Web Traffic Time Series Forecasting

DSIndy, 12/11/19

Andrew Hoblitzell

Introduction

Contest Information

Testing state-of-the-art methods designed by the participants, on the problem of forecasting future web traffic for approximately 145,000 Wikipedia articles.

Sequential or temporal observations emerge in many key real-world problems, ranging from biological data, financial markets, weather forecasting, to audio and video processing. The field of time series encapsulates many different problems, ranging from analysis and inference to classification and forecast. What can you do to help predict future views?

Evaluation

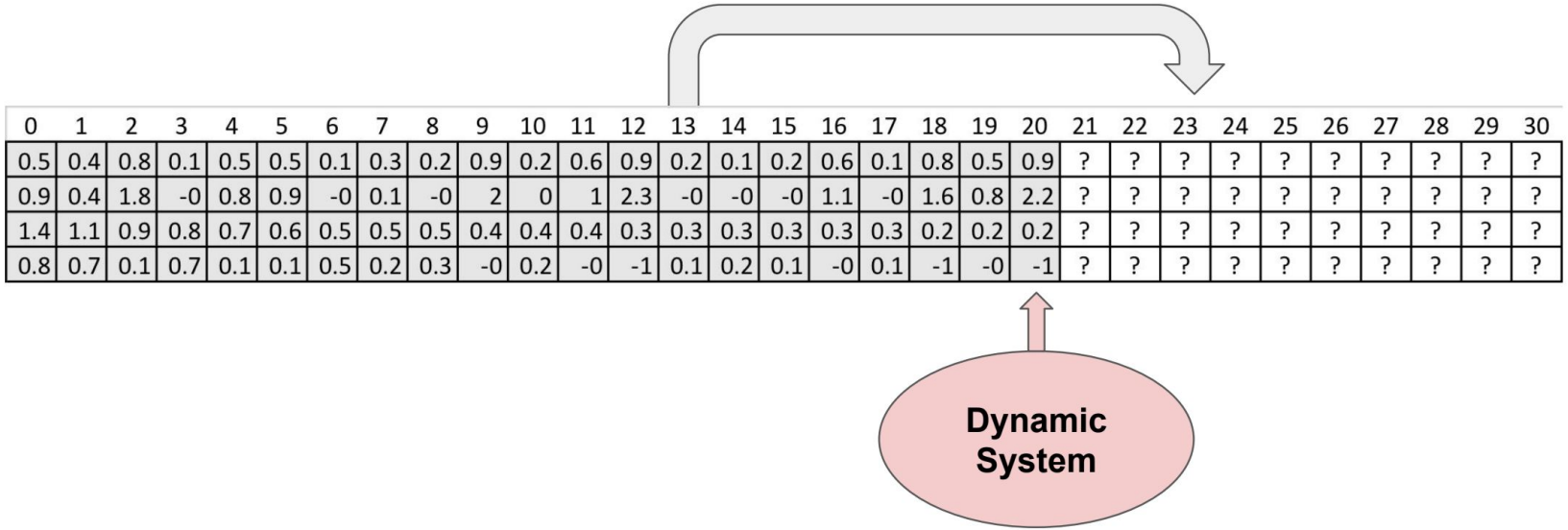
Submissions are evaluated on SMAPE between forecasts and actual values. We define SMAPE = 0 when the actual and predicted values are both 0.

Symmetric mean absolute percentage error (SMAPE or sMAPE) is an accuracy measure based on percentage (or relative) errors.

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

Overview

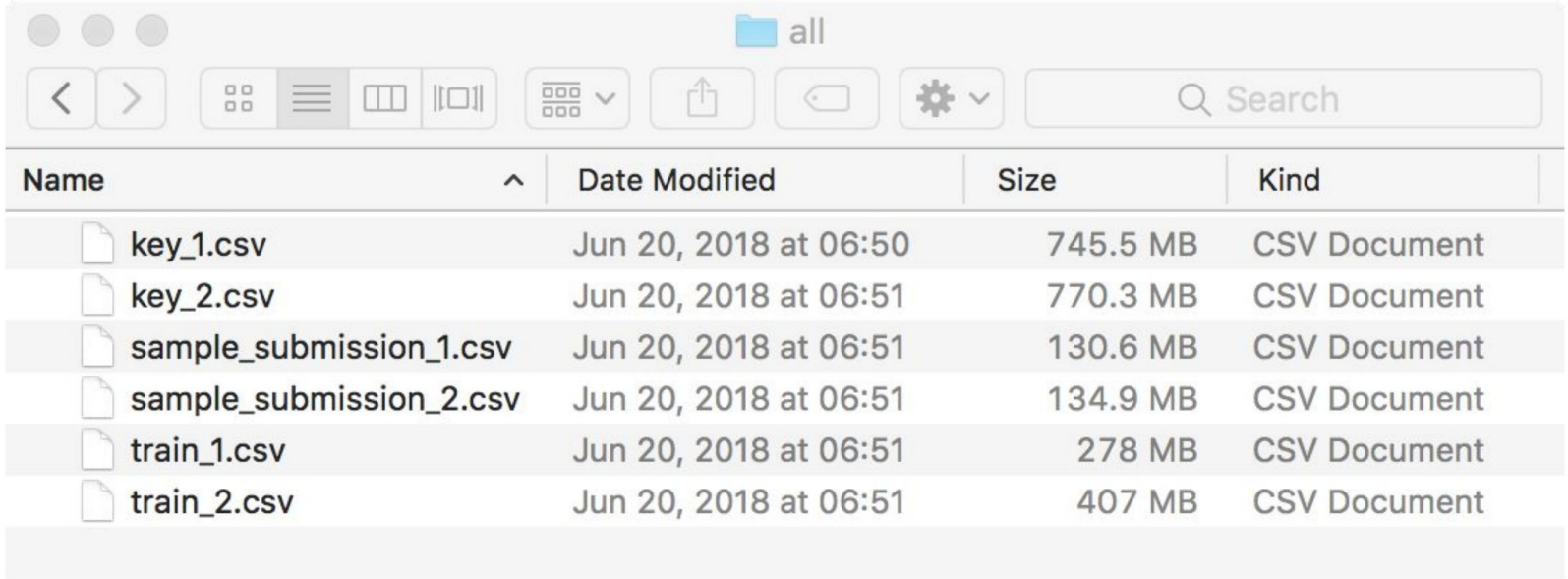
Time Series Analysis And Forecasting



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
0.5	0.4	0.8	0.1	0.5	0.5	0.1	0.3	0.2	0.9	0.2	0.6	0.9	0.2	0.1	0.2	0.6	0.1	0.8	0.5	0.9	?	?	?	?	?	?	?	?	?	?
0.9	0.4	1.8	-0	0.8	0.9	-0	0.1	-0	2	0	1	2.3	-0	-0	-0	1.1	-0	1.6	0.8	2.2	?	?	?	?	?	?	?	?	?	?
1.4	1.1	0.9	0.8	0.7	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.3	0.3	0.3	0.3	0.3	0.3	0.2	0.2	0.2	?	?	?	?	?	?	?	?	?	?
0.8	0.7	0.1	0.7	0.1	0.1	0.5	0.2	0.3	-0	0.2	-0	-1	0.1	0.2	0.1	-0	0.1	-1	-0	-1	?	?	?	?	?	?	?	?	?	?

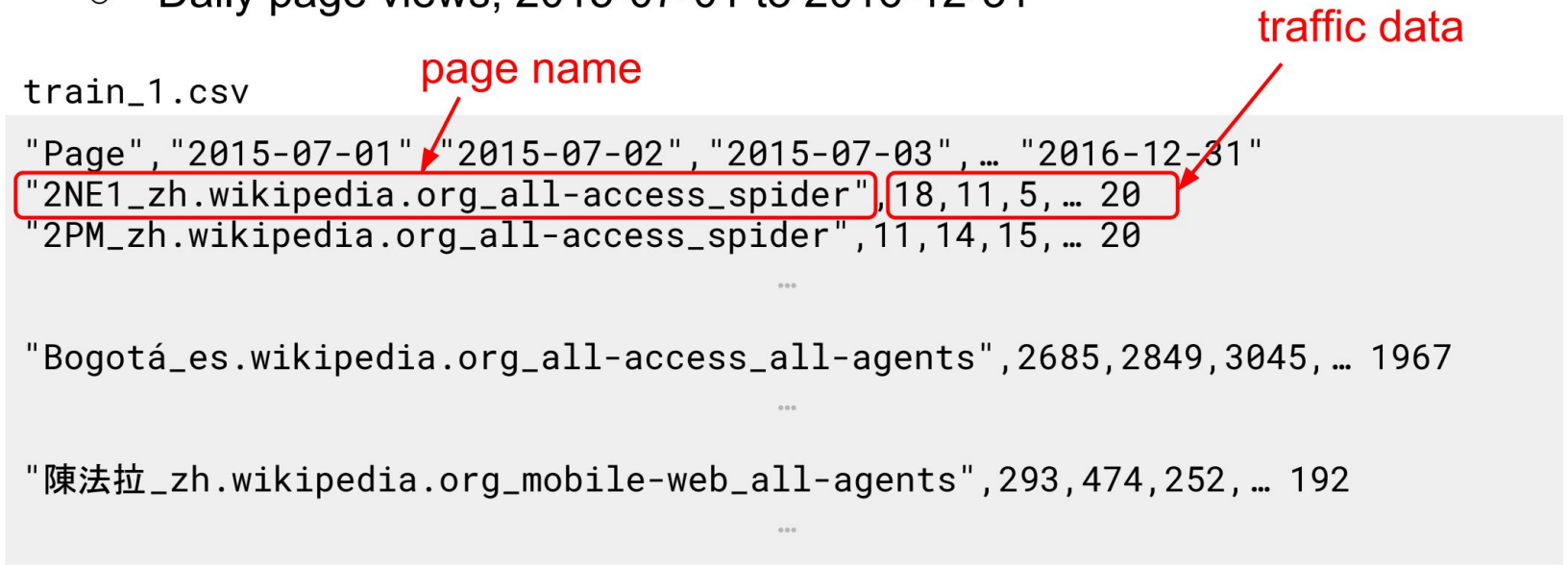
URL: https://en.wikipedia.org/wiki/Time_series

The Data



URL: https://www.kaggle.com/c/web-traffic-time-series-forecasting/data

- 145,000 time series:
  - Daily page views, 2015-07-01 to 2016-12-31



The Data

- List of keys

key_1.csv

shortened id

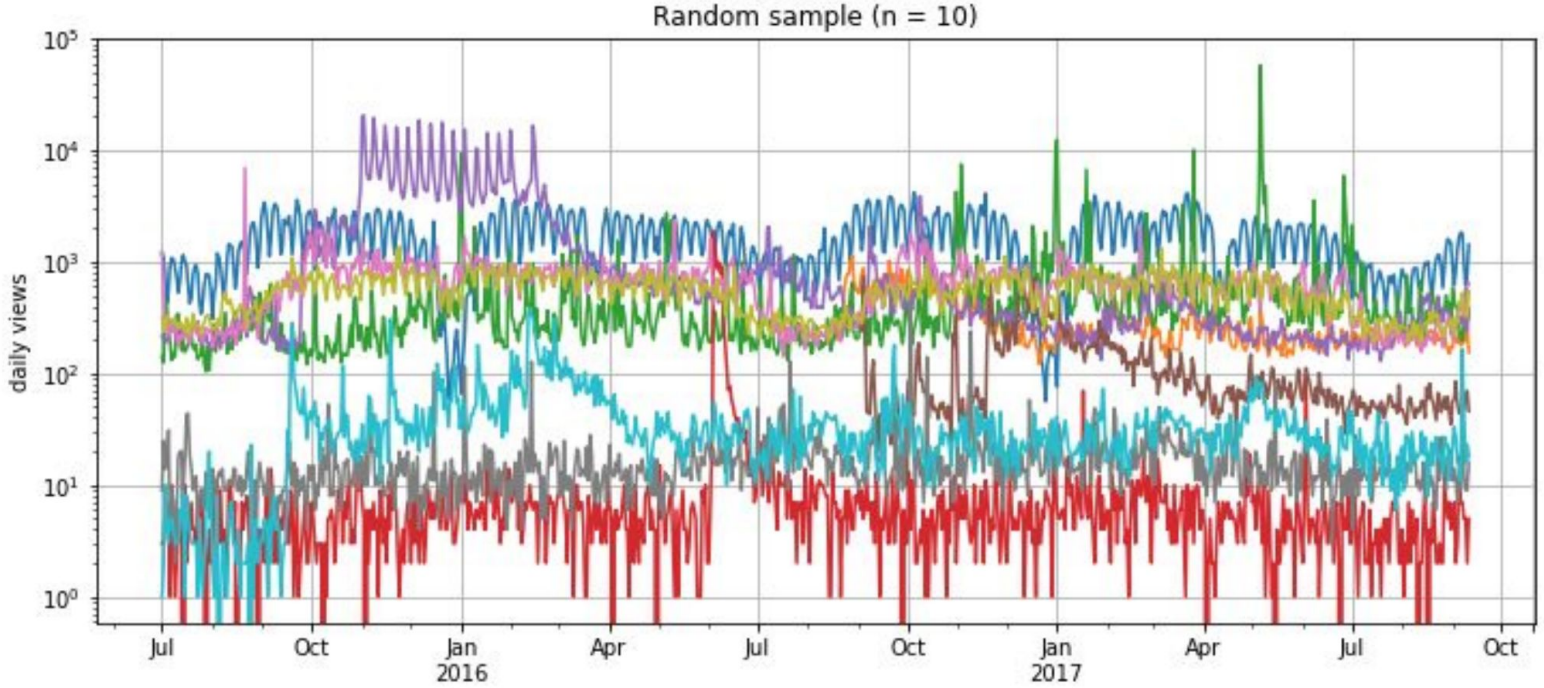
```
"Page","Id"
"!vote_en.wikipedia.org_all-access_all-agents_2017-01-01",bf4edcf969af
"!vote_en.wikipedia.org_all-access_all-agents_2017-01-02",929ed2bf52b9
...

"Bogotá_es.wikipedia.org_all-access_all-agents_2017-01-23",25e7cc352d8e
...

"陳法拉_zh.wikipedia.org_mobile-web_all-agents_2017-02-03",50fa6fe170be
...
```

- Sample submission

sample_submission_1.csv

```
Id,Visits
bf4edcf969af,0
929ed2bf52b9,0
ff29d0f51d5c,0
e98873359be6,0
fa012434263a,0
```

The Data


Random sample (n = 10)
daily views
10^5
10^4
10^3
10^2
10^1
10^0
Jul
Oct
Jan
2016
Apr
Jul
Oct
Jan
2017
Apr
Jul
Oct

Evaluation

- Metric:
  - SMAPE - Symmetric mean absolute percentage error
  - Based on percentage (or relative) errors

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

where A_t is actual value, F_t is forecast value

URL: https://en.wikipedia.org/wiki/Symmetric_mean_absolute_percentage_error

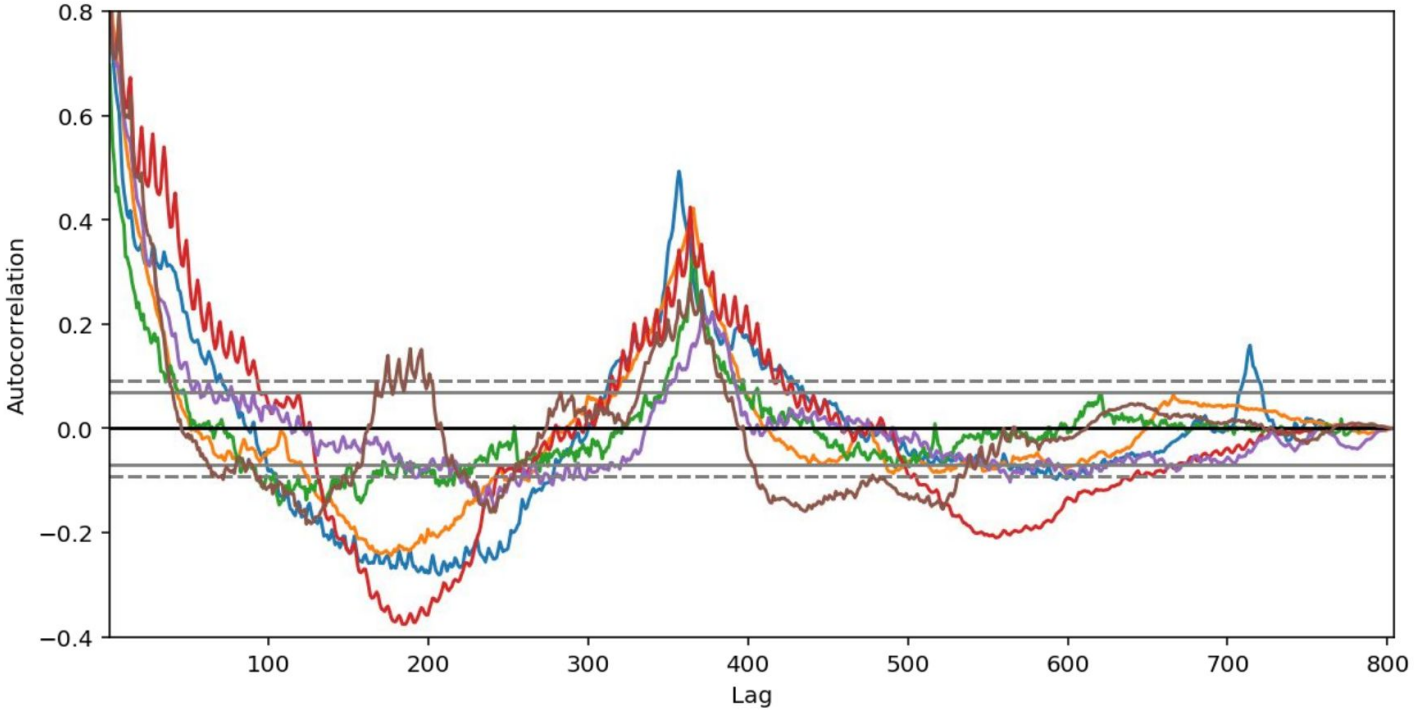
Intuition

There are two main information sources for prediction:

- Local features.
  - If we see a trend, we expect that it will continue (auto-regressive model)
  - If we see a traffic spike, it will gradually decay (moving-average model)
  - If we see more traffic on holidays, we expect to have more traffic on holidays in the future (seasonal model).
- Global features
  - If we look to autocorrelation plot, we'll notice strong year-to-year autocorrelation and some quarter-to-quarter autocorrelation.

Source: https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

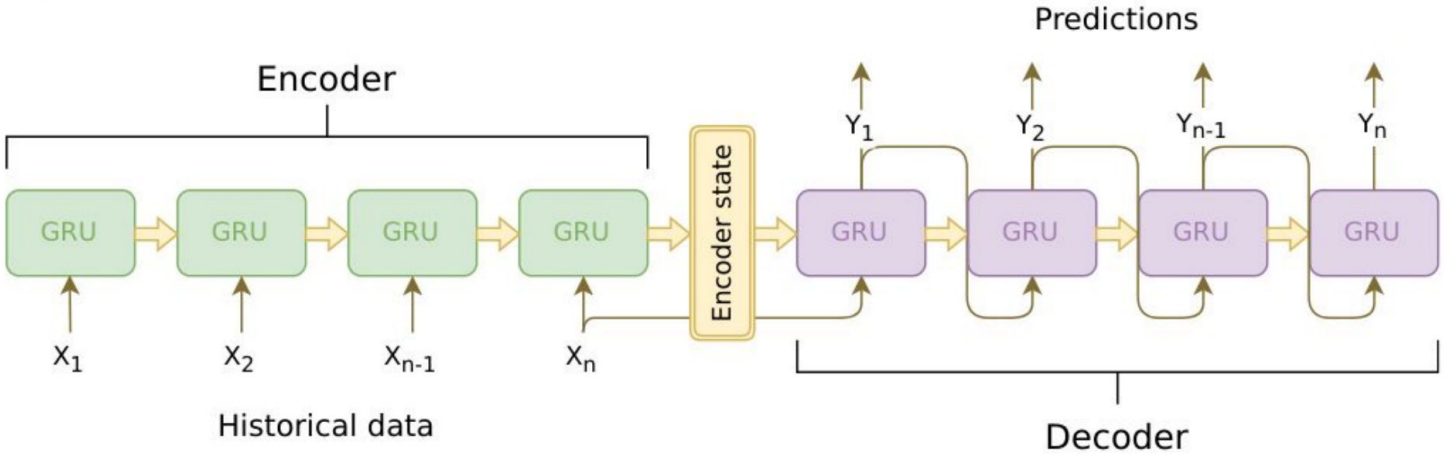
Autocorrelation Plot



Source: https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

Model

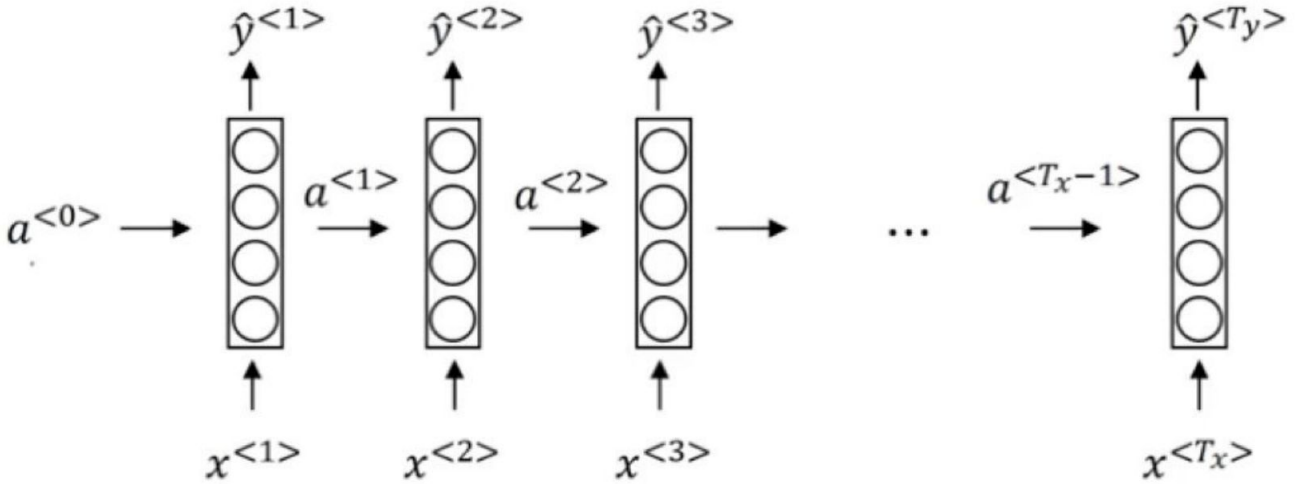
Sequence-to-sequence model



- Encoder is cuDNN GRU.
- Decoder is TF GRUBlockCell, wrapped in `tf.while_loop()` construct.

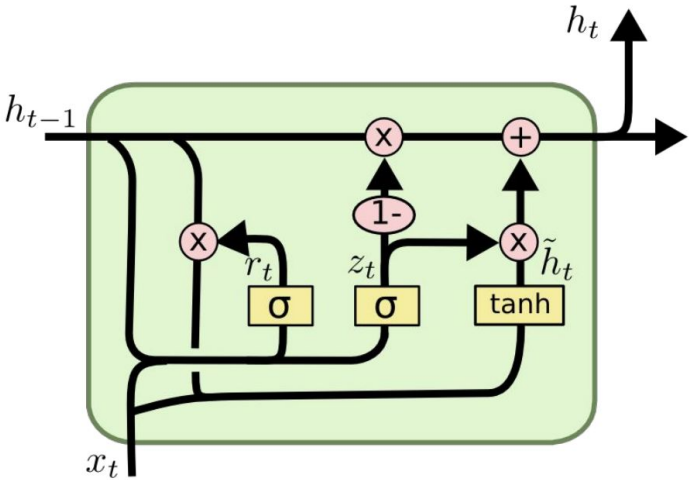
Source: https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

Recurrent Neural Networks (RNNs)



Source: https://www.coursera.org/specializations/deep-learning

Gated Recurrent Unit (GRU)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Model Design Choice

I decided to use a *seq2seq* model (RNN) for prediction, because:

- RNN can be thought of as a natural extension of well-studied ARIMA models, but much more flexible and expressive
- RNN is non-parametric, that greatly simplifies learning
- Accepts any exogenous feature (numerical or categorical, time-dependent or series-dependent) can be easily injected into the model
- seq2seq seems natural for this task: we predict next values, conditioning on joint probability of previous values, including our past predictions
- Deep Learning is all the hype nowadays.

Source: https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

Feature Engineering

Minimalistic because RNN is able to discover and learn features on its own:

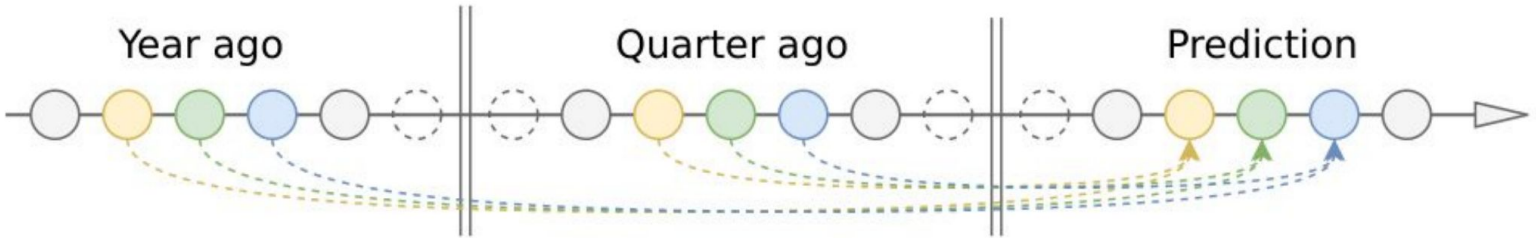
1. **pageviews** - raw values transformed by `log1p()` to get more-or-less normal intra-series values distribution, instead of skewed one.
2. **agent, country, site** - extracted from page urls and one-hot encoded
3. **day of week** - to capture weekly seasonality
4. **year-to-year, quarter-to-quarter autocorrelation** - to capture yearly and quarterly seasonality strength
5. **page popularity** (median of pageviews) - helps to capture traffic scale. High traffic and low traffic pages have different traffic change patterns.
6. **lagged pageviews** - I'll describe this feature later

Source: https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

Model

Working with long timeseries

- On sequences longer than 100-300 items, even LSTM/GRU can gradually forget the oldest items
- First method was to use some kind of attention
- E.g. Fixed-weight sliding-window attention:

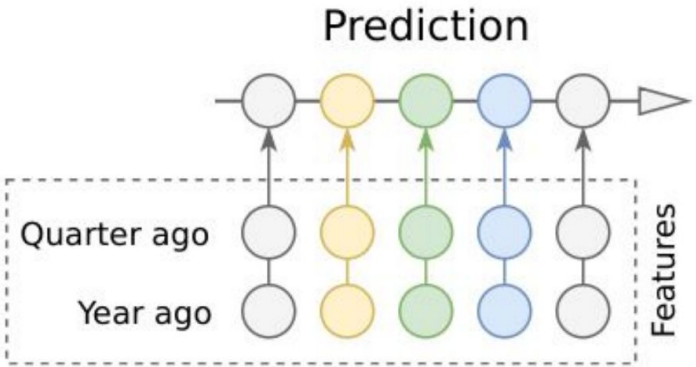


Source: https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

Model

Working with long timeseries

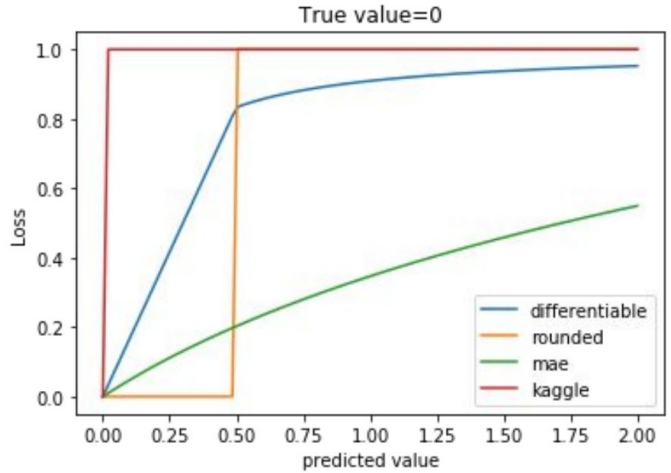
Unsatisfied by complexity of attention mechanics, I removed attention completely and just took important (year, half-year, quarter ago) data points from the past and used them as additional features for encoder and decoder.



Source: https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

Training

- SMAPE can't be used directly, because of unstable behavior near zero values (loss is a step function if truth value is zero)
- Used a smoothed differentiable SMAPE variant, which is well-behaved at all real numbers:




```
epsilon = 0.1
summ = tf.maximum(tf.abs(true) + tf.abs(predicted) + epsilon, 0.5 + epsilon)
smape = tf.abs(predicted - true) / summ * 2.0
```

Source: https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

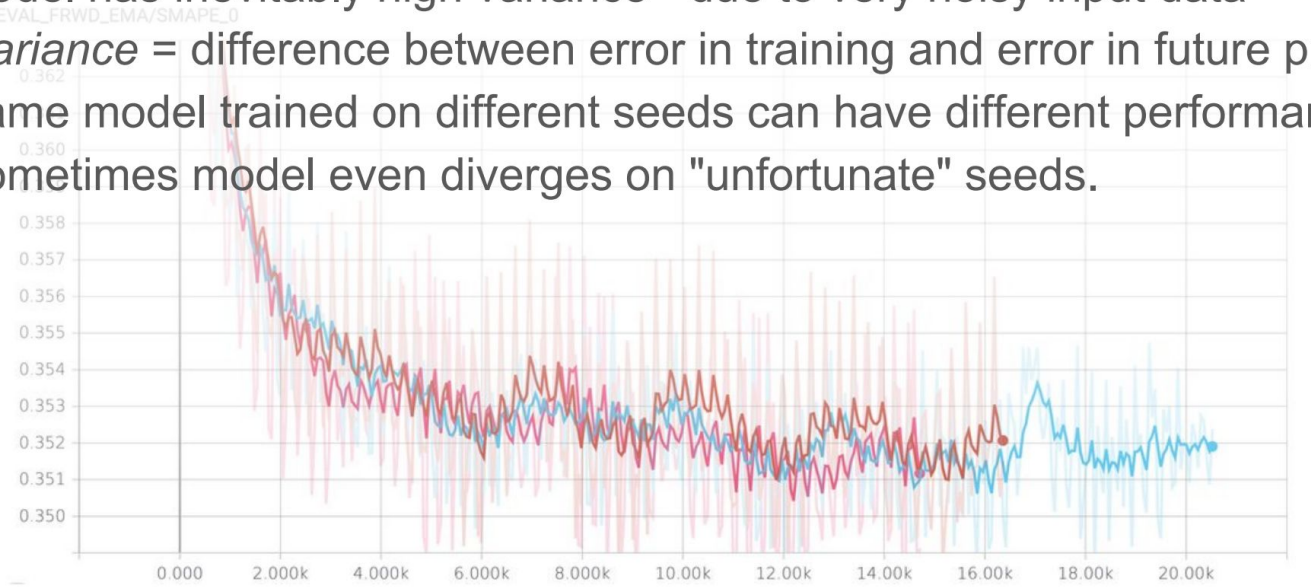
Training

- Model trains on random fixed-length samples from original time series.
- Training code randomly chooses starting point for each time series on each step, generating endless stream of almost non-repeating data.
- This sampling is effectively a data augmentation mechanism
- Used COCOB optimizer for training, in combination with gradient clipping
- COCOB tries to predict optimal learning rate for every training step, so you don't have to tune learning rate at all*
- It also converges considerably faster than traditional momentum-based optimizers, especially on first epochs, allowing me to stop unsuccessful experiments early.

* See paper Training Deep Networks without Learning Rates Through Coin Betting.
Source: https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

Issues

- Model has inevitably high variance - due to very noisy input data
  (*variance* = difference between error in training and error in future prediction)
- Same model trained on different seeds can have different performance
- Sometimes model even diverges on "unfortunate" seeds.



Source: https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

Reducing Model Variance

1. Chose a region when model is trained well enough, but has not started to overfit: 10,500 and 11,500 training steps then saved checkpoints every 100 steps in this region
2. Trained 3 models on different seeds and saved checkpoints from each model. Took average predictions from all 30 (10x3) models
3. Used SGD averaging (ASGD) - maintain moving averages of network weights during training and use these instead of original ones, during inference

- Combination of the three methods worked well
- Got roughly the same SMAPE error on leaderboard (for future data) as for validation on historical data.

Source: https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

Hyper-parameter Tuning

- There are many model parameters (number of layers, layer depths, activation functions, dropout coefficients, etc) that can be tuned to achieve optimal performance.
- Used the SMAC3 package to automate hyperparameter search.
- Contrary to my expectations, hyper-parameter search did not find well-defined global minima
- All best models had roughly the same performance, but different parameters.
- Probably RNN model is too expressive for this task, and best model score depends more on the signal-to-noise ratio than on the model architecture.

Source: https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

What I Learned

- Explore the data
- Look at performance of traditional methods
- Use intuition to find features that could be useful
- Choose the right cost function - matched to evaluation metric + optimizer
- The rest is luck or magic!
- ...

Also: Trying to run someone else's code on another person's machine is a nightmare!

2nd Place Submission

Based on 5 ideas:

1. Use the yearly seasonality of the data - it is huge
2. Don’t use RMSE. Approximate SMAPE with log1p transformed data and use custom objective functions for each optimizer
3. Get rid of outliers
4. Ensemble everything in xgboost by training it on the residuals of the Keras predictions and the same features as my XGBoost model plus out of fold predictions from Huber regressor and Keras model
5. Use medians as features instead of raw values.

Source: https://www.kaggle.com/c/web-traffic-time-series-forecasting/discussion/39395

2nd Place Submission

- Feedforward network: [200, 200, 100, 200] units in each layer
- Input is concatenated again with the output of the first layer (I don't know why but this boosted accuracy)
- Activation is relu except for last one which is linear
- Used dropout of 0.5 for almost all layers (0.5 was selected by CV)
- Used a batch normalization for the middle layer
- Model is compiled with adam optimizer and the loss function defined above.
- I tried CNNs and RNNs (LSTM, and Seq2Seq) but did not get results as good as the simple feed-forward network.

Source:
https://www.ibm.com/developerworks/community/blogs/jfp/entry/2nd_Prize_WInning_Solution_to_Web_Traffic_Forecasting_competition_on_Kaggle?lang=en
https://github.com/jfpuget/Kaggle/tree/master/WebTrafficPrediction

Other Submissions

3rd: NOT a fancy RNN, but it is not, is the not so glanorous kNN.

Also, it only has 2 key components:

1. nn = NearestNeighbors(n_neighbors=k, metric='canberra')
2. y_pred is the median of the neighbors

Split the training data by traffic type.

https://www.kaggle.com/c/web-traffic-time-series-forecasting/discussion/39876

6th: Seq2Seq w/CNN for Speed
https://www.kaggle.com/c/web-traffic-time-series-forecasting/discussion/39370

8th Place: Kalman Filter

Resources

Winning implementation https://github.com/Arturus/kaggle-web-traffic/

Slides mainly from https://www.slideshare.net/BillTubbs/web-traffic-time-series-forecasting?from_action=save

Links to several implementations https://www.kaggle.com/sudalairajkumar/winning-solutions-of-kaggle-competitions