Web service workload prediction using deep learning - Report 3

Stefan Sebastian

April 21, 2020

1 State of the art

TODO

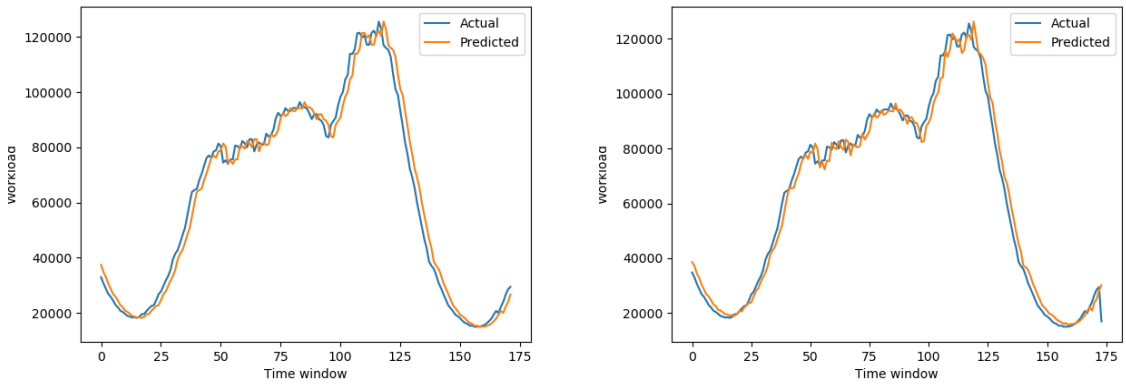
2 Approach

The main goal is to find a performant model for web application workload prediction, which can be later used by a proactive microservice scaler. The methods used are different architectures of deep learning models: MLP, CNN, CNN-LSTM hybrid. The main contribution of this research is the application of deep learning to this specific problem and the comparison with a classic timeseries approach (ARIMA).

The problem design has been influenced by the goal of integrating this model into a proactive microservice scaler. First of all, the choice of the workload measure is number of requests. The idea is that the scaling prediction should not influence the predicted value, as would be the case with CPU or memory usage. Also this is in line with research done by Jindal et al [5] who propose a metric for measuring microservice performance based on number of satisfied requests. Another consideration is the prediction interval. Taking into account the experience of Netflix [3], who run a microservice architecture in production, the time window should be in the order of minutes, so you can predict spikes and have time to deploy new service instances.

A realistic workload has been used for this experiment, a wikipedia trace for 12 days in september 2007. From this a subset of requests was extracted (all requests for Japanese wikipedia). The subset was selected in order to compare results with Kim et al. [6] which used the same dataset. In order to turn a web request log file into a supervised dataset the following steps were taken: create buckets which contain the number of requests in a time interval, iterate over the buckets using the sliding window technique [1]. Basically we generate training instances with input (t, t-1, ... t-n) and output (t+2). The predicted value is t+2 instead of t+1 because a scaler using this model would need to have a buffer window during which to deploy the services.

After preparing the dataset two baseline models were prepared: the naive approach (predict traffic in window t+2 to be traffic in t) and a classic timeseries model (ARIMA). The next step was to experiment with different deep learning architectures and measure the results.



(a) Naive prediction

(b) Arima prediction

Figure 1: Baseline predictions

3 Evaluation of the approach

3.1 Validation and tuning

Model tuning and validation was done on the Japanese wikipedia trace because although presenting some patterns it also has some interesting irregularities, like a huge spike which is not repeated. The selected time window for tuning is 10 min. (todo test on other windows - tune on other windows?)

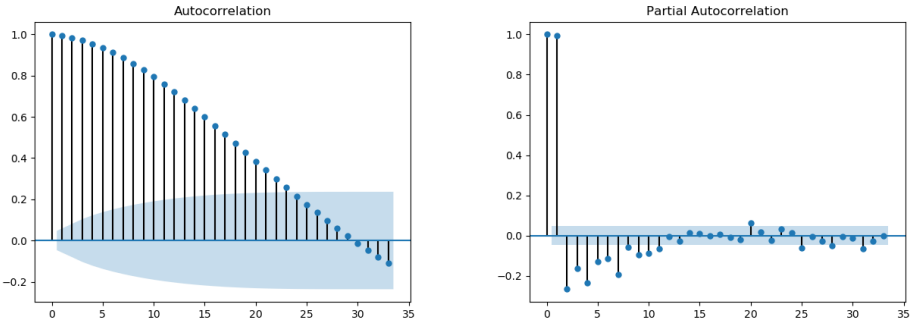
The dataset is split into a training and testing with a ratio of 0.9. The validation method is k-fold Cross-Validation [7] with k = 3, which means splitting the training dataset into k equal parts, perform training on k - 1 and evaluation on the part left out. This process is repeated k times. The main idea is to not touch the testing data while tuning the model, so the model will not be influenced by it.

3.2 Performance metrics

TODO detail mse, mape, mae

3.2.1 Naive baseline

A naive baseline is set, to get an idea if the models are useful at all. The naive predictor simply states that traffic in window t+2 will be the same as in the last measured window. The prediction is ploted in figure 1 and achieves MSE: 20234787.901, MAE: 3572.657, MAPE: 7.072.



(a) Autocorrelation plot (b) Partial autocorrelation plot

Figure 2: ARIMA params

3.2.2 ARIMA

ARIMA [4], which stands for autoregressive integrated moving average, is a classic approach to modelling timeseries. In order to apply this model we need to find appropiate values for its parameters: p, q, d.

The value of d means the number of times the series needs to be differentiated in order to make it stationary. The series stationarity was checked using the augmented Dickey-Fuller test [2] which found the p-value to be 1.0902496274664773e-08. This is lower than 0.05, the commonly used threshold, meaning we can set d to 0.

The partial autocorrelation plot was analyzed to set the autoregression parameter (p). From figure 2 we can see that the significance region is confidently passed at 1, with a steep decline afterwards. The moving average parameter (q) is approximated from the autocorrelation plot. It suggests a value of around 20 would be a good start. After fitting ARIMA(1,0,20) the final 2 layers had P-value of 0.547 and 0.758 which meant that they were not significant. After trying some values for q: 5,10,15,18 the best results were obtained on ARIMA(1,0,15) with MSE: 15289199.210, MAE: 3118.982, MAPE: 6.782.

References

[1] Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. *Machine Learning Strategies for Time Series Forecasting*, pages 62–77. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[2] Yin-Wong Cheung and Kon S Lai. Lag order and critical values of the augmented dickey–fuller test. *Journal of Business & Economic Statistics*, 13(3):277–280, 1995.

[3] Neeraj Joshi Daniel Jacobson, Danny Yuan. Scryer: Netflix's predictive auto scaling engine, November 2013. [Online; posted 05-November-2013].

[4] S. L. Ho and M. Xie. The use of arima models for reliability forecasting and analysis. *Comput. Ind. Eng.*, 35(1–2):213–216, October 1998.

[5] Anshul Jindal, Vladimir Podolskiy, and Michael Gerndt. Performance modeling for cloud microservice applications. pages 25–32, 04 2019.

[6] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey. Cloudinsight: Utilizing a council of experts to predict future cloud application workloads. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 41–48, 2018.

[7] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009.