Embedding principal component analysis for data reduction in structural health monitoring on low-cost IoT gateways

Published as a conference paper at ACM International Conference on Computing Frontiers 2019

Embedding Principal Component Analysis for Data Reduction in Structural Health Monitoring on Low-Cost IoT Gateways

Alessio Burrello*
Alex Marchioni*
alessio.burrello@unibo.it
alex.marchioni@unibo.it
DEI, University of Bologna
Bologna, Italy

Davide Brunelli
davide.brunelli@unitn.it
DII, University of Trento
Trento, Italy

Luca Benini
lbenini@iis.ee.ethz.ch
DEI, University of Bologna
Bologna, Italy
IIS, ETH Zurich
Zurich, Switzerland

ABSTRACT

Principal component analysis (PCA) is a powerful data reduction method for Structural Health Monitoring. However, its computational cost and data memory footprint pose a significant challenge when PCA has to run on limited capability embedded platforms in low-cost IoT gateways. This paper presents a memory-efficient parallel implementation of the streaming History PCA algorithm. On our dataset, it achieves 10× compression factor and 59× memory reduction with less than 0.15 dB degradation in the reconstructed signal-to-noise ratio (RSNR) compared to standard PCA. Moreover, the algorithm benefits from parallelization on multiple cores, achieving a maximum speedup of 4.8× on Samsung ARTIK 710.

KEYWORDS

Embedded platforms, Streaming PCA, Structural Health Monitoring, Edge computing, IoT

1 INTRODUCTION

The Internet of Things (IoT) envisions billions of devices that can sense, compute and potentially communicate with users or among them (machine-to-machine) [22]. The IoT involves large sensors and devices networks and poses new challenges in finding innovative and scalable approaches to collect and to process the potentially huge amount of data. A very promising IoT application is Structural Health Monitoring (SHM) [2, 24] whose aim is to provide information about the building conditions. To improve the maintainability of large structures such as space vehicles [10], masonry buildings [4], or bridges [20], large sensors networks are installed and supervised by local gateways. These monitoring systems continuously acquire and transmit data to a central unit for storage and processing purposes. Depending on the scenario, a single gateway manages the data flow from a group of sensors, with a possible focus on either the sensors-gateway communication, as in [5, 19], or on the link from the gateway to the main processor [13, 14].

Even though the role of the central unit is often played by a cloud platform [6], the distribution of the processing to the edge of the cloud, i.e. by including gateways and sensors in the data processing chain, has demonstrated several advantages such as improved security, reduced latency and lower costs [23].

In this framework, reducing the cloud storage space and the network traffic from the gateway to the cloud is extremely useful in many application scenarios [1]. Therefore, a compression algorithm running on the gateway is required. Among compression approaches, methods based on the Principal Component Analysis [11] (PCA) are drawing attention in the processing of large dataset [25]. For these class of methods, the dimensionality reduction is achieved by exploiting the correlation between data features. As a counterpart, an accurate estimation of the signal correlation requires greater memory and computational resources compared to other techniques [15]. This gap is partially filled by the recently emerging streaming approaches [3, 9, 16, 26], that, with a lower memory footprint, exhibit a high accuracy in the correlation estimation. Particularly, the History PCA [26] is demonstrated to outperform other streaming approaches. In details, the History PCA has already been tested on four different large-scale public datasets (NIPS and NYTimes from UCI data, and RCV1, KDDB from LIBSVM data sets as presented in [26]) achieving a lower approximation error compared to other algorithms. History PCA achieves an approximation error (difference between the computed eigenvalues and the PCA eigenvalues) lower than 10^{-6}, which is well suited for many tasks like anomaly detection or spectral analysis.

Tests conducted in [12, 21] explore PCA solutions to reduce the traffic load exploiting spatial correlation among sensors. One alternative solution to enhance the compression is to exploit the autocorrelation of the single sensor time series. This allows to treat the sensor as a unique entity, running a single PCA instance for each sensor.

In this paper we tackle the challenge of embedding PCA compression on low-cost IoT gateways, and we address two key limitations of these HW platforms, namely: limited on-board memory and low computational power. We describe the following contributions:

- We present the application of the History PCA on a real-life SHM problem, i.e. the monitoring of a viaduct. To the best of our knowledge, this is the first attempt to embed the training of a Principal Component Analysis algorithm on a large scale sensor network on a low-cost gateway. We efficiently train our algorithm on a single-sensor temporal series and we demonstrate the suitability of the gateway to manage a 45 sensor (3-axis accelerometers) network.

*Both authors contributed equally to this research.

CF '19, April 30-May 2, 2019, Alghero, Italy

ACM ISBN 978-1-4503-6685-4/19/05…$15.00
https://doi.org/10.1145/3310273.3322822

- Our HPCA implementation on an ARTIK 710 Module [7] with 8 cores achieves 4.8× training speed-up and 2.1× energy saving compared to its single core execution. We also demonstrate that the ARTIK 710 is superior to Raspberry Pi 3 [8], reaching 2.6× higher speed-up with 1.2× energy reduction.
- We further investigate how different configurations of the History PCA could fit different memory constraints, showing that we can save up to 59× memory.

The rest of the article is organized as follows: Sec. 2 presents the PCA and the History PCA algorithm. Sec. 3 introduces our dataset and describes the platforms. Finally, Sec. 4 discusses the experimental results, while Sec. 5 concludes the paper with final remarks.

2 HISTORY PCA

Given a dataset $X \in \mathbb{R}^{d \times N}$ with d features and N instances, the PCA consists in the eigen decomposition of the correlation matrix

$$C_x = \frac{1}{N} X X^\intercal = Q \Lambda Q^\intercal \tag{1}$$

where $Q \in \mathbb{R}^{d \times d}$ is an orthonormal matrix containing the column eigenvectors of C_x and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_\text{d})$ is the eigenvalues matrix with $\lambda_1 \geq \lambda_2, \geq \ldots, \geq \lambda_d$.

Dimensionality reduction is obtained by projecting the signal onto the k eigenvectors corresponding to the higher eignevalues. The average reconstruction error is the sum of the eigenvalues corresponding to the eigenvectors not involved in the projection. Let $x \in \mathbb{R}^d$ be a signal instance, and $\hat{x}$ its reconstruction, the compression error is computed as:

$$\mathbb{E}_x \left[\|x - \hat{x}\|_2 \right] = \sum_{j=k+1}^{d} \lambda_j \tag{2}$$

In this paper we apply the PCA-based compression to time series, by representing signals as non-overlapping windows of d subsequent samples. Firstly, N signal instances are collected in order to estimate the correlation matrix C_x (Eq. 1). As a result, by following Eq. 2, it is possible to fix the average compression error and determine the number of top eigenvectors involved in compression, k. Then, each next incoming signal instance $x \in \mathbb{R}^d$ is compressed by $y = Q^{(k)\intercal} x$, where $Q^{(k)} \in \mathbb{R}^{d \times k}$ is the matrix with the top k egenvectors. Eventually, signal recovery is obtained by $\hat{x} = Q^{(k)} y$.

The classical approach entails to simultaneously store all the N windows and then perform the PCA analysis, thereby demanding for a large memory array. In contrast, the edge computing platforms are tightly constrained by limited memory space. It is therefore interesting to adopt streaming approaches [3, 9, 16, 17, 26], which follow an incremental scheme to update the eigenvectors estimate every new block of B instances, with $B \ll N$. Among them, History PCA (HPCA) [26] has recently emerged. Based on the block stochastic power method [16], HPCA aims to improve accuracy by using more information about past signal instances.

The HPCA method is provided in Alg. 1 and it is deeply explained in [26]. The algorithm runs a new step τ every time a block $X_\tau \in \mathbb{R}^{d \times B}$ is gathered updating the top k eigenvectors estimate as columns of $Q_\tau \in \mathbb{R}^{d \times k}$. After $n = N/B$ steps the method returns the final estimate Q_n.

Algorithm 1 HPCA

1: Input: $X_1, \ldots, X_n$, block-size: B.
2: $S_0^{(i)} \sim N(0, I_{d \times d}), 1 \leq i \leq k$
3: $Q_1 \leftarrow QR_{decomposition}(S_0)$
4: **for** $i \leftarrow 1, \ldots, m$ **do**
5: $\quad S_1 \leftarrow Q_1 + \frac{1}{B} X_1 X_1^\intercal Q_1$
6: $\quad Q_1, \cdot \leftarrow QR_{decomposition}(S_1)$
7: **end for**
8: $\lambda_j \leftarrow \|S_1[:, j]\|_2$ for $j = 1, \ldots, k$
9: $\Lambda_1 \leftarrow diag(\lambda_1, \ldots, \lambda_k)$
10: **for** $\tau \leftarrow 2, \ldots, n$ **do**
11: $\quad Q_\tau \leftarrow Q_{\tau-1}$
12: $\quad$ **for** $i \leftarrow 1, \ldots, m$ **do**
13: $\quad\quad S_\tau \leftarrow \frac{\tau-1}{\tau} Q_{\tau-1} \Lambda_{\tau-1} Q_{\tau-1}^\intercal Q_\tau + \frac{1}{\tau} \frac{1}{B} X_\tau X_\tau^\intercal Q_\tau$
14: $\quad\quad Q_\tau, \cdot \leftarrow QR_{decomposition}(S_\tau)$
15: $\quad$ **end for**
16: $\quad \lambda_j \leftarrow \|S_\tau[:, j]\|_2$ for $j = 1, \ldots, k$
17: $\quad \Lambda_\tau \leftarrow diag(\lambda_1, \ldots, \lambda_k)$
18: **end for**
19: Output: Q_n

The core of the algorithm is contained in line 13, which represents a generalization of the classical power method ($\omega_\tau \leftarrow \frac{1}{B} X_\tau X_\tau^\intercal \omega_{\tau-1}$ with ω as top eigenvector estimate). The second term of the addition represents the power method of the current block whereas the first contains the information extracted from the previous blocks; the two τ-based coefficients weight the terms so that each block equally contributes to the final estimate. Besides, the QR decomposition of line 14 is necessary to ensure the orthonormality among the eigenvectors estimate. The repetition of these steps for m times ensures the estimate to quickly converge to the actual eigenvectors with good accuracy [26].

The computational cost of the HPCA is dominated by the matrix multiplications $O(dk(k+B))$ and by the QR decomposition $O(dk^2)$. The memory occupancy is $\sim O(d(k+B))$, implying a gain of $N/(B+k)$ with respect to the classical PCA. With $B = 1$ the algorithm reaches its lowest memory footprint.

These values are consistent with other streaming PCA algorithm known in literature [3, 16, 18] which present complexity $O(dk^2)$ and $O(dk)$ memory. The advantages of HPCA reside in the robustness in the parameter tuning as shown in Section 4.1 and the improved rate of convergence granted by the iteration of the internal loop as demonstrated in [26].

Summarizing, the HPCA algorithm is characterized by the following parameters:

- signal dimension d: a high value of d allows greater compression ratio, but introduces a delay equal to d/f_s that could not meet real-time requirements.
- rank k: the number of eigenvectors to estimate. It influences the compression ratio CR = d/k and, consequently, the compression quality.
- block size B: number of signal instances used in each step of HPCA algorithm.
- number of repetitions of the HPCA internal loop m.

The computational cost of the HPCA is dominated by the matrix multiplications $O(dk(k+B))$ and by the QR decomposition $O(dk^2)$. Besides, the memory occupancy is $\sim O(d(k+B))$, implying a gain of $N/(B+k)$ with respect to the classical PCA. With $B = 1$ the algorithm reaches its lowest memory footprint.

3 DATASET & PLATFORMS OVERVIEW

In this section, we describe the dataset, and the structural health monitoring installation. Then, we introduce the Artik 710 module, and the Raspberry Pi 3 platforms. We have chosen these two platforms as representatives of the last generation of gateways for edge computing.

3.1 Dataset

Our experiments target a structural health monitoring (SHM) installation on a viaduct which is monitored by 90 sensor nodes equipped with a 3-axis accelerometer, temperature and humidity sensor. The nodes sense the vibrations of the internal tendons in different sections with sample frequency $f_s = 100$ Hz. The readings are gathered by two gateways which pack the data as 32-bit integer and send it to the cloud for storage and analysis purposes. The PCA-based compression is used to reduce bandwidth and storage space on the cloud and it is applied independently to each time series produced by each acceleration axis.

For our analysis, we use a dataset comprising a single-axis time series of the accelerometer mounted on a sensor node. We consider two traces, each one 12 hours long, acquired in two different days. The former is used to estimate the top k eigenvectors while the latter is needed to measure the performances in terms of quality of reconstruction. As a figure of merit, we use the Reconstruction Signal to Noise Ratio (RSNR) defined as:

$$\text{RSNR} = \frac{\|\mathbf{x}\|_2^2}{\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2}$$

where x is the signal instance and $\hat{x}$ is the reconstructed instance after compression.

3.2 Platforms overview

3.2.1 Raspberry Pi 3. The Raspberry Pi 3 Module B [8] (Rpi3) is a single-board computer initially developed for teaching application. Now, it is actively used in many fields such as robotics, smart sensor control, and structural health monitoring. The board comprises the Broadcom BCM2837 SoC, equipped with a 1.2 GHz 64-bit 4-core Cortex A-53, and 1GB low power DDR2 clocked at 900 MHz.

3.2.2 Samsung Artik 710. The Samsung ARTIK 710 Module [7] is an embedded computing System-in-Module targeted at high-end gateways with local processing and analytics. It consists of a 8-core 64-bit ARM Cortex-A53 running at 1.4 GHz with 256KB shared L2-Cache, and two 512MB DDR3 16-bit memory chips with 32-bit memory interface, which provides a throughput of 6.4 GB/s.

We use the default power mode for all our experiments on the Rpi 3 and on the Artik 710 Module, and an external Keithley 2400 SourceMeter SMU for power measurements.

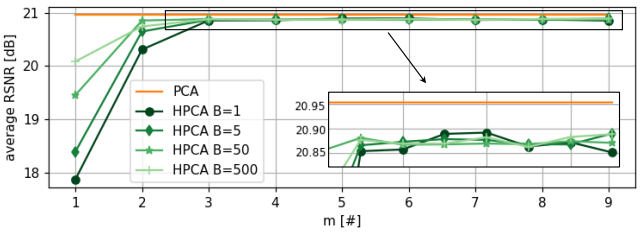

Figure 1: Average RSNR depending on the number of HPCA internal loops m for different values of block-size B with $N = 8650$ (12 hours), $d = 500$, $k = 50$.

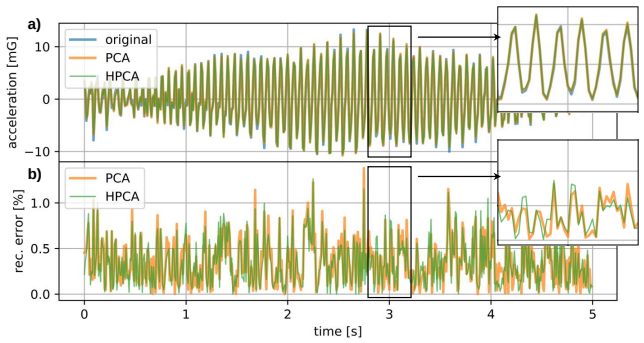

Figure 2: In a) a signal instance compared with its reconstructions after compression based on both PCA (RSNR=20.39dB) and HPCA (RSNR=20.31dB). b) shows the reconstruction errors relative to the RMS value of the signal instance.

4 RESULTS

The History PCA is implemented using optimized Numpy Python 3.5 library, relying on highly optimized BLAS and LAPACK libraries for linear algebra computation. In the following, we first present the reconstruction error and the tuning of parameters of the HPCA algorithm on our dataset (Sec. 4.1). Then we analyze memory occupation, execution time (along with the parallelization speed up), and energy consumption of the algorithm, on both the ARTIK 710 and the Rpi 3 introduced in Sec. 3.2 (Sec. 4.2).

4.1 Use case: Vibration based SHM

To evaluate the performance of HPCA on our dataset, we fix d to 500 samples (corresponding to 5 s) and the compression quality to 20 dB. We consequently set $k = 50$ (CR $= 10$). The parameters B and m are tuned to achieve the best performance.

Fig. 1 depicts the average RSNR with varying B and m. For low values of m the HPCA suffers from up to 3 dB RSNR loss compared to PCA. Lower values of B emphasize the performance gap. By increasing m to 3 or higher values, the RSNR difference reduces to just 0.15 dB and becomes independent of the block-size B. This result highlights the HPCA robustness to parameters variation and it represents the main advantage of the HPCA with respect to the other methods [3, 9, 16, 18] which need a fine parameter tuning.

We therefore fix $m = 3$ and $B = 50$.

Table 1: Performance of History PCA on Artik 710 versus Raspberry Pi 3. Results refer to the execution with d = 5000, B = 1, k = 500, m=3; MM, QR stand for matrix multiplication and QR decomposition.

CORES [#]	1	2	4	8
Samsung ARTIK 710				
time [s]	50.7 (1×)	28.0 (1.8×)	15.6 (3.3×)	10.5 (4.8×)
MM [s]	29.8 (1×)	16.5 (1.8×)	9.3 (3.2×)	5.4 (5.5×)
QR [s]	19.8 (1×)	11.4 (1.7×)	6.7 (3.0×)	5.1 (3.9×)
Rasberry Pi 3 Model B				
time [s]	59.7 (1×)	34.8 (1.7×)	23.1 (2.6×)	n.a.
MM [s]	37.7 (1×)	19.9 (1.9×)	12.6 (3.0×)	n.a.
QR [s]	21.6 (1×)	14.6 (1.5×)	12.7 (1.7×)	n.a.

On our test dataset, the HPCA achieves an average RSNR = 20.88 dB, almost matching the PCA performance (average RSNR = 20.95 dB). As we expect, varying the value of B from 1 to 500 does not impair the HPCA performance (RSNR $\in$ [20.85 dB, 20.90 dB]). In Fig. 2 we report an example of signal instance and its reconstructions after compression. Noteworthy, the punctual reconstruction error (defined as $(x-\hat{x})/\|x\|_2$) of HPCA and PCA are highly correlated, meaning that the vectors identified by the two algorithms are practically the same.

4.2 Rpi & Artik Measurements

To measure the execution time, the parallelization speed-up and the energy consumption of the algorithm on the two platforms, we consider a single step of the algorithm, i.e. an iteration of the τ-loop. To discuss the trade-off between memory and energy we execute a full pass through all the training data.

In Table 1, we analyze the timing and the speed up of the algorithm on both the ARTIK 710 and the Rpi3. For this experiment, we decide to increase d to 5000 and maintain a CR of 10×, extending our results to a more general scenario with a very high number of features. With this setting, one HPCA step requires 50.7 s (59.7 s) to run on single-core ARTIK 710 (Rpi3) and benefits of up to 4.8× (2.6×) speed-up from parallelization. Note that the speed-up is strongly limited by the QR decomposition, that nearly accounts for 50% of the execution time in the max-core configuration of both platforms. Indeed, the QR decomposition relies on many sequential steps, which implies a lower speed-up gain from multicore execution (3.9×/1.7× on Artik710/Rpi3). Furthermore, the lower speed up achieved by the Rpi3 with equal parallelization (2.6× vs 3.3× with 4-cores) shows that the application is memory bound by the DDR2 lower throughput.

Fig. 3 further analyzes these aspects for different features' numbers. As expected, both energy and time dramatically increase with the number of features (the complexity is $\sim dk^2$, with $k = d/10$). Moreover, the gap between execution time on Artik 710 and Rpi3 increases with the number of features, confirming that the DDR2 accesses limit the speed of the HPCA execution on Rpi3. With $d = 5000$ the execution of the HPCA is 2.6× faster on the ARTIK 710 and achieves 1.2× energy saving. Overall, the parallelized version of the HPCA on the Artik platform is executed in 21 % of the total time needed for training with $d = 5000$ (10.5 s every 50 s of signal acquisition), fitting well the real-time constraint given by the 100 Hz sampling frequency; decreasing d to 500 reduces the percentage to only 0.8 %, allowing the simultaneous training of more sensors on a single gateway.

We also evaluate the scalability of the algorithm by increasing the size of the block B with fixed value of k and d, respectively, 50 and 500. Note that a lower value of B implies lower memory requirement (memory $\sim O(d(k+B))$), at the expense of a higher number of steps n ($n = N/B$). More in details, the memory complexity of the HPCA algorithm is represented by

$$memHPCA = data_{size} \times (3 \times d \times k + d \times B + k \times k)$$

where $d \times k$ and $k \times k$ represent the occupation of the intermediate data to compute the Q_τ matrix and $d \times B$ represents the input X_i block. Hence, for $B \gg k$, the memory occupation depends linearly on B. On the other hand, increasing the value of B reduces the number of X_i blocks, and consequently the number of steps required by the HPCA to process the whole data needed for the training. For instance, by doubling the block size, HPCA requires 10-20% more CPU-time per step, but the number of execution step drops by 50%, i.e., an increase of B correspond to a decrease in the energy required to complete the training.

Fig. 4 shows the trade-off between necessary memory and energy consumption. As previously exposed, for high values of B ($B \gg k$) the storage for the data block X_τ ($X_\tau \in \mathbb{R}^{d \times B}$) dominates the memory occupation, while for $B \ll k$, it is determined by the matrices S and Q_τ ($S, Q_\tau \in \mathbb{R}^{d \times k}$). By increasing B from 1 to 50, the HPCA necessitates 1.5× memory (from 312 KB to 420 KB) and saves almost 50× energy consumption on both the platforms, whereas still increasing B to 512 causes further 3× memory occupation with only 5× more energy saving. Overall, setting $B = 1$ (minimum memory footprint) allows a 59× memory reduction with respect to standard PCA, moving from 18.3 MB to 312 KB. Moreover, managing our 45 sensor installation with 3 axes time series requires only 42.1 MB, with respect to the 2.5 GB needed by PCA.

Fig. 4 also depicts the different energy consumption between the two platforms. On a single-core, HPCA is compute bound and the Artik module shows higher energy consumption due to the Artik DDR3 RAM which is more power-hungry than the DDR2 mounted on Rpi3. When we move to the multi-core execution, the bottleneck becomes the memory access. The DDR3 in the Artik module grants a quicker access to the RAM that allows the 8 cores to run more efficiently, resulting in lower power consumption with respect to the single-core configuration. Conversely, the DDR2 mounted on RPi3 does not allow the 4 cores to exploit the maximum speed-up since most of the time is spent in load-store operations and the speed-up is not able to compensate the higher consumption due to the multi-core execution.

5 CONCLUSION

This work presents accelerating History PCA on Artik 710 and on Raspberry Pi 3, for data compression in a structural health monitoring system. We obtained the same performance of the standard PCA (10× compression factor with 20 dB of average RSNR), with 59× lower memory footprint. We also compare the execution of the

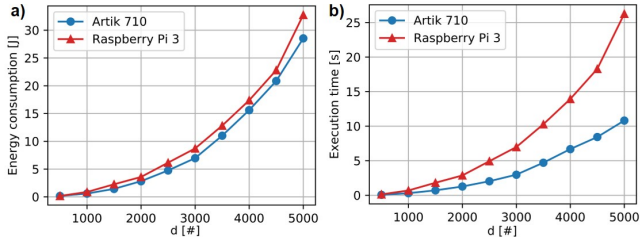

Figure 3: a) energy and b) time comparison of HPCA running on Artik 710 versus Raspberry Pi 3 (maximum core configuration) with varying number of features (d), $k = d/10$, $B = 1$ and $m = 3$.

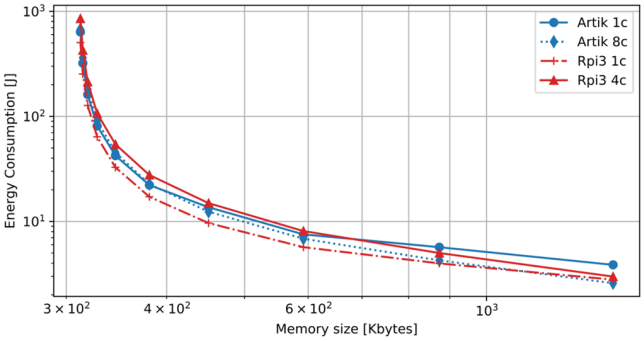

Figure 4: Trade-off between energy consumption and memory occupation on Artik 710 and Raspberry Pi 3 after a full pass over all the training data of the HPCA. Settings: $d = 500$, $k = 50$, $B \in [1, 512]$, $m = 3$.

algorithm on the two platforms for different values of d, demonstrating that Artik 710 achieves 2.6× faster execution and 1.2× energy reduction, with respect to Raspberry Pi 3.

Our future work will focus on moving the computation of the HPCA on the single sensor under even tighter memory and computational effort constraints.

6 ACKNOWLEDGMENTS

The research contribution presented in this paper has been funded by a research grant of ST Microelectronics and by the Emilia Romagna region Doctoral Program.

REFERENCES

[1] M. Aazam and E. Huh. 2014. Fog Computing and Smart Gateway Based Communication for Cloud of Things. In 2014 International Conference on Future Internet of Things and Cloud. 464–470. https://doi.org/10.1109/FiCloud.2014.83

[2] A. Abdelgawad and K. Yelamarthi. 2016. Structural health monitoring: Internet of things application. In 2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS). 1–4. https://doi.org/10.1109/MWSCAS.2016.7870118

[3] Z. Allen-Zhu and Y. Li. 2017. First Efficient Convergence for Streaming k-PCA: A Global, Gap-Free, and Near-Optimal Rate. In 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS). 487–492. https://doi.org/10.1109/FOCS.2017.51

[4] Cem Ayyildiz, H. Emre Erdem, Tamer Dirikgil, Oguz Dugenci, Taskin Kocak, Fatih Altun, and V. Cagri Gungor. 2019. Structure health monitoring using wireless sensor networks on structural elements. Ad Hoc Networks 82 (2019), 68 – 76. https://doi.org/10.1016/j.adhoc.2018.06.011

[5] D. Bortolotti, M. Mangia, A. Bartolini, R. Rovatti, G. Setti, and L. Benini. 2018. Energy-Aware Bio-Signal Compressed Sensing Reconstruction on the WBSN-Gateway. IEEE Transactions on Emerging Topics in Computing 6, 3 (2018), 370–381. https://doi.org/10.1109/TETC.2016.2564361

[6] A. Botta, W. de Donato, V. Persico, and A. PescapÃĬ. 2014. On the Integration of Cloud Computing and Internet of Things. In 2014 International Conference on Future Internet of Things and Cloud. 23–30. https://doi.org/10.1109/FiCloud.2014.14

[7] Samsung ARTIK 7 Family. 2016. Samsung ARTIK 710 IoT Module. (2016). https://www.artik.io/modules/artik-710/

[8] Raspberry Pi Foundation. 2016. Raspberry Pi 3 Module B. (2016). https://www.raspberrypi.org/documentation/hardware/raspberrypi/

[9] Moritz Hardt and Eric Price. 2014. The Noisy Power Method: A Meta Algorithm with Applications. In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14). MIT Press, Cambridge, MA, USA, 2861–2869. http://dl.acm.org/citation.cfm?id=2969033.2969146

[10] M. Hedley, N. Hoschke, M. Johnson, C. Lewis, A. Murdoch, D. Price, M. Prokopenko, A. Scott, P. Wang, and A. Farmer. 2004. Sensor network for structural health monitoring. In Proceedings of the 2004 Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004. 361–366. https://doi.org/10.1109/ISSNIP.2004.1417489

[11] I. T. Jolliffe. 1986. Principal Component Analysis. Springer New York, New York, NY. 115–128 pages. https://doi.org/10.1007/978-1-4757-1904-8_7

[12] J. Li, S. Guo, Y. Yang, and J. He. 2016. Data Aggregation with Principal Component Analysis in Big Data Wireless Sensor Networks. In 2016 12th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN). 45–51. https://doi.org/10.1109/MSN.2016.015

[13] M. Mangia, A. Marchioni, F. Pareschi, R. Rovatti, and G. Setti. 2018. Administering quality-energy trade-off in IoT sensing applications by means of adapted compressed sensing. IEEE Journal on Emerging and Selected Topics in Circuits and Systems 8, 4 (2018), 895–907. https://doi.org/10.1109/JETCAS.2018.2846884 cited By 1.

[14] M. Mangia, F. Pareschi, R. Rovatti, and G. Setti. 2018. Rakeness-based compressed sensing and hub spreading to administer short/long-range communication tradeoff in IoT Settings. IEEE Internet of Things Journal 5, 3 (2018), 2220–2233. https://doi.org/10.1109/JIOT.2018.2828647 cited By 2.

[15] M. Mangia, R. Rovatti, and G. Setti. 2012. Rakeness in the design of analog-to-information conversion of sparse and localized signals. IEEE Transactions on Circuits and Systems I: Regular Papers 59, 5 (2012), 1001–1014. https://doi.org/10.1109/TCSI.2012.2191312 cited By 51.

[16] Ioannis Mitliagkas, Constantine Caramanis, and Prateek Jain. 2013. Memory Limited, Streaming PCA. In Advances in Neural Information Processing Systems 26, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2886–2894. http://papers.nips.cc/paper/5035-memory-limited-streaming-pca.pdf

[17] Erkki Oja. 1982. Simplified neuron model as a principal component analyzer. Journal of Mathematical Biology 15, 3 (01 Nov 1982), 267–273. https://doi.org/10.1007/BF00275687

[18] Erkki Oja. 1983. Subspace methods of pattern recognition. Letchworth, Hertfordshire, England : Research Studies Press ; New York : Wiley. Includes index.

[19] F. Pareschi, M. Mangia, D. Bortolotti, A. Bartolini, L. Benini, R. Rovatti, and G. Setti. 2017. Energy Analysis of Decoders for Rakeness-Based Compressed Sensing of ECG Signals. IEEE Transactions on Biomedical Circuits and Systems 11, 6 (2017), 1278–1289. https://doi.org/10.1109/TBCAS.2017.2740059 cited By 4.

[20] M. Reyer, S. Hurlebaus, J. Mander, and O. E. Ozbulut. 2011. Design of a wireless sensor network for Structural Health Monitoring of bridges. In 2011 Fifth International Conference on Sensing Technology. 515–520. https://doi.org/10.1109/ICSensT.2011.6137033

[21] A. Rooshenas, H. R. Rabiee, A. Movaghar, and M. Y. Naderi. 2010. Reducing the data transmission in Wireless Sensor Networks using the Principal Component Analysis. In 2010 Sixth International Conference on Intelligent Sensors, Sensor Networks and Information Processing. 133–138. https://doi.org/10.1109/ISSNIP.2010.5706781

[22] Elmustafa sayed ali ahmed and Zeinab Kamal Aldein Mohammed. 2017. Internet of Things Applications, Challenges and Related Future Technologies. world scientific news (01 2017).

[23] W. Shi and S. Dustdar. 2016. The Promise of Edge Computing. Computer 49, 5 (May 2016), 78–81. https://doi.org/10.1109/MC.2016.145

[24] C. Arcadius Tokognon, B. Gao, G. Y. Tian, and Y. Yan. 2017. Structural Health Monitoring Framework Based on Internet of Things: A Survey. IEEE Internet of Things Journal 4, 3 (June 2017), 619–635. https://doi.org/10.1109/JIOT.2017.2664072

[25] N. Vaswani, Y. Chi, and T. Bouwmans. 2018. Special Issue on: Rethinking Principal Component Analysis (PCA) for Modern Data Sets: Theory, Algorithms, and Applications. *Proc. IEEE* 106, 8 (Aug 2018), 1269–1457.

[26] Puyudi Yang, Cho-Jui Hsieh, and Jane-Ling Wang. 2018. History PCA: A New Algorithm for Streaming PCA. arXiv (2018). https://arxiv.org/abs/1802.05447