Modeling approaches for time series forecasting and anomaly detection

Du, Shuyang
shuyangd@stanford.edu

Pandey, Madhulima
mpandey8@stanford.edu

Xing, Cuiqun
carriex@stanford.edu

Abstract

Accurate time series forecasting is critical for business operations for optimal resource allocation, budget planning, anomaly detection and tasks such as predicting customer growth, or understanding stock market trends. This project focuses on applying machine learning techniques for forecasting on time series data. The dataset chosen is web traffic time series for Wikipedia webpages. We explore three different approaches including K-Nearest Neighbors (KNN), LSTM-based recurrent networks (LSTM), and Sequence to Sequence with Causal CNN (SeqtoSeq CNN). These approaches will be verified through error analyses using SMAPE and time series plots.

1. Introduction

With the rapid rise of real time data sources, prediction of future trend and the detection of anomalies is becoming increasingly important. This project focuses on prediction of time series data for Wikipedia page accesses for a period of over twenty-four months [1]. We use a number of different machine learning methods for the web traffic prediction. Unexpected deviations in web-traffic or anomalies can be an indication of an underlying issue. Forecasting with such deviation is non-trivial because streams are variable and noisy with outlier spikes. This problem requires methods that work on temporal sequences rather than single points. The methods explored here are K-nearest neighbors (KNN), Long short-term memory network (LSTM), and Sequence to Sequence with Convolution Neural Network (CNN) and we will compare predicted values to actual web traffic. The predictions can help us in anomaly detection in the series.

2. Related Work

RNN based networks (based on LSTM or GRU units) have become popular for time-series analysis, where they encode the past information as a fixed-length vector and use the decoder to generate a prediction. Their performance deteriorates rapidly as the length of input sequence increases [4], and this is a limitation in time series analysis, as it could reduce predictions based on long segment of the target series. [2] uses an attention mechanism to select parts of hidden states across all the time steps, similar to [15] described below. [3] develop a GRU-based deep learning model that exploits information missing data in multivariate time series data to capture long-term temporal dependencies of time series observations and improve the prediction results such as medical outcome. This objective differs from our work of future forecasting of time-series data, however GRU-based recurrent networks are included as future work we intend to evaluate. Karim [7] discusses augmenting a fully-convolutional network (FCN) with LSTM to improve time series classification performance. Qin's work [15] comes closest to ours. They describe an attention-based recurrent neural network which consists of an encoder with an input attention mechanism and a decoder with a temporal attention mechanism which the authors claim captures long-range temporal information of the encoded inputs by adaptively selecting the most relevant input features and use for stock market prediction.

Sequence to sequence learning has been successful in tasks like time series modeling and machine translation. The dominant approach to date is with recurrent neural network based encoder-decoder architectures [10] [2]. Convolutional neural networks are less common for sequence modeling, despite several advantages [19] [9]. Compared to recurrent layers, convolutions create representations for fixed size contexts, however, the effective context size of the network can easily be made larger by stacking several layers on top of each other. This allows to precisely control the maximum length of dependencies to be modeled. Convolutional networks do not depend on the computations of the previous time step and therefore allow parallelization over every element in a sequence. This contrasts with RNNs which maintain a hidden state of the entire past that prevents parallel computation within a sequence. This project is also inspired by the seq2seq CNN machine translation [5] and the causal CNN WaveNet [5] [13].

3. Dataset and Features

The dataset consists of approximately 145k time series[1]. Each of these time series represent a number of daily views of a different Wikipedia article, starting from July, 1st, 2015 up until September 10th, 2017. We split the dataset into development (training and validation) and test set by July 9th, 2017.

For each time series, we have the name of the article as well as the type of traffic that this time series represent (all, mobile, desktop, spider). The page names contain the Wikipedia project (e.g. en.wikipedia.org), type of access (e.g. desktop) and type of agent

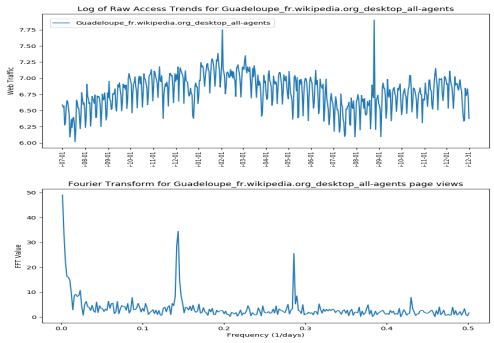

Figure 1. $Log_e(x+1)$ Normalization of data and its FFT

(e.g. spider). In other words, each article name has the following format: 'name_project_access_agent' (e.g. 'AKB48_zh.wikipedia.org_all-access_spider'). This is used as conditional features for each web traffic time series in addition to the time series values.

The data source for this dataset does not distinguish between traffic values of zero and missing values. A missing value may mean the traffic was zero or that the data is not available for that day. This introduces the extra difficulties in forecasting. In this project we treat all missing values as NaN and encode that as an extra features.

3.1. Data Processing

For different modeling approach we also apply different preprocessing steps. For seq2seq CNN approach, we take the logarithm of the time series values and normalize it to have zero mean. For conditional features for each time series, in addition to the features mentioned above (Wikipedia project, type of access and type of agent), we also use binary variable for null value indication and log mean value of each time series as additional encoding features and decoding steps as the decoding features. Using decoding steps as one feature can help the model know where the current step is and thus use this as a positional information during prediction.

For the LSTM approach, we follow the process described ahead. The are many series in which values are zero. This could be a missing value, or actual lack of web page access. In addition, there are significant spikes in the data, where values have a broad range from 1 to hundreds/thousands for several web pages. We normalize this data by adding 1 to all entries, taking the log of the values, and setting the mean to zero and variance to one. We have the results of fourier analysis for exploring periodicty on a weekly/monthly/quarterly basis. In Figure 1 the top graph shows the normlized time-series dataset. The second graph in Fig 1 clearly indicates periodicity of 3 and 7 days. In general, not all time-series dataset have an evident periodicity.

3.2. Feature Extraction

After data preprocessing we extract a number of the features from the data including the log(hit_count + 1), day of the week encoded as one-hot with zero mean and unit variance, monthly auto-correlation, quarterly auto-correlation, annual auto-correlation, traffic medians, country code encoded as one-hot with zero mean and unit variance (7 countries), and access type encoded as one-hot with zero mean and unit variance (4 access types).

4. Methods

We have described below three separate approaches for time-series forecasting in our project, KNN, Seq-to-Seq CNN, and LSTM. The KNN-based approach is our baseline method for prediction.

4.1. General Machine Learning-based Approach

4.1.1 KNN (Baseline)

K-nearest neighbor algorithms were commonly used for classification problems but have since been extended for time series regression and anomaly detection as well [17]. Some previous application of KNN regression have been forecasting traffic flow [20] and predict rice prices [18]. This approach is implemented by calculating the weighted average of the k-nearest neighbors, weighted using the inverse of their distance.

In order to calculate the distance between neighbors there are distance calculators including popular ones such as Euclidean, Hamming, Manhattan and Monkowski; this is all dependent on what type of dataset is used. The algorithm starts by calculating the distance between the query instance and all training samples. The distances are sorted and we determine the k-nearest neighbors using an optimization such as RMSE (root-mean-square deviation). The average of the nearest neighbors are calculated and we use this to predict the value of the query instance. This is further extended to anomaly detection where the distance between the query instance and the k-the nearest neighbor is a local density estimate and the larger the distance, the more likely the query is an outlier [16]. We will use KNN as a baseline to compare the effectiveness of other approaches

4.2. Sequence to Sequence with CNN

In this approach, we combine the sequence to sequence model with causal convolutions. Following is the graph for a typical model. Based on the input time series, the model will learn a set of parameters for encoding convolutional layers. Then for the prediction period, the model will learn another set of parameters for decoding convolutional layers based on inputs, previous encoding hidden units, previous decoding hidden units and predictions. We will describe the methodology of each component of this model in details in the following sections.

4.2.1 Sequence to sequence model

Sequence to sequence learning has been successful in many tasks. The dominant approach to date encodes the input sequence with a series of RNNs and generates a variable

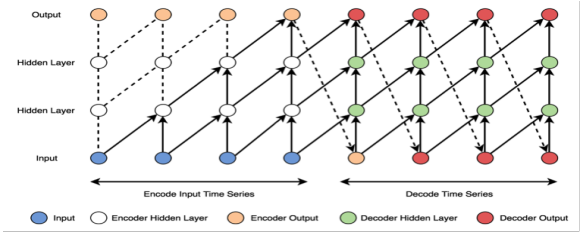

Figure 2. Sequence to Sequence Model.

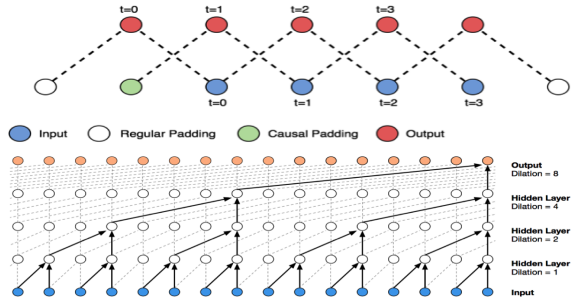

Figure 3. Dilated 1-D causal convolution network, and convolutions for dilations 1,2, 4 and 8.

length output with another set of decoder RNNs. We can easily adopt the seq2seq approach for time series prediction. The input would be series up till now and the decoding will be triggered by the first prediction after the latest point we know.

4.2.2 Dilated 1-D causal convolutions

1D convolutions can be used to model a series of data compared to traditional 2D convolutions used for image recognition. With a careful padding for the input and corresponding shift for the output, we can make sure the prediction emitted by the model at timestep t will not depend on any of the future timesteps. This is so called causal convolutions.

The graph in figure 3 is a simple example for causal padding. The 1D convolution has dilation one. Before the regular convolution, we have one extra causal padding and we only select the first 4 elements of the output (serves as the shift). It's obvious that without this causal padding, the output will end up with depending on future inputs.

A dilated convolution is a convolution where the filter is applied over an area larger than its length by skipping input values with a certain step. A dilated convolution effectively allows the network to operate on a coarser scale than with a normal convolution. Stacked dilated convolutions enable networks to have very large receptive fields with just a few layers, while preserving the input resolution throughout the network as well as computational efficiency. Figure 3 depicts dilated causal convolutions for dilations 1, 2, 4, and 8.

Replacing the RNN structure with the CNN structure described above gives us this new model framework. Compared to recurrent layers, convolutions create representations for fixed size contexts, however, the effective context size of the network can easily be made larger by stacking several layers on top of each other. This allows to precisely control the maximum length of dependencies to be modeled.

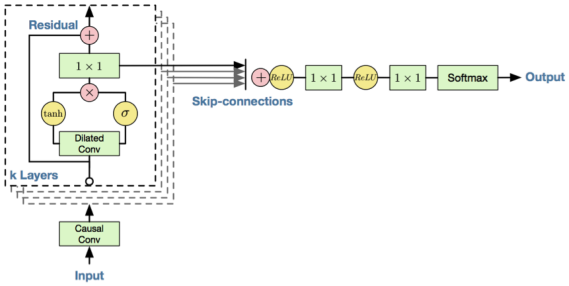

Figure 4. Figure 7: Residual block used in Seq to Seq CNN

4.2.3 Gated activation units and Residual, skip connections

We use the same gated activation unit defined as following.

$$z = tanh(W_{f,k} * x) \circ \sigma(W_{g,k} * x)$$

Using this type of activation makes us end up with 2 times number of parameters for convolution filters.

Both residual and parameterised skip connections are used throughout the network, to speed up convergence and enable training of much deeper models. To make the dimension compatible between different layers for residual and skip connections, we introduced a dense layer within each convolution block. Figure 7 in the appendix shows the residual block described above.

4.3. RNN with LSTM

A recurrent neural network (RNN) architecture is well suited for learning from the sequence of values in a time-series dataset, to solve the problem of predicting the output in the next step based on history of previous inputs. Given that the data input sequence has some form of periodicty and predictability, one can classify deep sequences with long intervals between important events. Since we will be dealing with data with monthly, quarterly or annual seasonality, we need to networks that can handle several hundred step deep history. We have used the Long Short-Term memory (LSTM) neural cell in our RNN implementation as they are relatively immune to problems of vanishing or exploding gradients even with 1000 discrete time steps [6]

Previous research[11] in time series prediction uses LSTM to capture nonlinear traffic dynamics, and has demonstrateed the capability for time series prediction with long temporal dependency. The dataset, in addition, has missing values (recorded as 0), and variants of this approach have been used to handle missing values in multivariate time series data [3].

We have implemented several LSTM configurations using TensorFlow. Our base model (Fig 5) consists of a 10

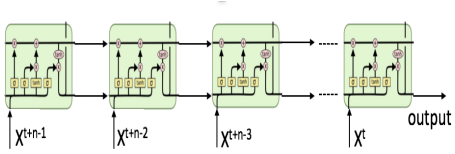

Figure 5. LSTM Recurrent Network

layer deep LSTM, with a hidden layer size of 80. In addition, we will evaluate the effectiveness of depth with a 5 and 15 layer deep LSTM. As our future work, we plan to explore additional RNN topologies such as skip connection between layers as in the style of Densenets [14], where each layer to every other layer in a feed-forward fashion. This strengthens feature propagation and reduce the number of parameters.

5. Experiment/Results/Discussion

5.1. Evaluation Metrics

We use Symmetric Mean Absolute Percentage Error (SMAPE) as our major evaluation metric. SMAPE is defined as

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

This metric is more robust towards outliers and it has a unified scale across different time series with different scale.

5.2. Hyperparameters

Since the training for deep learning models is time consuming, we use forward out-of-sample test instead of cross validation to do the hyperparameter optimization. We use the last 64 days in our training data set as the validation set and select the set of hyperparameters based on SMAPE on the validation set. The specific details of the different models are described below.

For KNN we modeled using a batch size of 4096 and tried various distances including Euclidean, Minkowski, Manhattan and Canberra. Based on the validation set the most effective distance was Canberra, which was suited for the positive and wide variance data. We also tried various k-values and k = 10 showed to be the value with the lowest validation error. Due to limitations of computing power a larger training set was not used, but may have led to a lower validation error. Cross-validation was also not used here and could have chosen a more precise k-value.

In SeqtoSeq CNN, the length of the output sequence is fixed to be 64 which is consistent with the length for our final test period. We use batch size of 128 and initial learning rate of 0.001. We also use the early stop here. We will only train the model for another 3000 steps and terminate the training if the validation error doesnt decrease.

For the neural network structure, we stacked 8 causal CNN layers with increasing dilation and width. As shown by the causal CNN figure above, the width and dilation for 1 to 8 layer would be 1, 2, 4, 8, 16, 32, 64 and 128. So the final reception field would be 128, which means the length of input sequence is 128. In the CNN, we use the padding to make each layer the same size. We use 32 as the number of neurons in fully connected layer for residual and skip connection. The decoding layer has the same structure without sharing any parameters from encoding layer.

For LSTM we have implemented several configurations. The parameters used for the experiments include back-propagation length and state size. We have used back propagation lengths between 5-60 and state size value from 10-60 (LSTM have both cell state and hidden state i.e., total state = 2*state size value). We found based on the validation set that back-propagation lengths between 30-36 and state size between 45-50 worked best on the dataset. Surprisingly longer back-propagation lengths or very large state sizes generally resulted in worse predictions. We also experimented with multiple learning rates and dropouts. The learning rate of 1.0 initially and decaying to 0.2 at the end of the training yielded the best results. A dropout value of 0.4 gave us the best validation set predictions.

5.3. Experiments

Experiments for KNN were run on a local machine using R. Since KNN can be computationally extensive with the initial storage of training data, we broke up the experiment in different batch sizes to compute the predicted values and decided to use a batch size of 4096 for optimal performance.

LSTM experiments were run using TensorFlow 1.4, python 3.6, and Ubuntu 17.10 running on a server with a i7-4790K cpu, 32GB ram and GTX-970 GPU. Since this problem was computationally intensive we tried several experiments with different batch sizes and LSTM cell types to determine ways to optimize the runtime and we have described these experiments ahead. In our efforts to optimize performance, we tried both the tf.contrib.cudnn_rnn.CudnnLSTM and rnn.BasicLSTMCell cells, however we did not see significant improvement in runtimes of CudnnLSTM over BasicLSTM.

Batch size	Epoch run time
64	6.1s
512	24.0s
1024	47.2s

The table above includes a few batch size vs epoch run-time results. For small batch sizes below 128, the overhead led to sub-linear performance. For batch-sizes greater than 2048, again the performance did not scale. We chose a batch size of 1024 for optimal performance. Even though the GPU was an older model and contained 4GB RAM, using only the CPU increased the runtime by a factor of 2.5. On an average, the runtime was 24hrs for the training set.

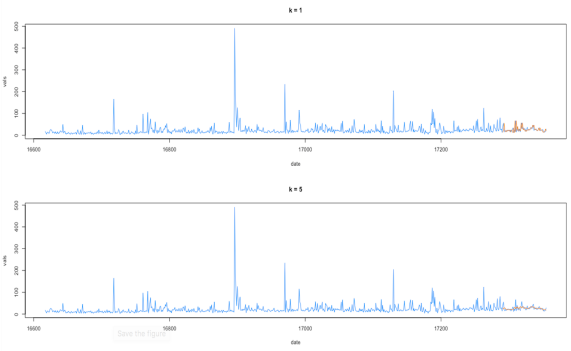

Figure 6. KNN based Prediction

Figure 7. LSTM based Prediction Results

5.4. Results

Here is the table for overall performance of the approaches we have used.

Metric	KNN	LSTM	SeqtoSeq CNN
SMAPE	165.51	143.12	42.37

Following are the time series plots for actual and predictions generated from the three models. Figure 6 shows the results for KNN based prediction. Figure 7 shows the results for LSTM based predictions and Figure 8 shows the plot for Seq to Seq CNN based predictions.

From the graph we can see the model could do a good job if the time series is stable or has certain seasonality.

We saw that model had a hard time predicting extreme values or spikes. Predicting outliers in time series is not feasible as these outliers are caused by external shock, which is not captured in the time series itself or property features. Laptev's work for extreme event forecasting [8], has used external features to predict extreme values.

6. Conclusion and Future Work

The seq2seq CNN approach has the best performance while KNN and LSTM have similar performance. In terms of improvement, KNN would have benefited from using cross-validation. Also, we have not fine tuned the parameters for deep learning models due to limited computation resources (especially LSTM) so there should be room for improvement for LSTM and seq2seq CNN approaches. With the conventional many-to-one LSTM topology, our results indicate that it is unable to adequately capture the temporal behavior of the time series. This is because at each step only one value of the output is compared to the ground truth or the expected value. In a sequence to sequence LSTM topology there is an opportunity to capture data sequentiality better by comparing multiple outputs of the network with multiple expected data values. We verified this with a small sequence to sequence network working on synthetic time-series dataset containing mixed multi-frequency sinusoidal value with ramp and noise.

Figure 8. Seq to Seq CNN based Prediction Results

Our approaches to time series prediction depends on features extracted from the the time series data itself. Our models learn periodicity, ramp and other regular trends quite well. However, none of our models are able to capture spikes or outliers that arise from external sources. Enhancing the performance of the models will require augmenting our feature set from other sources such as news events and weather.

In the future, we also plan to explore additional RNN topologies such as Seq to Seq LSTM as well as skip connection between layers as in the style of Densenets [14],where each layer to every other layer in a feed-forward fashion. Additionally, we will look at approaches to augment features from external sources. We will also look into methods to ensemble these different models for a general approach that can handle time series data from other publicly available time series such as [12].

7. Contributions

This project uses three different machine learning approaches. We each tackled one of these approaches: seq2seq with CNN (Du, Shuyang), LSTM (Pandey, Madhulima), KNN (Xing, Cuiqun)

References

[1] Web traffic time series forecasting- forecast future traffic to wikipedia pages.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu. Recurrent neural networks for multivariate time series with missing values. *arXiv preprint arXiv:1606.01865*, 2016.

[4] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[5] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

[6] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 2017.

[7] F. Karim, S. Majumdar, H. Darabi, and S. Chen. Lstm fully convolutional networks for time series classification. *arXiv preprint arXiv:1709.05206*, 2017.

[8] N. Laptev, J. Yosinski, L. E. Li, and S. Smyl. Time-series extreme event forecasting with neural networks at uber. In *Int. Conf. on Machine Learning*, 2017.

[9] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[10] M.-T. Luong, I. Sutskever, Q. V. Le, O. Vinyals, and W. Zaremba. Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*, 2014.

[11] X. Ma, Z. Tao, Y. Wang, H. Yu, and Y. Wang. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, 54:187–197, 2015.

[12] Numenta.

[13] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[14] A. Prakash, S. A. Hasan, K. Lee, V. Datla, A. Qadir, J. Liu, and O. Farri. Neural paraphrase generation with stacked residual lstm networks. *arXiv preprint arXiv:1610.03098*, 2016.

[15] Y. Qin, D. Song, H. Cheng, W. Cheng, G. Jiang, and G. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*, 2017.

[16] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *ACM Sigmod Record*, volume 29, pages 427–438. ACM, 2000.

[17] A. Sasu. K-nearest neighbor algorithm for univariate time series prediction. *Bulletin of the Transilvania University of Brasov Vol*, 5(54), 2012.

[18] D. Sinta, H. Wijayanto, and B. Sartono. Ensemble k-nearest neighbors method to predict rice price in indonesia. *Applied Mathematical Sciences*, 8(160):7993–8005, 2014.

[19] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, 37(3):328–339, 1989.

[20] L. Zhang, Q. Liu, W. Yang, N. Wei, and D. Dong. An improved k-nearest neighbor model for short-term traffic flow prediction. *Procedia-Social and Behavioral Sciences*, 96:653–662, 2013.