ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 26-28 April 2017, i6doc.com publ., ISBN 978-287587039-1.
Available from http://www.i6doc.com/en/.

Deep convolutional neural networks for detecting noisy neighbours in cloud infrastructure

Bruno Ordozgoiti[1], Alberto Mozo[1], Sandra Gómez Canaval[1],
Udi Margolin[2], Elisha Rosensweig[2], Itai Segall[2] *

1- Departamento de Sistemas Informáticos
Universidad Politécnica de Madrid

2- Nokia Israel

Abstract. Cloud infrastructure in data centers is expected to be one of the main technologies supporting Internet communications in the coming years. Virtualization is employed to achieve the flexibility and dynamicity required by the wide variety of applications used today. Therefore, optimal allocation of virtual machines is key to ensuring performance and efficiency. Noisy neighbor is a term used to describe virtual machines competing for physical resources and thus disturbing each other, a phenomenon that can dramatically degrade their performance. Detecting noisy neighbors using simple thresholding approaches is ineffective. To exploit the time-series nature of cloud monitoring data, we propose an approach based on deep convolutional networks. We test it on real infrastructure data and show it outperforms well-known classifiers in detecting noisy neighbors.

1 Introduction

In the next years, it is expected that more than 90 percent of Internet traffic will go through data centers, which now rely strongly on virtualization. Thanks to mature software stacks and to the widespread availability of virtualization platforms all over the world, the Cloud paradigm is now available for many applications of different kinds in these data centers. Despite the advantages of virtualized infrastructure, this new setting poses new management challenges, such as optimal virtual machine (VM) placement. Nowadays, virtualization is used in Cloud Computing as the sole mechanism to provide performance isolation between multiple tenants. In principle, there should be a clear separation between the different tenants on the same physical machine. In practice, however, this isolation is far from perfect and many resources such as internal networking and memory access resources are shared at some level, which can significantly impact performance. Noisy neighbor is a term commonly used to describe the situation in cloud computing where applications or VMs running on the same cloud node compete for resources such as memory, CPU or network bandwidth, resulting in a degradation of performance [1]. This problem is very interesting to Cloud infrastructure managers for two reasons. Firstly, it is not easy to detect, as performance degradation can happen for different reasons, such as increased

*The research leading to these results has received funding from the European Union under the FP7 grant agreement n. 619633 (project ONTIC) and H2020 grant agreement n. 671625 (project CogNet).

load on the application itself. Secondly, once detected it is easy to address, as relocating one of the machines usually solves the problem. The identification of this problem in real time is a critical building block in creating flexible, reliable and autonomous orchestration mechanisms for Cloud infrastructure.

The problem of determining whether or not the behavior of a VM is being caused by the presence of a noisy neighbor is non-trivial. Based on the available resource monitoring metrics (e.g., cpu time, memory usage, I/O bandwidth), a simple thresholding approach or a set of rules would not suffice [2]. Therefore, to address this problem we propose the use of machine learning methods, and in particular to model it as a classification problem. To the best of our knowledge, there is no previous work addressing the application of machine learning to this problem, with the exception of previous work by the authors [2], where well-known classifiers (support vector machines and random forests) were applied to determine whether or not a noisy neighbor situation was taking place.

In this paper we propose the application of convolutional neural networks to the noisy neighbor problem in order to obtain substantially better accuracy results. In particular, we exploit the temporal nature of the metrics collected at virtualized cloud infrastructure to design a convolutional neural network architecture for this purpose. We show that convolutional networks outperform other machine learning methods, attaining high levels of accuracy. We note that significant depth in the architecture was key to achieving these results. The developed methods are evaluated using data collected at real virtualized infrastructure from a specific scenario set up in Nokia premises.

2 Problem setting

We set up an environment on real cloud infrastructure to generate and collect a data set containing both normal behavior and noisy neighbor events. The resulting records are then labelled according to whether or not they correspond to a noisy neighbor, so that a learning model can be trained to detect them. The environment consists of various physical nodes, where different VMs are instantiated. One or more of these VMs offer a service, and the rest act as noisy neighbors, creating load on the same physical nodes. A noisy neighbour is defined as (one or more) VMs sharing resources with the server under test, thus affecting its performance. In our setting we focused on CPU noise, i.e. we try to detect when a VM is suffering intereference in its access to CPU resources. Collecting relevant metrics is relatively computationally demanding, so we consider the problem of detecting noisy neighbors using a very small number of them collected at the server VM: (1) CPU usage; (2) Inbound network traffic; (3) Outbound network traffic. We also collected CPU usage of the noise VM(s) to label the data records as noisy neighbors or normal behavior. We emphasize that with these three features, thresholding mechanisms or linear classifiers are not effective for detecting noisy neighbors [2]. To deal with missing values and inconsistencies in the sampling frequency, we aggregate metrics over 30-second periods and take their mean. A sample of the resulting dataset is shown in table 1.

CPU	BW-In	BW-Out	Noisy neighbor
47.13	11243.14	10316.18	No
48.06	11778.70	10930.87	No
57.06	12771.26	7066.166	Yes

Table 1: A small sample of the data set

3 Convolutional Neural Networks

In order to design an effective classifier for this scenario, we exploit the time-series nature of the data. The key observation is that analyzing the data over a sufficiently wide time interval is likely to yield more information on whether or not a noisy neighbor is occuring than simply taking the last reading into account. A first approach is simply to concatenate various data samples together. As we show in the experiments, this simple approach significantly boosts the performance of random forests.

To achieve further improvements, we take advantage of the adequacy of convolutional neural networks [3] for data of this kind, and propose the following architecture. Each input sample consists of 11 consecutive readings concatenated together (11 worked best on our data, but different lengths can be considered). Each of the three input features is fed to the network in a separate channel. The resulting data set is thus an $N \times T \times D$ tensor, where N is the number of data points (the total number of records minus the number of concatenated readings), T is the length of the concatenated strings of events and D is the number of collected features. Each of the resulting tensor records, of dimensionality $1 \times T \times D$, is processed by a stack of convolutional layers as shown in figure 1.

The first convolutional layer uses a set of three-channel convolution filters of size c. We zero-pad the input data to preserve its dimensionality. Since the dimensionality is relatively low, we don't employ subsampling so as to allow for depth. Each of these filters therefore produces a vector of length 11, each of whose elements undergoes a non-linear transformation. The resulting vectors are further processed by similar convolutional layers, with as many channels as convolution filters in the previous layer. We can therefore define the entries output by filter f of convolutional layer l at position i given an input record x as

$$a_{f,i}^{(l)} = \begin{cases} \phi\left(\sum_{j=0}^{2} \sum_{k=0}^{c-1} w_{fjk}^{(l)} x_{j,i+k-\lfloor c/2 \rfloor} + b_{fl} \right) & \text{if } l = 0 \\ \phi\left(\sum_{j=0}^{n(l-1)-1} \sum_{k=0}^{c-1} w_{fjk}^{(l)} a_{j,i+k-\lfloor c/2 \rfloor}^{(l-1)} + b_{fl} \right) & \text{otherwise} \end{cases}$$

- $x_{j,i}$ is the value of feature (or channel) j at position i of the input record (if i is negative or greater than 10, then $x_{j,i} = 0$).
- $w_{fjk}^{(l)}$ is the value of channel j of convolution filter f of layer l at position k, and b_{fl} is the bias of filter f at layer l.
- $n(l)$ is the number of convolution filters at layer l.

ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 26-28 April 2017, i6doc.com publ., ISBN 978-287587039-1.
Available from http://www.i6doc.com/en/.

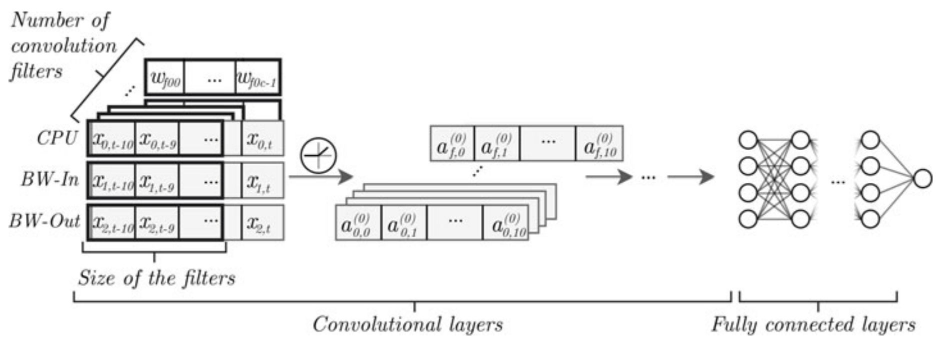

Fig. 1: The proposed convolutional network architecture

- ϕ is the non-linear activation function.

The last convolutional layer is flattened and fed to a feedforward network of fully connected layers.

4 Experiments

We perform experiments to evaluate the performance of the proposed model.

Testbed: The environment consists of five high performance servers with a proprietary management system running on an OpenStack cloud. Two servers are dedicated to management. The rest are compute machines. An open source voice-over-IP (VoIP) application (Asterisk) is deployed on one of the compute machines, in a single VM utilizing one core and 1024 MB of memory. A traffic generator, SipP is deployed on another machine, and configured to initiate calls to a line in which the server is playing music-on-hold. The traffic generator creates constant reasonable load on the server, e.g. 40% CPU utilization. Another application that was considered was a Fibonacci server, designed to mimic virtual network functions in behaviour, but be simple enough for experimentation and analysis. It computes elements in the Fibonacci series on demand. Different implementations were considered to examine the noisy neighbour effect on them. CPU noise was generated using the "stress" Linux application on several VMs launched on the same server as the VoIP server. We experimented with both a single large VM occupying all physical CPUs and multiple small VMs, each occupying only one or two cores. To collect the metrics described in section 2, we used Ceilometer, an OpenStack service for this purpose. We ran around 100 experiments of the described system. The metrics were collected every 10 seconds, and aggregated into 30 second periods to avoid the impact of missing values or irregularities in the sampling frequency. Each of the resulting data points thus represents the average CPU load, inbound and outbound bandwidth of the monitored machine over 30 seconds. The corresponding binary label — the thresholded CPU load of the noise machines— determines whether the noisy neighbor was inflicting load during that time. The resulting data set[1] consists of 9169 records, out of which 3088 correspond to noisy neighbors.

Machine learning models: We design a convolutional neural network like the one described above. We perform a random search of the hyperparameter space and choose the best performing set. Interestingly, we note that despite the relative simplicity of the studied scenario, the employed networks benefit significantly from the addition of depth. This suggests that the noisy neighbor phenomenon manifests itself in a complex fashion, and motivates further research in the application of deep learning methods to network infrastructure management. The best performing model is composed of 6 convolutional layers, each of which learns 32 convolutional patches of width 5 with zero padding. After the convolutional layers, we stack 6 fully connected layers. The convolutional layers are regularized using dropout [4] with a probability of dropping units of 0.275. Dense layers incorporate $l2$ norm penalty with λ=3.75e-5. All layers are batch-normalized [5] and use ReLu units [6] for activation (except for the last one, which is sigmoidal). The model is trained using the Adam optimizer [7], minimizing cross-entropy loss, with minibatches of size 256. We set aside 10% of the data for validation, and keep the model that achieves the best F1 score on the validation set. We stop the training when no improvement is achieved on the validation set (neither in cross-entropy loss nor in F1 score) for 250 epochs.

We compare our model with a random forest (RF) with 500 trees —no significant improvements were obtained beyond that value—, gini index and no depth limit. Support vector machines were also considered, but the results were not significantly better than those shown in [2] and are thus not reported here. We evaluate both models using a 10-fold cross validation (CV) procedure. The folds are chosen randomly without replacement and cover the whole data set (there is no intersection between them). For each model we report precision, recall, F1 score and the area under the ROC and precision-recall curves in table 2. For each value we report the average and standard deviation obtained during the 10-fold CV process. Repeated runs of the same experiment with different randomly chosen folds yielded similar results. Figures 2 and 3 show the ROC and precision-recall curves for the models that attained the AUC closest to the average value. These results show that the proposed CNN model consistently outperforms RFs by a noticeable margin. It should be noted that while each RF model only took a few seconds to train, the CNNs took about 10 minutes for each fold on a GTX 1080 GPU. Predictions, however, were significantly faster on average with CNNs (7.14e-5 seconds) than with RFs (6.16e-4 seconds).

5 Conclusions

We have proposes a method to detect noisy neighbors —virtual machines negatively impacting each other's performance—, a recurring problem in cloud infrastructure. We have designed a deep convolutional network architecture to

[1]The data set is available at http://www.cognet.5g-ppp.eu/wp-content/uploads/2015/08/NoisyNeighbour.zip

	Random forests (500 trees)	CNN
Precision	0.9461 ±0.0062	0.9697 ±0.0056
Recall	0.9406 ±0.0079	0.9318±0.0063
F1 score	0.9432 ±0.0033	0.9502 ±0.0032
AUC-ROC	0.9868 ±0.0021	0.9905 ±0.00077
AUC-PR	0.9881 ±0.0024	0.9913 ±0.0016

Table 2: Performance of the best models.

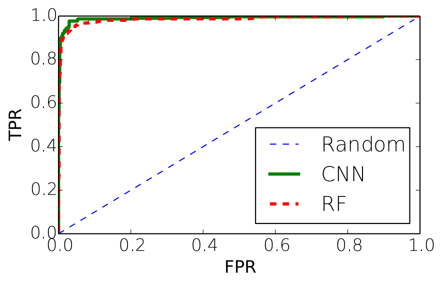

Fig. 2: ROC curves for RFs and CNNs

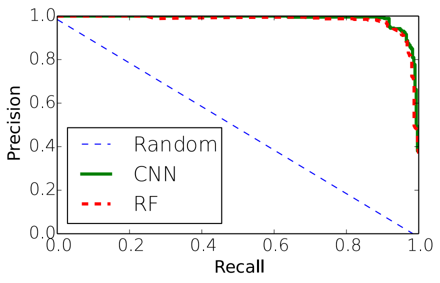

Fig. 3: PR curves for RFs and CNNs

exploit the time-series nature of the data. Using monitoring metrics collected at real data center infrastructure, we have shown that the proposed convolutional network outperforms well-known classifiers. We observe that depth was crucial for this result, motivating further research in the application of deep learning to cloud infrastructure management.

References

[1] Seyed Hossein Nikounia et al. Hypervisor and neighbors noise: Performance degradation in virtualized environments. *IEEE Transactions on Services Computing*, PP(99), 2015.

[2] Udi Margolin, Alberto Mozo, Bruno Ordozgoiti, Danny Raz, Elisha Rosensweig, and Itai Segall. Using machine learning to detect noisy neighbors in 5g networks. *arXiv preprint arXiv:1610.07419*, 2016.

[3] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[4] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[6] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[7] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.