# Wikipedia Web Traffic Forecasting

**Yu Sun**
ys3225@nyu.edu

**Yicheng Pu**
yp653@nyu.edu

**Dinglun Fu**
df1777@nyu.edu

## Abstract

In this paper we built and tested different models for predicting Wikipedia webpage's visits. Our experiments ranged from RNN, Seq2Seq, CNN, Kalman Filter to Gaussian Process. We analyzed different models' drawbacks and concluded the best models in the end. We also confirmed the existence of each page's patterns and a general pattern across different pages. Our work could be helpful for other similar tasks.

## 1 Introduction

Web traffic volume is critical for a website's operation, being able to forecast web traffic would greatly benefit website owners from saving unnecessary cost, preventing server crashes and detecting abnormal visiting volumes. Google released this dataset of Wikipedia's traffic history as an competition of forecasting future web traffic on Kaggle in 2017. Many teams achieved great performance with different approaches, ranging from traditional methods like Kalman Filter to new methods like RNN and CNN.

There are two ways to solve this problem: either train different models for different pages, or train a single model for all pages. In this paper, we would firstly discuss our experiments on forcasting values page by page. We took Kalman Filter as baseline and tried Gaussian Process Model with different kernel types. Then we would talk about using a single algorithm for all articles.

Unlike the Kaggle competition, our goal is not to achieve the best performance of original dataset, but to compare and analyze different models' performance and capacity on capturing complex patterns in time series.

## 2 Related Work

On Kaggle, solutions for this task are very diverse, the first place solution is a modified Seq2Seq model combined with many hand-crafted features. The features include page's meta-data (country,agent,language,etc.), page view's popularity, year-to-year autocorrelation and quarter-to-quarter autocorrelation. For the Seq2Seq model, the author found adding a year ago and aquarter ago's pageview as extra input to both encoder and decoder greatly improves the performance. Other methods, for example, Klaman Filter, also ranked at 8th place, simple feed-forward neural networks with hand-crafted features was ranked 7th place. Some other high-ranking solutions also used ensemble models.

The main aim in analyzing any time series and predicting the future values based on the old observations is mathematical model that is used[1]. The most popular model would be Autoregressive Integrated Moving Average (ARIMA)[4] and Neural Network[3]. However, the obvious drawback of ARIMA is that the data needs to show the evidence of stationarity, which means it's not suitable for non-linear time series forcasting. However, it's still challenging to forecast the future values of large amount of time series at one time.

## 3 Data

The dataset contains 145,063 records of time series in total, where each record represents the number of daily views of a Wikipedia article page from July 1st, 2015 up until September 10th, 2017. The dataset provides each article page with its name, language, source of traffic. In short, we have around 145k Wikipedia pages, and we are given 803 days of traffic of these pages. A few samples of the data will be showed in Figure 1:

| | Page | 2015-07-01 | 2015-07-02 | 2015-07-03 | 2015-07-04 | 2015-07-05 | 2015-07-06 | 2015-07-07 | 2015-07-08 | 2015-07-09 | ... | 2017-09-01 | 2017-09-02 | 2017-09-03 | 2017-09-04 | 2017-09-05 | 2017-09-06 | 2017-09-07 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2NE1_zh.wikipedia.org_all-access_spider | 18.0 | 11.0 | 5.0 | 13.0 | 14.0 | 9.0 | 9.0 | 22.0 | 26.0 | ... | 19.0 | 33.0 | 33.0 | 18.0 | 16.0 | 27.0 | 29.0 |
| 1 | 2PM_zh.wikipedia.org_all-access_spider | 11.0 | 14.0 | 15.0 | 18.0 | 11.0 | 13.0 | 22.0 | 11.0 | 10.0 | ... | 32.0 | 30.0 | 11.0 | 19.0 | 54.0 | 25.0 | 26.0 |
| 2 | 3C_zh.wikipedia.org_all-access_spider | 1.0 | 0.0 | 1.0 | 1.0 | 0.0 | 4.0 | 0.0 | 3.0 | 4.0 | ... | 6.0 | 6.0 | 7.0 | 2.0 | 4.0 | 7.0 | 3.0 |
| 3 | 4minute_zh.wikipedia.org_all-access_spider | 35.0 | 13.0 | 10.0 | 94.0 | 4.0 | 26.0 | 14.0 | 9.0 | 11.0 | ... | 7.0 | 19.0 | 19.0 | 9.0 | 6.0 | 16.0 | 19.0 |
| 4 | 52_Hz_I_Love_You_zh.wikipedia.org_all-access_s... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 16.0 | 16.0 | 19.0 | 9.0 | 20.0 | 23.0 | 28.0 |

Figure 1: Sample Data

The original dataset used in the Kaggle competition contains a lot of time series that has missing values and highly noisy web spider visiting data. Since our main objective is to compare different models' capability of learning complex time series pattern, instead of dealing with noise and NA values, we removed all missing values and spider data for all experiments. We only tested original dataset on two final models–Gaussian Process and CNN in the very end.

## 4 Problem Definition

In this project, we aim to forecast the future values of multiple time series. We analyzed the deep features and potential patterns within this Wikipedia article traffic dataset to detect explicit trends and periodicity. At the end, we were able to utilize the information we obtained to forecast the future traffic of these Wikipedia articles traffics. More precisely, the input of this problem is times series data of multiple Wikipedia pages, and the output is the traffic of these pages in the future 63 days.

## 5 Algorithms

### 5.1 Traditional Models

In this section, we utilized two traditional models Kalman Filter and Gaussian Process algorithms to tackle multiple time series one by one. For each web page, our goal was to predict the page's future visits with length l , $\tilde{x} \in R^l$ with known its m historical data, $x \in R^m$.

#### 5.1.1 Kalman Filter

The Kalman Filter has been recorded as one of most standard and optimal models for many time series and general data analysis tasks. Therefore we use Kalman Filter as our first model to experiment on the data.

Given the original setup of Latent State Model that $z_i = Az_{i-1} + w_i$ and $x_i = Cz_i + v_i$ where $w_i$ and $v_i$ are i.i.d. Gaussian noise, the goal of Kalman Filter is to computer posterior distribution of the latent state $z_i$ given past observations $\{x_1, x_2, ...x_i\}$. The posterior distribution are computed by:

$$\mu_{i|i} = \mu_{i|i-1} + K_i(x_i - C\mu_{i|i-1})$$

$$\Sigma_{i|i} = \Sigma_{i|i-1} - K_i C\Sigma_{i|i-1}$$

where $K_i = \Sigma_{i|i-1}C^T(C\Sigma_{i|i-1}C^T + R)^{-1}$, $\Sigma_{i|i-1} = A\Sigma_{i-1|i-1}A^T + Q$ and $\mu_{i|i-1} = A\mu_{i-1|i-1}$

### 5.1.2 Gaussian Process

One of the model we use for this project is Gaussian Process regression with multiple kernels. A Gaussian Process for time series data is $X = \{1, 2, 3, 4....n\}$ as time index, and corresponding $Y = \{y_1, y_2, ....y_n\}$, where every finite set of Y with respect to their time indexes is in a consistent multivariate Gaussian distribution. That is:

$$f(x) \sim GP(m(x), K(x, x'))$$

Where $m(x)$ is the mean function and $K(x, x')$ is the covariance function, which is also known as kernel function. In this project, we experiments multiple different kernels and their combinations. The kernel we use in this final report is:

$$K(x, x') = K_{rbf}(x, x') + K_{mlp}(x, x')$$

where

$$K_{rbf}(x, x') = exp(-\frac{(x - x')^2}{2\sigma^2})$$

and $K_{mlp}$ is the Multi Layer perceptron kernel, also known as sine kernel that

$$K_{mlp}(x, x') = \sigma^2 \frac{2}{\pi} asin(\frac{\sigma_w^2 x^T x' + \sigma_b^2}{\sqrt{\sigma_w^2 x^T x + \sigma_b^2 + 1}\sqrt{\sigma_w^2 x'^T x' + \sigma_b^2 + 1}})$$

## 5.2 Neural Networks Models

Traditional models could only be trained on a single time series, which prevents them from learning patterns across different pages. Also, training thousands of different models could be very time consuming. In our case, we have more than 100k pages in total, which means that we need to build models for 100k times.

So in this section, we explored different algorithms to tackle multiple time series at the same time. Thus, our problem needs be formulated as given m historical data points for n pages $X \in \mathbb{R}^{n \times m}$, predicting future visits with length l $\tilde{X} \in \mathbb{R}^{n \times l}$ with a single model. We normalized all time series by subtracting mean and divided by 2 times std for all RNN models.

### 5.2.1 Linear Regression

We take the linear regression as the baseline of non-parametric models. In this model, we directly regard historical time series data $X$ as features, the future data $\tilde{X}$ as outputs and update the weight matrix $W \in \mathbb{R}^{m \times l}$ and bias $b \in \mathbb{R}^{n \times l}$with the loss of SMAPE by optimizer of stochastic gradient descent.

$$WX^T + b = \tilde{X}$$

### 5.2.2 Recurrent Neural Network

The Recurrent Neural Network(RNN) is a popular model for sequence to sequence problems with its unique memory unit to store previous information. Basicly, we expect RNN model to summarize the underlying features in historical data and then use fully connected(FC) layer to predict a future period of visits. We use GRU here since it could better capture the long term dependence with relatively less parameters. The hidden state for each time t should be calculated as following equations.

$$\tilde{h}^t = tanh(WX^t + U(r^t \odot h^{t-1}))$$

$$h^t = (1 - u^t) \odot h^{t-1} + u^t \odot \tilde{h}^t$$

where $r^t$ is the reset gate, $u^t$ is the update gate.
However, the values of consecutive days are not so relevant since our data is full of huge oscillations. Even though RNN could capture some long-term dependencies, with such limited feature size and long sequence length, the model is hardly feasible to detect the pattern of different time series. From the aspect of architecture, if we directly feed the sequence of daily visits values to the RNN, the input of each neuron would be 1, so the weights inside each unit is limited. Thus, we explore different methods to help model capture the patterns in a larger feature dimension.

- **Phase-to-Phase** One basic way to expanding feature dimension is slicing the time series of 1 dimension into sequences of phases. Daily visits only represent the numerical value of this day, while a period of data could contain some underlying patterns. By feeding a phase of visits to each neuron, we could make long term recurrent process shorter by reducing the number of neurons in each layer. For example, we could obtain a sequence of phases with length of 12 by reshaping a tensor in shape of [360,1] to [12,30]. And then feed each phase with size of 30 to the RNN unit. In this way, we hope the model could better capture the dependencies among phases. And then obtain output from the last hidden state through the FC layer.
- **Convolutional Neural Network** Even though the phase-to-phase method could expand the embedding size, the embedding is not trainable since we fix the input as the actual values of phases. However, the Convolutional Neural Network (CNN) could generate embedding by passing number of filters through the series. In this way, we could expect it to capture the underlying patterns by updating the embedding. For example, setting the number of filters as 50, kernel size as 100 and stride as 20 then operating the 1D convolution with a time series with size of 360, we could get the output with shape of (14,50). Similarly, we feed the sequence with length of 14 and embedding size of 50 into the GRU, put the memory states to FC layer and obtain the ouput.

### 5.2.3 Sequence-to-Sequence Model

Seq2Seq model can comprehend historical sequential data through an encoder structure into a fixed-length vector and then reconstruct into its original form through the decoder structure[2]. We use GRU both as encoder and decoder structure. As showed in Figure 2, the encoder part is like above naive RNN model and output the last hidden state $h_{en}^m$. At the first time point of decoder, we pass $h_{en}^m$ through a dummy layer as the input of decoder GRU. For the following time, we use the same dummy layer to transfer the output of last neuron as the input of current cell. The hidden state of encoder and decoder for each time t should be calculated as following equations.

$$\tilde{h}_{en}^t = tanh(W_{en}X^t + U_{en}(r_{en}^t \odot h_{en}^{t-1}))$$

$$h_{en}^t = (1 - u_{en}^t) \odot h_{en}^{t-1} + u_{en}^t \odot \tilde{h}_{en}^t$$

$$y^t = f(h_{de}^t)$$

$$\tilde{h}_{de}^t = tanh(W_{de}y^{t-1} + U_{de}(r_{de}^t \odot h_{de}^{t-1})$$

$$h_{de}^t = (1 - u_{de}^t) \odot h_{de}^{t-1} + u_{de}^t \odot \tilde{h}_{de}^t$$

where $r_{en}^t$, $u_{en}^t$ is the reset gate and update gate for encoder, $r_{de}^t$, $u_{de}^t$ is the reset gate and update gate for decoder. $f$ is the dummy function to modify the output of time t. Basically, we could take it as a linear mapping. However, as we mentioned in RNN model, the input size of 1 would limit model to explore more information. Thus, we also try a method of Position Encoding to handle this problem.

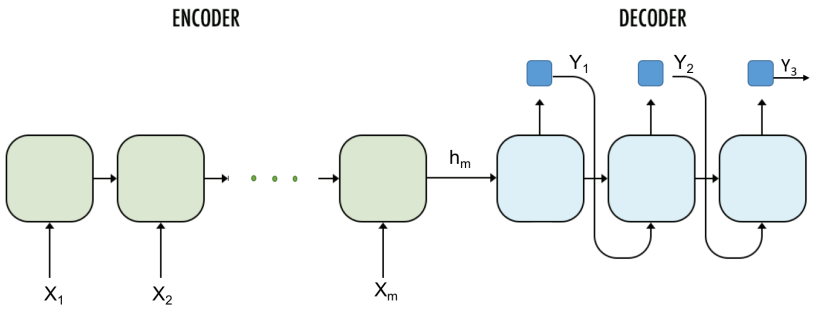


Figure 2: Seq2Se1 Model Architecture

- **Position Encoding** To add more features to input, we encoded the day of week as a one-hot vector P with length 7. For each time step t, we multipled P by input scalar $x_t$ and send it into a RNN cell of Seq2Seq model. For each output $h_t$ from a RNN cell, we multiplied it with P to get a scalar output.

#### 5.2.4 Rethinking using RNN model

As showed in Table 3, all of the RNN models we tried did not get satisfying results. Through analysis, we concluded that normal RNN framework might not fit in this task.

| Model | Best SMAPE |
|---|---|
| Linear | 52.78 |
| 1-D Seq2Seq | 67.57 |
| Seq2Seq+position encoding | 50.34 |
| Phase2Phase | 65.39 |
| RNN+CNN | 68.35 |

Table 1: Comparison of RNN models

In Seq2Seq model, we predict the next time step $Y_t$ by hidden memory $H_{t-1}$ and last time step's prediction $Y_{t-1}$.

$$Y_t = f(h_{t-1}, Y_{t-1})$$

In our very first Seq2Seq model, where we only used 1-D hidden vector, all the predictions converge to a straight line around mean of training data in just a few steps. Because a RNN with 1-D hidden vector only has very few parameters thus there is no way it could capture the complex patterns. In that case, the optimal strategy RNN learned is to push all inputs to the mean, that's why all the predictions converge to the mean in a few steps. And to explain the generated straight line, since previous predictions have been pushed to mean, the following predictions do not even have to move.

We believe adding more parameters would help, so we tried positional encoding, however, it was not really a success. Although the positional encoding successfully predicted something other than a straight line, the prediction became replications of a fixed pattern in Figure 3.

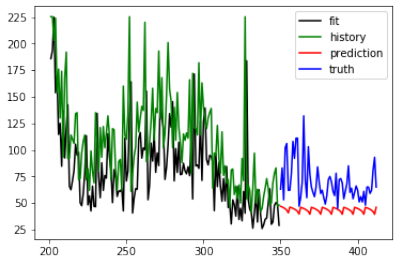


Figure 3: Example of Seq2Seq+Positional Encoding Prediction

To explain this, we add positional encoding $P$ into the formula:

$$Y_t = f(h_{t-1}, Y_{t-1}, P_t)$$

The model successfully learned that data with different positions are very different, but failed to learn the influence of $Y$ and $h$. So $Y_t$ greatly depends on $P_t$.

Let's set period as 7, assume that $Y_1$ and $Y_8$ are pretty far away, 7 steps later, $Y_8$ and $Y_{15}$ would be closer. Because much more influence of P has been brought into $Y$, the divergence between $Y_t$ and $Y_{t+7}$ would finally become very small, that's how our prediction becomes replications of a fixed pattern.

To make things worse, predictions for different pages are also similar. To explain it, let's assume we have two very different predictions for two pages at first time point $Y_1^a$ and $Y_1^b$. As the time step moves forward, $Y_2^a$ and $Y_2^b$ would be closer, compared with one step ago, because $Y_2$ mostly depends on $P_2$. And finally $Y^a$ and $Y^b$ would converge to each other.

In conclusion, when using Seq2Seq models, we suffer from two drawbacks:

- Lack of features
  Seq2Seq model was originally designed for natural language processing tasks. The words, which are always the input units of NLP tasks,contain rich information and could be treated as high-dimensional embeddings. But in our task, the data is numerical page view, and does not have rich features at each time step to use.
- Prediction error accumulation
  The lack of features results in the limited hidden vector size, which is critical for predicting a long sequence. The errors of previous predictions would be added to new predictions due to the decoder structure, so the later predictions get out of control easily.

How about the RNN+Linear Framework? The results of Phase2Phase and RNN+CNN told us it might not be an ideal alternative. These models still suffer from lack of features, essentially RNN encodes huge amount of input into a hidden vector, during the encoding process a lot of information has been filtered, this could be beneficial for high-dimensional word embeddings, but not for our 1-D daily page views or low dimensional embeddings.

#### 5.2.5 Convolutional Neural Network

We turned to look for models other than RNN to slove this problem, and chose convolutional neural network as the next step. Unlike the previous arichitecture which combines CNN with RNN, this time we used all historical data as features to increase feature dimension, and predict all next 63 days in one time to avoid accumalating prediction errors. Here CNN was used to capture the dependency of future time points on the past data.

We firstly reshape a input vector $X$ with length n into a matrix M with length $(7, \frac{n}{7})$, we chose 7 because the data has strong weekly pattern. Then send M into a one-layer neural network as showed in Figure 4, with a convolution kernel $\Pi$ and bias $b_A$, and a non-linear activation function $\sigma$.

$$A = \sigma(\Pi * M + b_A)$$

We then apply a max-pooling operation on each row of A to extract the most important features $A'$.

$$A' = \{max(A_1), max(A_2), ..., max(A_p)\}$$

After we get the features $A'$, we treat it as weights for all past data, and mutiply it elementwisely with input X.

$$Y = W(A' \otimes X) + b_W$$

The final output Y is a linear combination of weighted input vector.

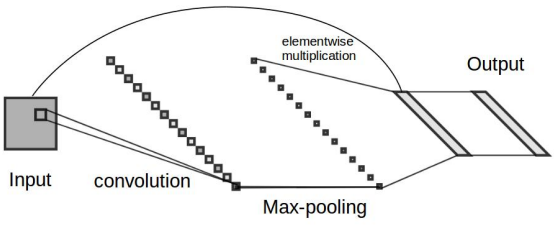


Figure 4: CNN model

# 6 Experiments

## 6.1 Data Preprocess

We start investigating the data by applying basic Kalman Filter Algorithm. And we found that the data is extremely noisy and there exists multiple extreme outliers that fundamentally destroy the patterns and relationships within the data. We removed outliers by replacing the points outside the 2 standard deviation with the mean,then replicated this operation again. A simple example will be the following:

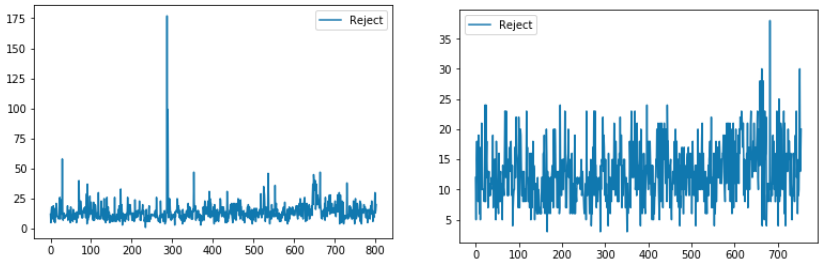


Figure 5: Left is the data with outliers, right is the data with outliers removed

We can see that from the sample time series data above that when we remove the outliers from the original data points, we are able to recover low noise and periodical pattern within the time series data. We trained Kalman Filter on both original data and the cleaned data, and used the trained model to perform inference on the test data set. The result further validated our hypothesis that by removing outliers the model will be able to better capture the inner trend and periodicity in the data. Here is the result:

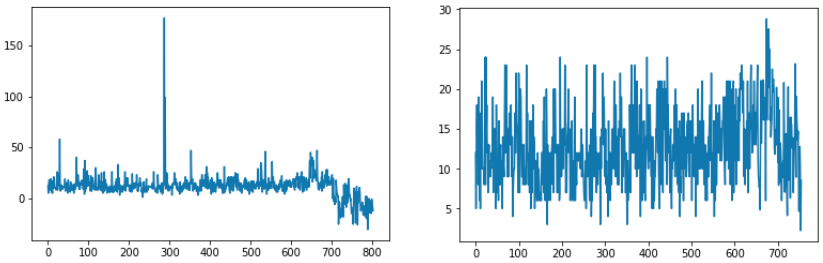

Figure 6: On the left when we use Kalman Filter trained on original data to inference the test data, we get SAMPE=163.47. On the right, we use Kalman Filter trained on the cleaned data to inference the test data, we get SAMPE=43.23

Our result shown above demonstrate that when we use original data, Kalman Filter is going to perform poorly. And when we remove the outliers, the Kalman Filter will perform much better and is able to capture the periodicity within the data.

## 6.2 Evaluation

For Kalman Filter and Gaussian process, we kept the last 63 days of traffic as our test data set and used everything before the last 63 days as our training data.

For neural network models, we used a rolling train/test split. For each data instance of 803 days, we set a training length, for example, 350 days, and predicting length as 63, then roll a window of length 350+63 over the original data instance, each window moves 63 days forward.

The evaluation metric we use in this project is Symmetric Mean Absolute Percentage Error (SMAPE).

$$SMAPE = \frac{100\%}{n} \sum_{t=1}^{n} \frac{|F_t - A_t|}{(|A_t| + |F_t|)/2}$$

where $A_t$ is the ground truth value and $F_t$ is the predicted value.

### 6.3 Results and Analysis

#### 6.3.1 Traditional models

Kalman Filter works as our baseline model for page by page analysis, we get SMAPE score= 62.45 for Kalman Filter.

During the experiments with Gaussian Process, we first test the performance of different kernels, and we found that the best kernel is $K = K_{rbf} + K_{mlp}$, here is result.

| Kernels | RBF | MLP | PeriodicExponential | RBF + MLP |
|---|---|---|---|---|
| SMAPE | 44.35 | 42.73 | 47.21 | 32.48 |

Table 2: SMAPE score for Gaussian Process with different kernels

Figure 7 is a example of how Gaussian Process performs inference.

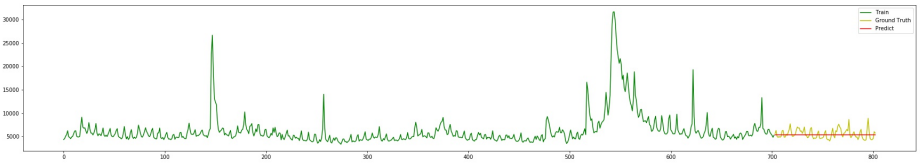


Figure 7: Gaussian Process Example

We also experiments on whether the number of training points will affect the performance of the model. We trained the same model on different training data, including the last 300 days of traffic, last 400 days of traffic, last 500 days of traffic and all traffic data. We found out that the best model is trained with the largest amount of data. Here is the result:

| Number of days | last 300 | last 400 | last 500 | all days |
|---|---|---|---|---|
| SMAPE | 38.22 | 37.10 | 35.12 | 32.48 |

Table 3: SMAPE score for Gaussian Process with different size training data

#### 6.3.2 Neural Networks

For the neural network models, as we discussed previously, the CNN model outperforms others greatly.

| Model | Best SMAPE |
|---|---|
| Linear | 52.78 |
| 1-D Seq2Seq | 67.57 |
| Seq2Seq+position encoding | 50.34 |
| Phase2Phase | 65.39 |
| RNN+CNN | 68.35 |
| CNN | 35.84 |

Table 4: Comparison of non-parametric models

As showed in Figure 8, the CNN models turned out to perform pretty well.

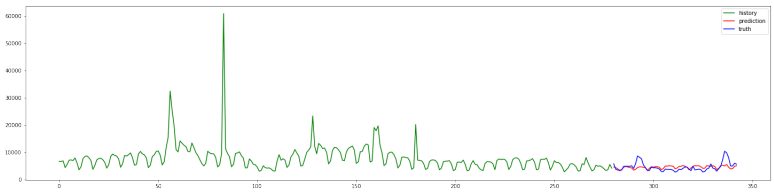


Figure 8: Example of CNN Prediction

We also observed that as we increase the number of training data, the performance gets better. This proves that there exists common patterns between different wikipedia pages' visits. Here we set training length to be 140, kernel size as $5\times$ 5, number of filters to be 150. We also tried to modify the training length and hyperparameters for CNN, but didn't observe great improvement.

| Number of training data instances | 1000 | 4000 | 10000 | all( 80000) |
|---|---|---|---|---|
| SMAPE | 42.32 | 39.90 | 37.91 | 35.84 |

Table 5: SMAPE score for CNN model with different number of training data instances

### 6.3.3 Comparison between CNN and Guassian Process

Gaussian process trains one model for each webpage, while CNN trains a single model for all webpages. From the comparison we can see that Gaussian process did a pretty good job for cleaned data, but CNN works better when dealing with missing values and high noise, since it could apply knowledge of other webpages to a noisy webpage.

| Model | Best SMAPE on cleaned data | Best SMAPE on original data |
|---|---|---|
| Gaussian Process | 32.48 | 52.36 |
| CNN | 35.84 | 42.31 |

Table 6: Comparison of CNN and Gaussian Process

## 7 Conclusion and Future Work

From the results above, we conclude that:

- Removing outliers would greatly improve performance.
- There exists pattern for each single page.
- There exists pattern across different pages.
- Focusing on each single pages' pattern could benefit the prediction of pages with smoothy history.
- Learning patterns across different pages could benefit the predition of noisy pages.

Due to time limit, we haven't fully tuned model's hyperparameters and expolred ensembling of different models. From the conclusion above, we believe combining CNN and Gaussian Process' predictions would improve our results again. So we will work on it for the next step.

## 8 Student Contribution

- Dinglun Fu: Worked on Gaussian Process, Kalman Filter.
- Yicheng Pu: Worked on Seq2Seq model, CNN model.
- Yu Sun: Worked on Phase2Phase, RNN+CNN Model .

## References


[1] R. Madan and P. SarathiMangipudi. Predicting computer network traffic: A time series forecasting approach using dwt, arima and rnn. pages 1–5, 08 2018.

[2] T. Wong and Z. Luo. Recurrent auto-encoder model for multidimensional time series representation, 2018.

[3] G. P. Zhang and M. Qi. Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research*, 160(2):501–514, 2005.

[4] P. Zhang. Zhang, g.p.: Time series forecasting using a hybrid arima and neural network model. neurocomputing 50, 159-175. *Neurocomputing*, 50:159–175, 01 2003.