Web service workload prediction using deep learning - Report 3

Stefan Sebastian

April 22, 2020

Contents

1 State of the art

TODO

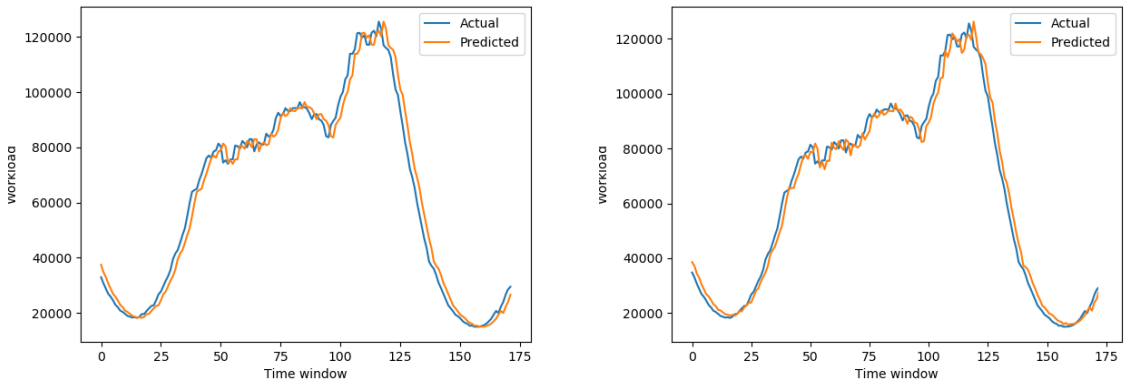
2 Approach

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. [7] 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 [10] 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.

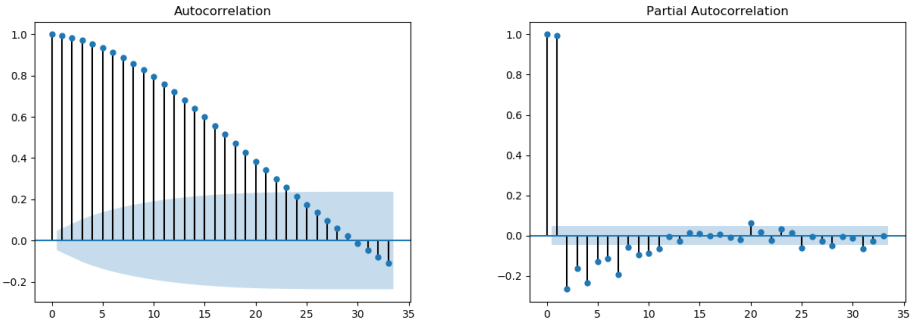
3.2 Performance metrics

TODO detail mse, mape, mae

3.3 Baseline

3.3.1 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

3.3.2 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 14263566.564, MAE: 3056.765, MAPE: 6.349.

3.4 Deep learning models

3.4.1 MLP

After some manual experiments started with a MLP with 2 hidden layers (150, 100 neurons) and sliding window size of 24 (input size).

To find an optimal combination of batch size and epoch no a 2d grid search was performed 3. Batch size should ideally be a power of 2 for extra perfromance on GPU architectures, as some experiments were ran on Google Colab. Lower batch size is more accurate but training is slower [6]. As expected the best MSE is obtained for the lowest batch size(4) however it does not drop significantly at 8, regardless of epochs no. The selection of epoch no is again a tradeoff between speed and accuracy. We see a smaller no

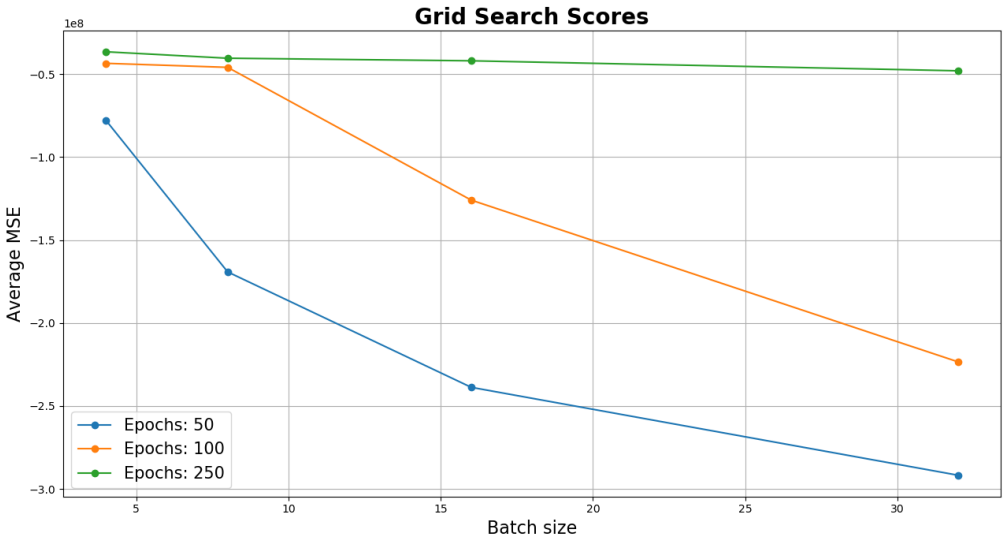


Figure 3: 2d grid search for epoch no and batch size

Table 1: Selecting optimizer and activation. Scores are averaged MSE.

Optimizer	
RMSprop	24734800
Adadelta	19291753
Adagrad	190604349
Adam	25828119
Adamax	29578706
Nadam	20400557

Activation function	
softmax	4875804739
softplus	20314197
softsign	4034049993
relu	19571788
tanh	4175933232
sigmoid	3055609656
linear	20661311

of epochs(50) performs poorly, while the difference between 100 and 250 is not that great, meaning that we can get a good approximation of a model using a batch size of 100.

Some experiments were done with adding Dropout layers on different values (0.2, 0.1, 0.05) however it did not improve performance. These are generally used to prevent overfitting, when the network is too big, the data is scarce or training is done for too long [11], which was not the case for this experiment.

Various optimizer and activation functions were tested. The Adadelta optimizer and the relu activation were selected. A comprehensive grid search was performed for sliding window size and number and content of hidden layers, of around 90 combinations. Some of the best performing are presented in table 2.

Table 2: MLP layers and size tuning

Sliding window	Layers	MSE
4	(100, 50, 25, 20, 10)	18889414
4	(10, 10, 10, 10, 10, 10)	18935098
4	(100, 50, 50, 20, 10)	18847516
8	(100, 50, 25, 20, 10)	17044955
8	(150, 50, 50, 50, 50, 10)	17216134
8	(50, 50, 50, 50)	18394493
16	(10, 20, 30, 40, 50)	18116299
16	(100, 20, 20, 20, 10)	18466524
16	(10, 10, 10, 10, 10, 10, 10)	18311036

Table 3: CNN layers and size tuning

Sliding window	Layers	MSE
8	(25, 10, 5)	35385451
64	(100, 20, 10, 5)	35012864
128	(100, 20, 10, 5)	21086942
128	(300, 50)	22287266
128	(10, 10, 10)	23441869
256	(100,20,10,5)	23783826

3.4.2 CNN

Firstly, a baseline model was selected through manual experimentation. This had the following structure: intput of size 20, a 1d convolutional layer, a maxpooling layer, a flatten layer, a dense layer of size 150 and the output layer. The same batch size, epoch no grid search was performed and it yielded similar results to 3. This was followed by iterating the same optimizers and activation function in order to select Adadelta and softplus.

The main idea behind using a CNN model for this task is to pass a larger history window as input. CNN layers build different filter that learn certain characteristics of the input data. They are well suited for data which has a spatial relationship(like timeseries) [8].

3.4.3 CNN-LSTM Hybrid

This model was applied on a range of timeseries tasks by Lin et al. [9]. It relies on LSTM to find long range dependencies and historical trends and on CNN to extract

Table 4: CNN-LSTM layers and size tuning

Sliding window	**LSTM layers**	**MSE**
144	(20, 10, 5)	51700082
todo	todo	todo

important features from raw timeseries data. The starting values for some parameters were influenced by this research: 32 cnn filters, 1 lstm layer with a couple hundred units.

3.5 Experiment Results

Each of the most promising models were then trained again on all training data, for a larger number of epochs and repeated multiple times to account for the random weight initialization. The best results were then compared to baselines and to eachother as seen in table 5.

Table 5: Final results

Dataset	**Naive**	**ARIMA**	**MLP**	**CNN**	**CNN-LSTM**
Jp10	20234787	15289199	8591086	12766376	todo
Jp15	87806378	56662039	todo	todo	todo
De10	todo	todo	todo	todo	todo
De15	todo	todo	todo	todo	todo

TODO t=5,10,15 TODO german wiki 5,10,15 TODO compare with cloudinsight article

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] Nitish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tang. On large-batch training for deep learning: Generalization gap and sharp minima. 09 2016.

[7] 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.

[8] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[9] Tao Lin, Tian Guo, and Karl Aberer. Hybrid neural networks for learning the trend in time series. pages 2273–2279, 2017.

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

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.