NARPCA: Neural Accumulate-Retract PCA for Low-Latency High-Throughput Processing on Datastreams

Cristian Axenie(✉), Radu Tudoran, Stefano Bortoli, Mohamad Al Hajj Hassan, and Goetz Brasche

Huawei German Research Center, Riesstrasse 25, 80992 Munich, Germany
{cristian.axenie,radu.tudoran,stefano.bortoli, mohamad.alhajjhassan,goetz.brasche}@huawei.com

Abstract. The increasingly interconnected and instrumented world, provides a deluge of data generated by multiple sensors in the form of continuous streams. Efficient stream processing needs control over the number of useful variables. This is because maintaining data structure in reduced sub-spaces, given that data is generated at high frequencies and is typically follows non-stationary distributions, brings new challenges for dimensionality reduction algorithms. In this work we introduce NARPCA, a neural network streaming PCA algorithm capable to explain the variance-covariance structure of a set of variables in a stream through linear combinations. The essentially neural-based algorithm is leveraged by a novel incremental computation method and system operating on data streams and capable of achieving low-latency and high-throughput when learning from data streams, while maintaining resource usage guarantees. We evaluate NARPCA in real-world data experiments and demonstrate low-latency (millisecond level) and high-throughput (thousands events/second) for simultaneous eigenvalues and eigenvectors estimation in a multi-class classification task.

Keywords: Distributed stream processing · PCA · Neural networks

1 Introduction

In many cases, data representations are redundant and the variables are correlated, which means that eventually only a small sub-space of the original representation space is populated by the sample and by the underlying process [24]. Due to considerable practical relevance, there is renewed interest in Principal Component Analysis (PCA), [2,8,34]. PCA is good for maintaining data structure in reduced subspaces in an unsupervised way. It is also useful in updating the decision boundaries and adding discriminately informative features with newly added samples and then updating the feature vectors by incremental eigenvector updates [17,19].

I. V. Tetko et al. (Eds.): ICANN 2019, LNCS 11727, pp. 253–266, 2019.
https://doi.org/10.1007/978-3-030-30487-4_20

The core computational element of PCA is performing a (partial) singular value decomposition, and much work over the last half century has focused on efficient algorithms [11]. The recent focus on understanding streaming high-dimensional data, where the dimensionality of the data can potentially scale together with the number of available sample points, has led to an exploration of the complexity of covariance estimation underlying PCA [4,7,12,16]. Such algorithms have provable complexity guarantees but either store all samples (i.e. for looping through samples) or explicitly maintain the covariance matrix. In high-dimensional applications, where data points are high resolution images, video or bank transactions data, storing all data is prohibitive. Different from previous approaches, our work doesn't only tackle memory constraints, but brings the focus on three critical quantities: latency, throughput and judicious resource allocation.

In the streaming setting, many approaches for incremental or online PCA have been developed, some focusing on replacing the inefficient steps (i.e. high latency of low-rank modifications of the singular value decomposition [18], others on conditioning in householder transformation for QR decomposition etc.) or memory- and computation-efficient operations [5,6,10].

Despite the multitude of successful dedicated algorithms [14,33,35], there is no algorithm that provably recovers the principal components in the same noise and complexity regime as the batch PCA algorithm does and maintains a provably light memory/storage footprint. Moreover, none considers also low-latency and high-throughput at processing streaming data. This work utilizes a new paradigm for stream processing and learning [3] and proposes NARPCA for efficient computation (i.e. incremental learning in neurons) with low-latency and high-throughput (i.e. through orchestration of the stream data flow) while maintaining judicious memory/storage usage.

2 Formalizing the Problem

While much work has focused on memory-constrained PCA, [28,29], there is little work that provides complexity guarantees competitive with batch algorithms, [18], and, to our knowledge, no work on low-latency, high-throughput in large-scale streaming machine learning applications with programmatic resource allocation. In its basic formulation, PCA computes the eigenvectors and eigenvalues of the sample covariance matrix, using a numerical method such as the QR method [24]. This approach requires that all the training data are available before the principal components can be estimated. An incremental method, on the other side, is required to compute the principal components for observations arriving sequentially, where the principal components estimates are updated by each arriving observation [25]. In order to formalize the problem, consider a stream of n-dimensional data vectors $x(t)$. The problem of extracting incrementally the principal components assumes reducing the number of linear combinations of inputs by discarding low variances and only keeping the combinations with large variance: $w_1^T x, w_2^T x, w_3^T x, ..., w_p^T x, p \leq n$ which maximize the expectation

$$E\{(w_i^T x)^2\}_{i=1:p} \tag{1}$$

under the constraints

$$w_i^T w_j = \delta_{ij}, j < i, \tag{2}$$

where δ_{ij} is the Kronecker product. The solution for the vectors $w_1, w_2, w_3, ..., w_p$ are the p dominant eigenvectors of the data covariance matrix calculated incrementally,

$$C = E\{xx^T\}. \tag{3}$$

These are p orthogonal unit vectors $z_1, z_2, z_3, ..., z_p$ given by

$$Cz_i = \lambda_i z_i \tag{4}$$

where $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_p$ are the largest p eigenvalues of C in descending order $\lambda_1 > \lambda_2 > \lambda_3 > ... > \lambda_p$. The first principal component is $c_1^T x$ whereas the minor components are linear combinations $c_n^T x, c_{n-1}^T x$ where c_n is the eigenvector corresponding to the smallest eigenvalue. In a structured form, the basic formulation, PCA follows the following steps:

Data: Stream of n-dimensional data vectors $x(t)$
Result: Eigen features w_p and λ_p
Compute the mean feature vector $\mu = \frac{1}{n}\sum_k^n x_k(t)$;
Compute the covariance matrix $C = E\{x - \mu\}\{x - \mu\}^T$;
Compute eigenvalues λ_i and eigenvectors z_i of C;
while *Estimate high-value eigenvectors* **do**
  Sort eigenvalues λ_i in decreasing order;
  Choose a threshold θ ;
  Choose the first p dominant λ_i to satisfy $(\sum_i^p \lambda_i)(\sum_i^n \lambda_i)^{-1} \geq \theta$;
  Select eigenvectors w_p corresponding to λ_p;
end
Extract principal components from x, $P = V^T x$ where V is the matrix of principal components

For large problems this routine is not computationally efficient and various numerical methods have been used to improve it, using QR decomposition [26] or the Householder reflections [13] replacing Gram-Schmidt, known to lead to cancellation that causes inaccuracy of the computation and a non-orthogonal matrix. In our work we analysed the inefficient operations which impede PCA to achieve the three critical quantities: latency, throughput and memory/storage, when operating on streams. We identified three aspects in the existing approaches which impact a streaming PCA formulation:

- the continuous calculation of the mean μ and other descriptive statistics (i.e. covariance) on the datastream;
- sorting the dominant eigenvalues in the rank update of the QR decomposition and the ordering of lower/upper triangular sub-matrices;
- the complexity of computations performed at each training step.

Stream processing applications distinguish themselves from the traditional store-and-process data analysis through four features: continuous data sources, continuous and long-running analysis, time-to-respond performance requirements, and failure tolerance requirements [1,9]. These characteristics together with the identified bottlenecks will constitute design objectives for our approach that leverage the advantages of PCA for low-latency, high-throughput, resource-efficient stream processing, as shown in the next sections.

3 Materials and Methods

This section covers the design, implementation details, and the motivation to tackle the inherent problems in traditional PCA impeding it to achieve low-latency, high-throughput and fixed memory/storage. Our motivation for this work stems from the need to guarantee performance and efficiency. Dissecting the theory of PCA we found those bottlenecks that kept PCA away from streaming applications.

Employing a novel method and system for online machine learning [3], capable of incrementally computing machine learning models on data streams, we successfully instantiated PCA keeping performance indices (i.e. latency, throughput) and bounded memory allocation in intensive real-world scenarios. The underlying computational mechanism in our approach is the accumulate-retract framework. Such a framework allows for dual model updates as soon as new data comes into the system and leaves the system as the stream progresses. The accumulate-retract framework allows for incremental calculation of the statistical quantities used in the streaming PCA. For example, the average calculation in the eigenvalue update in incremental form can be visualized in Fig. 1.

Such incremental computation tackles successfully the first bottleneck, namely the incremental calculation of the mean μ and other descriptive statistics on the datastream.

The second problem that the accumulate-retract framework solves, is sorting the dominant eigenvalues in the rank update of the QR decomposition. For this, let's assume we need to sort the current list of dominant eigenvalues, as shown in Fig. 2. In this case the caches are used to store contents (i.e. buckets) on updates depending on counts (i.e. histogram). Updates are done in buckets, which contain sorted eigenvalues. Each time new eigenvalues are computed sorting is triggered. In this instantiation the retraction cache stores the last calculated eigenvalues (in time) in each bucket, whereas when the accumulation cache moves according to the sliding convention, new buckets are brought and the entire structure is sorted. Moreover, in this instantiation, the buckets stored on disk or 3rd party storage devices contain data organized based on value/indexes. The last eigenvalue (time-wise) in each bucket has a reference in the retraction cache.

In order to comply with the low-latency, high-throughput requirements of stream processing, our approach uses a single layer neural network, which employs only local, simple operations. There are many models for learning PCA in neurons [22], from stochastic approximation models (i.e. Hebbian and Oja's

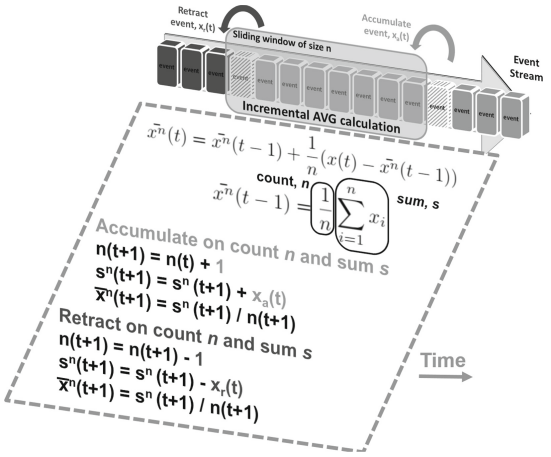

Fig. 1. NARPCA incremental average update using accumulate-retract.

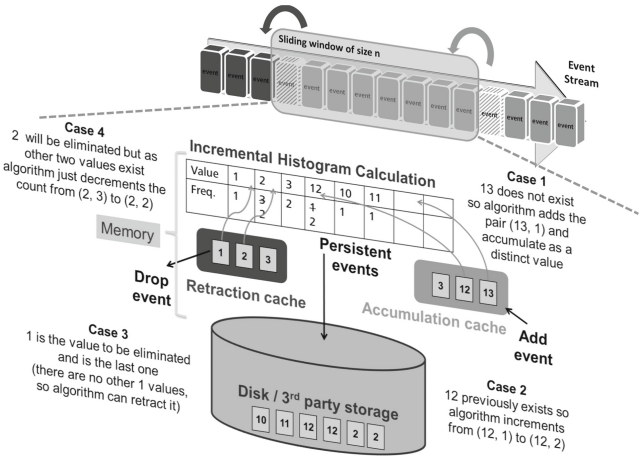

Fig. 2. NARPCA eigenvalues sorting on streams: histogram update.

Learning Rules [20]), to subspace learning [21], nonlinear PCA networks [15] up to denoising autoencoders [27]. NARPCA, Fig. 3, is a single layer neural network based on the Stochastic Gradient Ascent [21]. We chose this model because it efficiently provides a description of the covariance matrix, which is typically too expensive to be estimated online [30]. NARPCA learning rule is given by

$$\Delta w_j(t-1) = \gamma(t) y_j(t)(x(t) - y_j(t) w_j(t-1) - 2 \sum_{i<j} y_i(t) w_i(t-1)) \qquad (5)$$

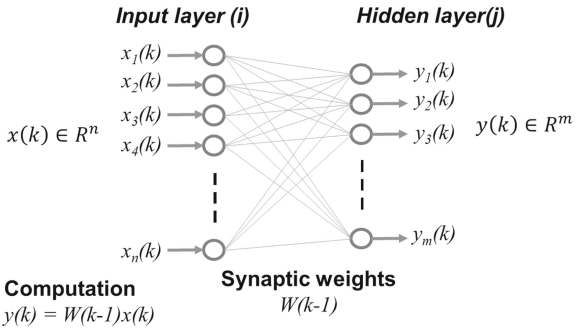

Fig. 3. Neural PCA network.

where $\gamma(t)$ is the learning rate. This learning rule is purely local in the sense that the change in each individual weight only depends on factors that would be locally available at that neuron's position. This rule behaves better for extracting the dominant eigenvectors than other methods [23]. The weight update takes advantage of the accumulate-retract framework, incrementally incorporating new knowledge while decreasing the impact that old data has upon the update, as show in Figure 4. The synaptic weight w_j converge to the eigenvectors c_i as NARPCA finds the unique set of weights which is both optimal and gives uncorrelated outputs. In order to leverage low-latency high-throughput processing on the evolving datastream, we derive the closed form incremental learning rule in the accumulate-retract framework. As shown previously, the eigenvectors $z_1, z_2, z_3, ..., z_p$ are given by

$$Cz_i = \lambda_i z_i, \tag{6}$$

where λ_i are the eigenvalues of the covariance matrix, $C[c_{jk}]$. We can then rewrite C as

$$c_{jk} = \frac{1}{n-1}\sum_{i=1}^{n}(z_{ij} - \bar{z_j})(z_{ik} - \bar{z_k}) \tag{7}$$

where

$$\bar{z} = \frac{1}{n}\sum_{i=1}^{n} z_i \tag{8}$$

is the incremental average. Hence, we can rewrite the covariance matrix C as

$$C = \frac{1}{n-1}\sum_{i=1}^{n}(z_i - \bar{z})(z_i - \bar{z})^T, \tag{9}$$

which is, in fact, the autocorrelation. The problem is that these measures are not robust statistics and hence not resistant to outliers. We replace z with its estimate at time t $z(t)$ so that $v = \lambda z$ estimate at time t is

$$v(t) = \frac{1}{n}\sum_{i=1}^{n} x(t)x^T(t)z(t). \tag{10}$$

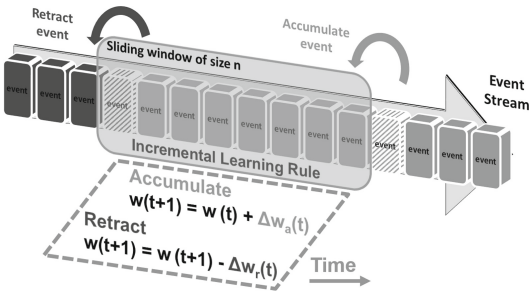

Fig. 4. NARPCA weight update in the accumulate-retract framework.

We can now calculate the eigenvalues and eigenvectors given v as we know that $\lambda = \|v\|$. If we consider

$$z = \frac{v}{\|v\|} \tag{11}$$

we can rewrite

$$v(t) = \frac{1}{n}\sum_{i=1}^{n} x(t)x^T(t)\frac{v(t-1)}{\|v(t-1)\|}. \tag{12}$$

Such computation steps provide a closed form implementation of learning rule in the accumulate-retract framework [3].

NARPCA converges from an initially random set of synaptic weights to the eigenvectors of the input autocorrelation in the eigenvalues order. The optimal weights are found by minimizing the linear reconstruction error $E\{(x-\hat{x})^2\}$ when the rows of W span the first p eigenvectors of C and the Linear Least Squares (LLS) estimate of x given y is

$$\hat{x} = CW^T(WCW^T)^{-1}y. \tag{13}$$

If the rows of W are the first eigenvectors then $WW^T = I$ and $C = W^T\Lambda W$ where Λ is the diagonal matrix of C in descending order. Then, $y = Wx$ is the Karhunen-Loève Transformation (KLT). In the LLS optimization routine, if we have an unknown function f the best estimator of y is

$$\min_f \sum_{i=1}^{n}(y_i - f(x_i))^2. \tag{14}$$

Moreover, if f is linear in x and $y = ax + b$ then the best estimator is the search for the best a, b that minimize

$$\min_f \sum_{i=1}^{n}(y_i - ax_i - b)^2. \tag{15}$$

In the accumulate-retract framework such a problem is incrementally solved as the datastream progresses, using simple updates shown in Fig. 5 Given that covariance can be incrementally calculated as

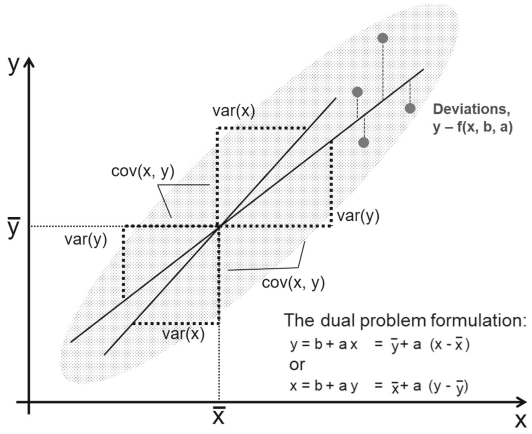

Fig. 5. NARPCA optimization using LLS in accumulate-retract framework.

$$cov_{xy}(t) = \frac{n-2}{n-1} cov_{xy}(t-1) + \frac{1}{n}(x^n(t) - \bar{x}^{n-1}(t))(y^n(t) - \bar{y}^{n-1}(t)), \quad (16)$$

the problem in closed form assumes calculating incrementally a and b as

$$a(t) = \frac{cov_{xy}(t)}{m_2(t)}, b(t) = \bar{y}(t) - a\bar{x}(t) \quad (17)$$

where $m_2(t)$ is the 2^{nd} statistical moment in incremental Yet, up to now we made the assumption that only y values contain errors while x are known accurately. This is not true in practical applications, thus we are seeking the values of a and b that minimize

$$\min_{a,b} \sum_{i=1} \frac{(y_i - ax_i - b)^2}{1 + a^2}, \quad (18)$$

which amounts for the Total Least Squares (TLS). It has been shown that the TLS problem can be solved by performing a Minor Component Analysis (MCA) [32], thus finding the linear combinations (or directions) which contain the minimum variance. Clearly, every eigenvector is a solution of the minimization of TLS error. In the following section we instantiate NARPCA within the accumulate-retract framework for a multi-class classification problem. NARPCA, due to its incremental nature, preserves the discriminant information within the data and can provide classification boundaries [31].

4 Experiments and Discussion

This section introduces the results and the analysis for the instantiation of our framework using Apache Flink [9]. Flink is an open source system for parallel scalable processing on real-time streaming data. At its core, Flink builds on an optimized distributed dataflow runtime that supports our accumulate-retract framework, crucial in obtaining low-latency high-throughput online machine learning. The experimental setup for our tests used 4 machines, each with 24 CPU cores and 196 GB RAM, and Flink for cluster management. During the experiments we consider a fixed sliding window, but the cache-disk orchestration mechanism can support also adaptive windowing. At the same time the caches (i.e. in RAM) mechanism allows to maintain new and old data in order to allow the retraction of individual stream events when sliding. This allows our system to learn from continuous data in a single pass. We used a real-world stream with online bank transactions and queried the eigenvalues and eigenvectors. The data is available online from the PKDD'99 Discovery Challenge - Guide to the Financial Data Set. The dataset and has 10 input features (i.e. transaction id, account id, transaction amount, balance after transaction, transaction bank partner, transaction account partner, transaction type, transaction operation, transaction symbol, transaction date) describing various aspects of the executed transaction. The datastream contained 2M incoming events at 40 kHz. Moreover the datastream had the property that the eigenvalues of the input X are close to the class labels (i.e. $1, 2, ..., d$) and the corresponding eigenvectors are close to the canonical basis of R^d, where d is the number of principal components to extract and the class number for the multiclass classification task (i.e. types of valid and fraud transactions - in our scenario 10 classes). To give the user a sample of the possible output, some sample class labels were, "high-risk fraud", "recurrent fraud", "low-risk valid", "recurrent valid".

In order to evaluate our NARPCA, we implemented an efficient QR-based PCA of [26] using Householder transformation and ran it in the accumulate-retract framework on the same experimental system. Important to note that NARPCA does not need to compute the correlation matrix in advance, since the eigenvectors are derived directly from the data. This is an important feature of NARPCA, particularly if the number of inputs is large. Using the accumulate-retract framework, both algorithms kept the scale of observations and computed the mean of observations incrementally. The scope of our analysis is to emphasize that using simple incremental operations in a single layer neural network and exploiting an efficient data orchestration can leverage low-latency high-throughput streaming PCA with fixed/programmable resource allocation. Such a platform allows NARPCA to learn from datastreams in a single pass. In order to emphasize the advantages the accumulate-retract framework, we now analyse large-scale experiments to extract the eigenfeatures for the multi-class fraud detection scenario (i.e. 2M events streamed at 40 kHz).

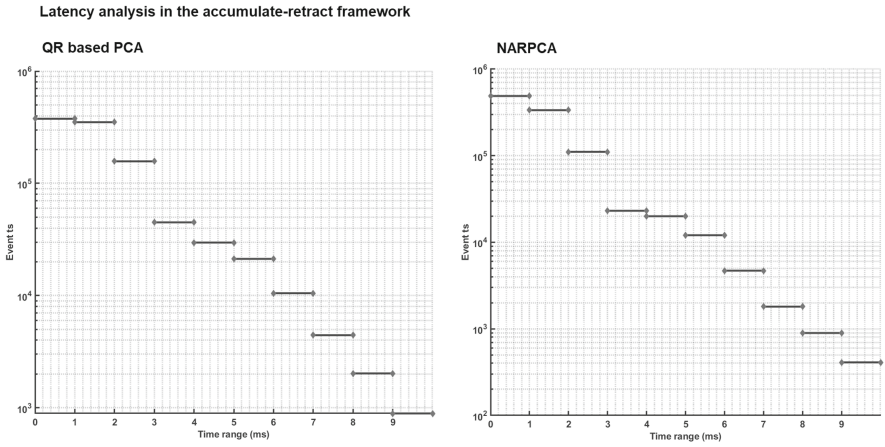

Fig. 6. Comparative analysis of latency in the accumulate-retract framework.

To offer an understanding of the actual estimation performance of the system the next table (Table 1) shows the eigenvalues of the input and how close they are to the class labels (i.e. 1, 2, ..., d=10) and the corresponding eigenvectors variance with respect to the canonical basis of R^d). In terms of latency one can observe that NARPCA outperforms the QR-based PCA, with a substantial distribution of events processed at 1 ms and just a limited number of events processed at over 8 ms, as shown in Fig. 6. This is supported by the gain of 8k events throughput, as shown in Fig. 7. This is also visible in the core distribution of throughput ranges peaking at around 40k events/s. Pushing real-world performance constraints, the accumulate-retract framework instance of NARPCA stands out as a good candidate for low-latency high-throughput systems for dimensionality reduction, in critical applications such as fraud detection. This work is a new instantiation of the accumulate-retract framework [3], promoting a generic streaming machine learning platform with demonstrated potential in real-world applications. The core STARLORD codebase and benchmarking is available at[1](Table 1).

[1] https://github.com/omlstreaming/icmla2018.

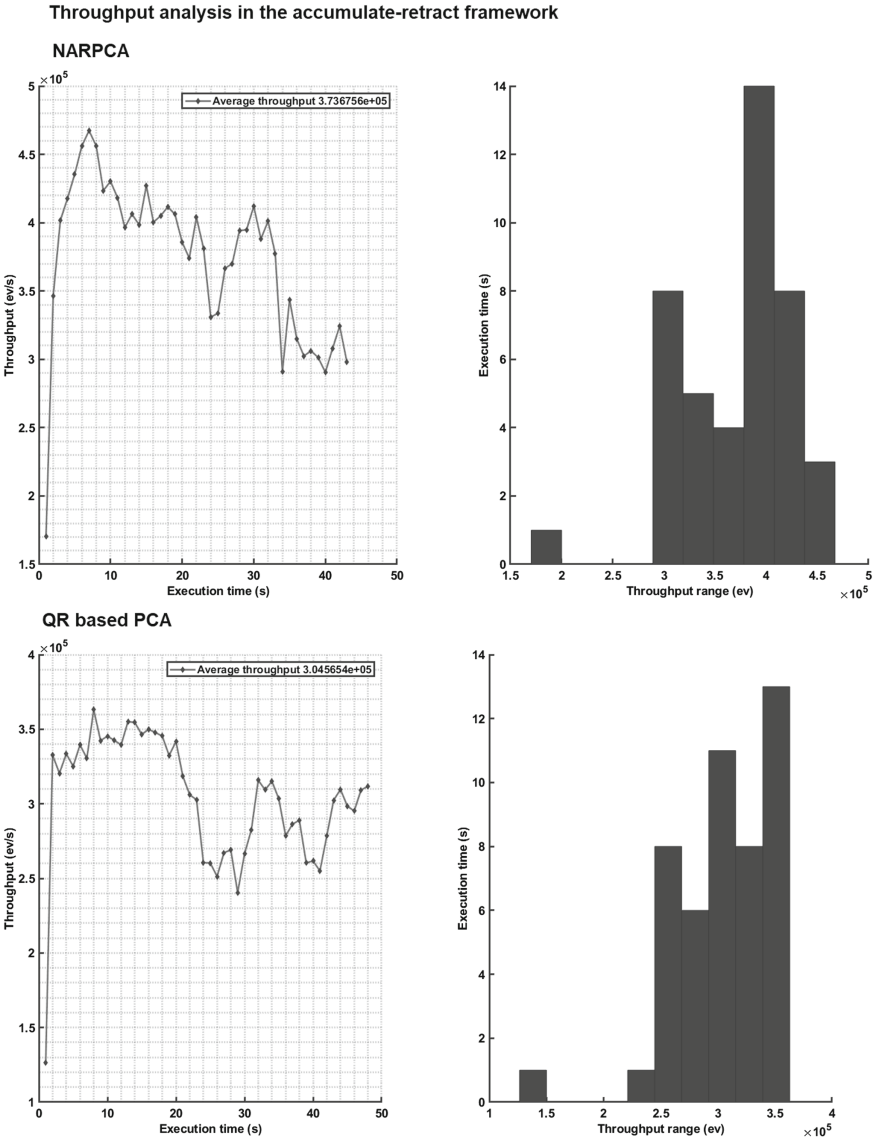

Fig. 7. Comparative analysis of throughput in the accumulate-retract framework.

Table 1. Eigenvalues and Eigenvectors estimates analysis

Eigenvalue	Eigenvalue estimate	Eigenvector variance
1	0.994071965	0.033719458
2	1.99658601	0.023661145
3	3.00600192	0.013884741
4	4.00420688	0.025106736
5	5.04173253	0.022354039
6	5.95475267	0.007637369
7	6.88985141	0.011129644
8	7.87972194	0.015864081
9	8.90795326	0.007244545
10	10.0642228	0.014663302

5 Conclusion

Tackling the theoretical bottlenecks in the traditional PCA algorithm and focusing on three critical quantities, namely latency, throughput and memory/storage, NARPCA supports the renewed interest and performance improvements in streaming machine learning. Traditional statistical packages require to have available prior to the calculation, a batch of examples from the distribution being investigated. While it is possible to run multiple flavours of PCA models with the accumulate-retract framework, neural approaches are capable of performing PCA in real- time, as our experiments show. The adaptive and incremental methodology used in neural approaches is particularly important if storage constraints are important, addressing one of the three critical quantities mentioned before. Strictly speaking, PCA is only defined for stationary distributions. However, in realistic situations, as streaming data, it is often the case that we are interested in compressing data from distributions which are a function of time; in this situation, the NARPCA is a good solution in that it tracks the moving statistics of the distribution and provides as close to batch PCA. Following a distribution's statistics, NARPCA is an example of trade-off between tracking capability and accuracy of convergence. Finally, NARPCA keeps memory usage fixed (i.e. only relevant "hot" data) and disk usage flexible (i.e. processed "cold" data) as the amount of events in the data stream increases. Such a system offers flexibility, allowing for arbitrary combinations of multiple functions (i.e. average, least squares regression, sorting) to be calculated on the stream, with no time and resource penalty, by exploiting the underlying hardware, data processing and data management for true low-latency, high-throughput stream processing.

References

1. Albert, B., Ricard Gavaldà, G.H., Pfahringer, B.: Machine learning for data streams with practical examples. In: MOA. MIT Press (2018)

2. Arora, R., Cotter, A., Livescu, K., Srebro, N.: Stochastic optimization for PCA and PLS. In: 2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 861–868. IEEE (2012)
3. Axenie, C., Tudoran, R., Bortoli, S., Hassan, M.A.H., Foroni, D., Brasche, G.: STARLORD: sliding window temporal accumulate-retract learning for online reasoning on datastreams. In: 17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018, Orlando, FL, USA, 17–20 December 2018, pp. 1115–1122 (2018)
4. Baker, C.G., Gallivan, K.A., Van Dooren, P.: Low-rank incremental methods for computing dominant singular subspaces. Linear Algebra Appl. **436**(8), 2866–2888 (2012)
5. Balsubramani, A., Dasgupta, S., Freund, Y.: The fast convergence of incremental PCA. In: Advances in Neural Information Processing Systems, pp. 3174–3182 (2013)
6. Boutsidis, C., Garber, D., Karnin, Z., Liberty, E.: Online principal components analysis. In: Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 887–901 (2015)
7. Brand, M.: Incremental singular value decomposition of uncertain data with missing values. In: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (eds.) ECCV 2002. LNCS, vol. 2350, pp. 707–720. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-47969-4_47
8. Bubeck, S., Cesa-Bianchi, N., et al.: Regret analysis of stochastic and nonstochastic multi-armed bandit problems. Found. Trends® in Mach. Learn. **5**(1), 1–122 (2012)
9. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., Tzoumas, K.: Apache flink™: Stream and batch processing in a single engine. IEEE Data Eng. Bull. **38**, 28–38 (2015). https://flink.apache.org/introduction.html
10. Chin, T.J., Suter, D.: Incremental kernel principal component analysis. IEEE Trans. Image Process. **16**(6), 1662–1674 (2007)
11. Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd edn. Johns Hopkins University Press, Baltimore, MD, USA (1996)
12. Hallgren, F., Northrop, P.: Incremental kernel PCA and the nyström method. arXiv preprint arXiv:1802.00043 (2018)
13. Householder, A.S.: Unitary triangularization of a nonsymmetric matrix. J. ACM **5**(4), 339–342 (1958). https://doi.org/10.1145/320941.320947
14. Jain, P., Jin, C., Kakade, S.M., Netrapalli, P., Sidford, A.: Streaming PCA: matching matrix Bernstein and near-optimal finite sample guarantees for Oja's algorithm. In: Conference on Learning Theory, pp. 1147–1164 (2016)
15. Kramer, M.A.: Nonlinear principal component analysis using autoassociative neural networks. AIChE J. **37**(2), 233–243 (1991)
16. Li, Y.: On incremental and robust subspace learning. Pattern Recogn. **37**(7), 1509–1518 (2004)
17. Lois, B., Vaswani, N.: A correctness result for online robust PCA. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 3791–3795. IEEE (2015)
18. Mitliagkas, I., Caramanis, C., Jain, P.: Memory limited, streaming PCA. In: Proceedings of the 26th International Conference on Neural Information Processing Systems, NIPS 2013, vol. 2, pp. 2886–2894. Curran Associates Inc., USA (2013)
19. Nadler, B.: Finite sample approximation results for principal component analysis: a matrix perturbation approach. Ann. Statist. **36**(6), 2791–2817 (2008)
20. Oja, E.: Simplified neuron model as a principal component analyzer. J. Math. Biol. **15**(3), 267–273 (1982)

21. Oja, E.: Principal components, minor components, and linear neural networks. Neural Netw. **5**(6), 927–935 (1992)
22. Qiu, J., Wang, H., Lu, J., Zhang, B., Du, K.L.: Neural network implementations for PCA and its extensions. In: 2012 ISRN Artificial Intelligence (2012)
23. Sanger, T.D.: Optimal unsupervised learning in a single-layer linear feedforward neural network. Neural Netw. **2**(6), 459–473 (1989). http://www.sciencedirect.com/science/article/pii/0893608089900440
24. Sarveniazi, A.: An actual survey of dimensionality reduction. Am. J. Comput. Matt. **4**(4), 55–72 (2014)
25. Shamir, O.: Convergence of stochastic gradient descent for PCA. In: International Conference on Machine Learning, pp. 257–265 (2016)
26. Sharma, A., Paliwal, K.K., Imoto, S., Miyano, S.: Principal component analysis using GR decomposition. Int. J. Mach. Learn. Cybern. **4**(6), 679–683 (2013). https://doi.org/10.1007/s13042-012-0131-7
27. Valpola, H.: From neural pca to deep unsupervised learning. In: Advances in Independent Component Analysis and Learning Machines, pp. 143–171. Elsevier (2015)
28. Vershynin, R.: How close is the sample covariance matrix to the actual covariance matrix? J. Theor. Probab. **25**(3), 655–686 (2012)
29. Warmuth, M.K., Kuzmin, D.: Randomized PCA algorithms with regret bounds that are logarithmic in the dimension. In: Advances in Neural Information Processing Systems, pp. 1481–1488 (2007)
30. Weng, J., Zhang, Y., Hwang, W.S.: Candid covariance-free incremental principal component analysis. IEEE Trans. Pattern Anal. Mach. Intell. **25**(8), 1034–1040 (2003)
31. Woo, S., Lee, C.: Incremental feature extraction based on decision boundaries. Pattern Recogn. **77**, 65–74 (2018). http://www.sciencedirect.com/science/article/pii/S003132031730496X
32. Xu, L., Oja, E., Suen, C.Y.: Modified hebbian learning for curve and surface fitting. Neural Netw. **5**(3), 441–457 (1992)
33. Yin, Y., Xu, D., Wang, X., Bai, M.: Online state-based structured SVM combined with incremental PCA for robust visual tracking. IEEE Trans. Cybern. **45**(9), 1988–2000 (2015)
34. Zhan, J., Lois, B., Guo, H., Vaswani, N.: Online (and offline) robust PCA: novel algorithms and performance guarantees. In: Artificial intelligence and statistics, pp. 1488–1496 (2016)
35. Zhao, F., Rekik, I., Lee, S.w., Liu, J., Zhang, J., Shen, D.: Two-phase incremental kernel pca for learning massive or online datasets. Complexity **2019** (2019)