 2006: 475-486.

[2] R. Bellman and R. Kalaba, On adaptive control processes[J]. IRE Transactions on Automatic Control, 1959, 4(2): 1-9.

[3] P. Chakraborty, M. Marwah, M. Arlitt, et al. Fine-grained photovoltaic output prediction using a bayesian ensemble[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2012, 26(1).

[4] K. Cho, B. Van Merriënboer, D. Bahdanau, Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259, 2014.

[5] J. Chung, C. Gulcehre, K. H. Cho, et al. Empirical evaluation of gated recurrent neural networks on sequence modeling[J]. arXiv preprint arXiv:1412.3555, 2014.

[6] J. T. Connor, R. D. Martin, L. E. Atlas. Recurrent neural networks and robust time series prediction[J]. IEEE transactions on neural networks, 1994, 5(2): 240-254.

[7] M. Cuturi, M. Blondel. Soft-dtw: a differentiable loss function for time-series[C]//International Conference on Machine Learning. PMLR, 2017: 894-903.

[8] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. Knowledge and Data Engineering, IEEE Transactions on, 2007, 19(3):355-369.

[9] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[10] S. Hochreiter, J. Schmidhuber. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[11] S. Humeau, K. Shuster, M. A. Lachaux, P. Weston. Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring[J]. arXiv preprint arXiv:1905.01969, 2019.

[12] S. Hwang, G. Jeon, J. Jeong, et al. A novel time series based Seq2Seq model for temperature prediction in firing furnace process[J]. Procedia Computer Science, 2019, 155: 19-26.

[13] Y. Liu, M. Ott, et al. Roberta: A robustly optimized BERT pretraining approach[J]. arXiv preprint arXiv:1907.11692, 2019.

[14] Z. Liu, M. Hauskrecht. A regularized linear dynamical system framework for multivariate time series analysis[C]. Proceedings of the AAAI Conference on Artificial Intelligence. 2015, 29(1).

[15] T. Mikolov, K. Chen, G. Corrado, et al. Efficient estimation of word representations in vector space[J]. arXiv preprint arXiv:1301.3781, 2013.

[16] T. C. Mills. Time series techniques for economists[M]. Cambridge University Press, 1991.

[17] B. Perozzi, R. Al-Rfou, and S. Skiena. DeepWalk: Online learning of social representations. In KDD, 2014.

[18] N. Petluri and E. Al-Masri, "Web Traffic Prediction of Wikipedia Pages," 2018 IEEE International Conference on Big Data, Seattle, WA, USA, 2018, pp. 5427-5429.

[19] Y. Shibata, T. Kida, S. Fukamachi, et al. Byte Pair encoding: A text compression scheme that accelerates pattern matching[R]. Technical Report DOI-TR-161, Department of Informatics, Kyushu University, 1999.

[20] C. Sun, X. Qiu, Y. Xu, X. Huang, et al. How to fine-tune BERT for text classification? [C]//China National Conference on Chinese Computational Linguistics. Springer, Cham, 2019: 194-206.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin. Attention is All You Need, CoRR, vol. abs/1706.03762, 2017.

[22] N. Wu, B. Green, X. Ben, et al. Deep transformer models for time series forecasting: The influenza prevalence case[J]. arXiv preprint arXiv:2001.08317, 2020.

[23] Yue Wu, Jose Miguel Hernandez-Lobato, and ´ Zoubin Ghahramani. Dynamic covariance models for multivariate financial time series. In ICML, pages 558–566, 2013.

[24] "An-introduction-to-sequence-prediction",http://data-mining.philippe-fournierviger.com/an-introduction-to-sequence-prediction.

[25] "Nearest_neighbor_search:Approximate_nearest_neighbor",https://en.wikipedia.org/wiki/Nearest_neighbor_search#Approximate_nearest_neighbor

[26] "Research: Wikipedia clickstream", https://meta.wikimedia.org/wiki/Research:Wikipedia_clickstream#Applications

[27] "seq2seq-signal-prediction: Signal forecasting with a Sequence-to-Sequence Recurrent Neural Network (RNN) model in TensorFlow",https://github.com/guillaumechevalier/seq2seq-signal-prediction

[28] Web Traffic Time Series Forecasting- 1st place solution, https://www.kaggle.com/c/web-traffic-time-series-forecasting/discussion/43795