CS230

CS230 Final Report: Multivariate Wikipedia Web Traffic Time Series Forecasting Using Clickstream Graph Embedding

Eric Liu
Stanford Center for Professional Development
Stanford University
ericxliu@stanford.edu

1 Problem Statement

Accurately forecasting traffic of web pages can provide guidance for capacity planning, input for application performance optimization, and many more practical use cases. In this work, we focus on this page forecasting problem by using the daily views of the Wikipedia pages time series dataset.

Naively, for any multivariate forecasting problem, a model can individually learn and predict the time series based on historical values. This method, however, might not scale up well, if dealing with a very large number of time series. Intuitively, a better approach is to learning conceptually similar time series and forecast their trend. In this work, we explore a few time series embedding techniques, including a random integer embedding, a simple trainable embedding layer, graph embedding using the Clickstream for user jumping from one page to the other, as well as Wikipedia2Vec, an embedding based on page content. Overall, we hypothesise a good embedding, like Clickstream and Wikipedia2Vec should outperform others since they provide a richer context for the models.

2 Dataset and Features

2.1 Time Series Data Source

The main data source is the data dump of the Wikipedia page per entity per day from the Wikimedia Foundation's Analytics Engineering team. [1]. Specifically, the raw Pageview table has the following format.

- `wiki_code (subproject.project)`
- `article_title`
- `page_id`
- `daily_total`
- `hourly_counts`

We use data from Jan 1st, 2018 to Jan 1st, 2021, spanning in total of 1097 days.

2.2 Page Feature Data Source

2.2.1 Click-Stream and Node2Vec

In addition, we use the Wikipedia Clickstream graph[2] , which contains the relationship between pages in terms of how often people navigate from one page to another. The data format is transformed into:

- `prev_page`
- `curr_page`
- `n_clicks`

where `prev_page` and `curr_page` refer to the referrer and resources wiki pages pairs, and `n_clicks` refers to number of click through in a month. We use the extract of Clickstream data in Jan, 2018.

To better incorporate wikipage and its click stream graphs into the time series model, we choose Node2Vec as a method of graph embedding. Node2Vec is a popular graph node embedding methods. Here we use it to encode the

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

click stream graph to embed a click steam graph (weighted) to represent a node. [3][5]. The embedding output features a graph walk and if two nodes has similar graph walk, it means they are more likely belong to the same click-stream.

2.2.2 Page Entities and Wikipedia2Vec

In additional, we also introduce a representation of the page content itself, along with another framework: Wikipedia2Vec to get a appropriate embedding results for the page content. It is a framework for embedding wikipage words and entities. We use the the pretrained model [10] which has majority of the pageview in our data.

2.3 Data Preprocessing

2.3.1 Training and Validation Set

The time series data are split into training and validation set by 1,037 and 60. That means we use 1,037 days to predict next 60 days' page view. The following data processing and embedding methods are applied to both training and validation dataset. Here in the graph shows some sample time series for training and validation set, respectively.

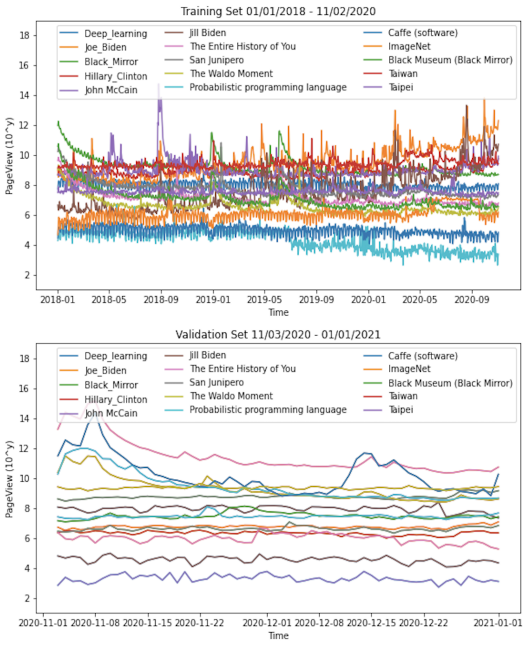

Figure 1: Training and Validation Time Series

2.3.2 Data Selection

We collect wiki pages which have rich click through activities and use it as the list to pull the page view time series data. At the same time, we also use a random selection of pages in to a different training data. Our assumptions are

- If multiple pages are trained together, we would result in better model performance than individual time series
- If multiple pages are highly correlated, defined as there are high degree of click through connectivity, the model would result in a even better result.

The click-stream data can be represented as graph, so we firstly encode the click-stream data into a graph and analyze the click-stream data with multiple graph algorithms to find the most highly connected nodes. We use methods where to sum of the in-degree and out-degree to find the top 100 nodes, and find all related pages that are directly linked to these nodes within two degree. Then excluding pages that has missing values in the time series, unavailable embedding data, as well as some nodes with too few clicks (doing this to reduce the number of data due to the limitation of the computation power without hurting the overall graph structure), totally we get 30,545 page nodes.

Then, we use these 30,545 pages to get the time series data over 3-year span to get their daily view count, which is going to our main data to be feed in to the models.

2.3.3 Time Series Transformation

In order to fit into the framework of sequential model, we choose window size = 7, 35 as a choice to reformat the data in to multiple batches of data. That is, each batch of input data has 7/35 days' data, used to predict the next one day's page view.

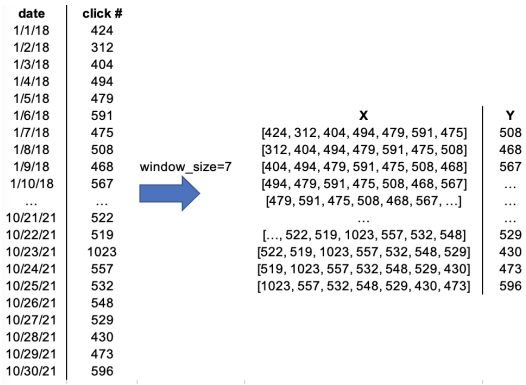

Figure 2: Time series transformation to windowed dataset

2.3.4 Embedding and Data Concatenation

To improve the model performance, we explore introducing varieties of embedding methods in

the Deep Learning models, in additional the time series dataset. We concatenate embedding array to the number of windowed function for Fully-Connect Model. For LSTM, as we cannot simply feed data by concatenate windowed view (x in Figure 1) with embedding array, we broadcast the embedding array to each step of windowed functions.

Below diagram shows how to concatenate embedding data:

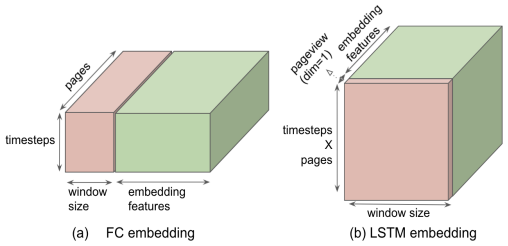

Figure 3: Embedding Dataset

Here our final processed dataset has following dimensions:

- `window_size`. Number of days for window sliding data. Hyper parameter to tune: 7, 35.
- `embedding_features`.
- `timesteps`. Number of windowed time series, which numerically equals to `#_days_time_series - window_size` = 1,097 in this project.
- `pages`. Number of wikipages.

In the model, there are a few types of embedding methods we explore:

- **Integer.** It is a arbitrarily set embedding vector with dimension of 1, where each page has an unique integer.
- **Learned.** A trainable embedding layer maps the index integer to a low dimension floating point vector.
- **Node2Vec.** Embed each node in the click-stream graph into a vector with dimension of 32.
- **Wikipedia2Vec.** Embed wikipage content to a vector with dimension of 100.

Detailed comparison of the model performance is shown in section 4.

3 Methods

We setup the baseline using two native forecasting methods that do not require learning: Persistence and moving average. The persistence model is simple a $lag(1)$ function, namely use the last value as the prediction. The moving average model is to use the mean of values of the window as the forecast. In this experiments, we choose the window size 7, which exploits the weekly seasonality of the time-series. Note that the native methods make the prediction individually; Therefore, they do not use any embedding features.

Next, we implement a fully connected neural network in the Keras sequential model with two different window size, 7 (FC-7) and 35 (FC-35), as hyper-parameter. For all layers, we use Rectified Linear Unit (ReLU) as the activation function. Specifically, we have the following layers.

- `Dense(200, activation="relu")`
- `Dense(100, activation="relu")`
- `Dense(100, activation="relu")`
- `Dense(10, activation="relu")`
- `Dense(1, activation="relu")`

Note that in the Learned Embedding experiment, we need to add an Embedding layer before the first fully connected layer to convert the integer embedding to a vector embedding.

Last but not least, we explore the recurrent neural network architecture. In particular, we design a deep Long short-term memory (LSTM) model with Batch normalization and dropout.

- `BatchNormalization()`
- `LSTM(20, return_sequences=True, dropout=0.2)`
- `BatchNormalization()`
- `LSTM(20, return_sequences=True, dropout=0.25)`
- `BatchNormalization()`
- `LSTM(10, return_sequences=True, dropout=0.2)`
- `Flatten()`
- `Dense(256, activation="relu")`
- `Dense(64)`
- `Dense(10)`

For all trainable models (FCs and LSTM), we use Huber loss to fit the data. The Huber loss combines squared loss and absolute loss. It is a quadratic function for small value less than the threshold δ and linear for large value bigger than δ. We choose it for its insensitive to outliers in data.

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases} \quad (1)$$

		No	Integer	Learned	Wikipedia2Vec	Node2Vec
Persistence		371				
Moving Average-7		497				
FC-7	Train	303	316.7	301.1	306.3	296.9
	Valid	299.2 (baseline)	309 (-3.28%)	298.4 (0.27%)	300.8 (-0.53%)	295.6 (1.20%)
FC-35	Train	281.4	286.3	280.6	285.5	279.4
	Valid	266.3 (baseline)	269.8 (-1.31%)	266.4 (-0.04%)	268.7 (-0.90%)	265.5 (0.30%)
LSTM	Train	371.6				
	Valid	487				

Table 1: The results for the experiments with different models and embedding methods

We use the Mean Absolute Error (MAE) as the key metric to compare the performance of different models. MAE has the form

$$MAE = \left[\frac{1}{nT} \sum_{i=1}^{n} \sum_{t=1}^{T} \left| \frac{Y_t^{(i)} - \hat{Y}_t^{(i)}}{Y_t^{(i)}} \right| \right] \quad (2)$$

, where $Y_t^{(i)}$ is the observed data of page i in window t, $\hat{Y}_t^{(i)}$ is the counterpart of our predication. We choose the MAE over the mean square error for the same concerns of outliers.

We use early stopping against the validation set as the regularization method for FC models to avoid over-fitting. we also used additional dropout in the LSTM layers to further regularize the large recurrent models. We use the popular Adam optimizer for its fast training speed and adapt learning rate.

On the hardware side, we use the Colab Pro+ product from Google for compute along with the Google Cloud Storage for storing data, model, and logs. We are allocated a single Nvidia Tesla P100 GPU coming in an dual core Intel Xeon Haswell virtual machine with 120 GB of memory. On the software side, we implement all the models using Keras APIs in Tensorflow 2.7.0 in Python.

4 Results and Discussion

4.1 Result

Table 1 shows performance of the models we have tried. Our baseline are two non-learning native models: persistence and 7-day moving average. As a result, the fully Connect Models far outperforms both of the baselines and the LSTM model barely outperforms one of the easier baseline.

For a given fully connect models, we also compared different embedding methods with no embedding information as the baseline. In this case, the results is a little bit mixed with only Node2Vec consistently outperforms the baseline and all other mode performs either worse or on par with the baseline.

Initially, we were surprised by the good performance from the no embedding model, but then we quickly realized that the way we use 1D convolution on a 3D tensor is already providing a form of potential embedding. To confirm this hypothesis, we retrained the models with random shuffling the page dimension of the validation set. After the random shuffling, all models fail to converge.

Therefore, it is expected that the random integer embedding consistent performs worse that the baseline, given it only adding noise into the model. For the trained embedding layer, it does not add new features into the model, which make its performance on par with the baseline.

As for the two most complicated embedding, Node2Vec consistently outperform the baseline, which confirms our hypothesis. On the other hand, Wikipedia2Vec consistently underperforms the baseline. Wikipedia2Vec is mostly based on the contexts of neighboring words, which likely does not provide additional information to this forecasting issue.

4.1.1 Node2Vec Embedding discussion

To further buildup the intuition, we use two methods to visualize the Node2Vec Embedding results. First, we use the Tensorboard to project the high dimension embedding into low dimensions. Figure 4 shows the 2D projection of 200 points that are most closest to the page "Joe Biden" defined by the cosine distance using the principal component analysis (PCA). As one can see, all the terms are conceptually related to the family of Joe Biden or the president or the presidential election.

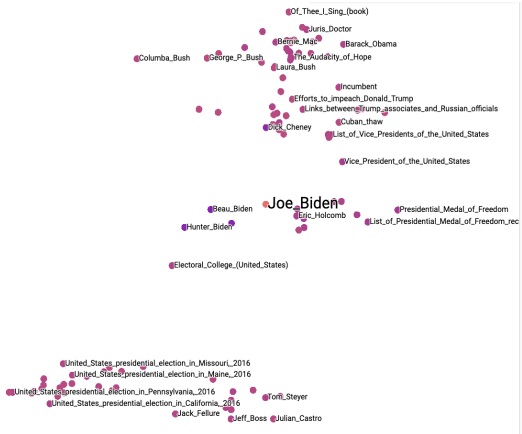

Figure 4: Wikipedia2Vec PCA Projection

The second approach we build the intuition is that we 1) pick three category and their closest neighbors in the Node2Vec Embedding space, 2) calculate their pairwise correlation, 3) use an unsupervised hierarchical clustering algorithm to sort the list 4) visualize the time series in a heat map.

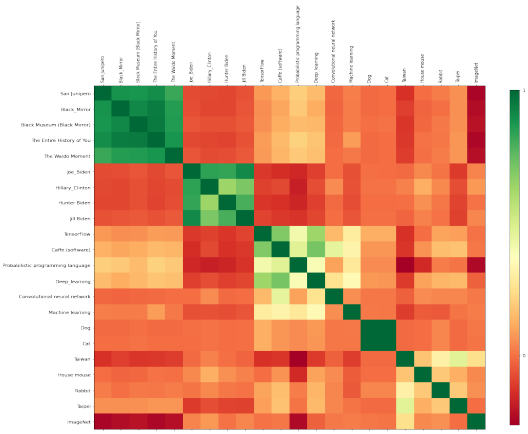

Figure 5: Time series correlation and clustering

As one can see from this graph, the time series of closer pages in the embedding layers in general have higher correlation.

5 Future work

Given more time and computational resources, we would like to first fine tune the LSTM model to achieve better performance and study the effects of embedding on them. Next, we would like to compare our method with the state of art commercial service such as the DeepAR+ algorithm [7] from Amazon and Prophet [13] from Meta. Furthermore, we would also like to leverage a transformer architecture on this problem.

6 Related Works

Many previous works have tackled this multivariate time series forecasting problem, primarily motivated by the Kaggle competition in 2016. [4] [8] [12] [14] have used the Wikipedia web traffics to predict Wikipedia web traffics using various machine learning and deep learning technique, including linear regression, generalized linear model, NN, RNN, LSTM and ensemble techniques. We draw heavy inspiration from those previous work in designing the neutral network architecture used in this paper.

Two previous projects [9] [6] from CS229 and CS230 also tried. The first one using conventional machine learning with generalized linear model and shadow NN with simple embedding of the pages for prediction. The other project reformulate the forecasting problem as a cache replacement policy and developed LSTM network architectures with custom loss function to minimize false positives.

We also have found one previous work [11] create graph embedding using the Wikipedia Clickstream data to formulate a prerequisite graph.

References

[1] Pageview complete dumps. Available at `https://dumps.wikimedia.org/other/pageview_complete/readme.html`.

[2] Research:wikipedia clickstream. Available at `https://meta.wikimedia.org/wiki/Research:Wikipedia_clickstream`.

[3] Scalable feature learning for networks. Node2Vec by Standford SNAP.

[4] Roberto Casado-Vara, Angel Martin del Rey, Daniel Pérez-Palau, Luis de-la Fuente-Valentín, and Juan M. Corchado. Web traffic time series forecasting using lstm neural networks with distributed asynchronous training. *Mathematics*, 9(4), 2021.

[5] Elior Cohen. node2vec: Embeddings for graph data, Apr 2018.

[6] Shuyang Du, Manish Pandey, and Cuiqun Xing. Modeling approaches for time series forecasting and anomaly detection. 2017.

[7] Valentin Flunkert, David Salinas, and Jan Gasthaus. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *CoRR*, abs/1704.04110, 2017.

[8] Naveena Mettu and T. Sasikala. *Prediction Analysis on Web Traffic Data Using Time Series Modeling, RNN and Ensembling Techniques*, pages 611–618. 01 2019.

[9] Anthony Miyaguchi, Shaon Chakrabarti, and Nicolai Garcia. Forecasting wikipedia page views with graph embeddings. 2019.

[10] S. Ousia. Pretrained embeddings - wikipedia2vec., 2021. Available at: `https://wikipedia2vec.github.io/wikipedia2vec/pretrained/`.

[11] Mohsen Sayyadiharikandeh, Jonathan Gordon, Jose-Luis Ambite, and Kristina Lerman. Finding prerequisite relations using the wikipedia clickstream. In *Companion Proceedings of The 2019 World Wide Web Conference*, WWW '19, page 1240–1247, New York, NY, USA, 2019. Association for Computing Machinery.

[12] Philipp Singer, Denis Helic, Andreas Hotho, and Markus Strohmaier. Hyptrails: A bayesian approach for comparing hypotheses about human trails on the web. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1003–1013, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[13] Sean J. Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[14] Yujia Yang, Shi Lu, Huan Zhao, and Xiaoqian Ju. Predicting monthly pageview of wikipedia pages by neighbor pages. In *Proceedings of the 2020 3rd International Conference on Big Data Technologies*, ICBDT 2020, page 112–115, New York, NY, USA, 2020. Association for Computing Machinery.