

---

# Speeding up PCA with priming

---

Bálint Máté

Department of Computer Science  
University of Geneva  
1227 Carouge, Switzerland  
`balint.mate@unige.ch`

François Fleuret

Department of Computer Science  
University of Geneva  
1227 Carouge, Switzerland  
`francois.fleuret@unige.ch`

## Abstract

We introduce primed-PCA (pPCA), a two-step algorithm for speeding up the approximation of principal components. This algorithm first runs any approximate-PCA method to get an initial estimate of the principal components (priming), and then applies an exact PCA in the subspace they span. Since this subspace is of small dimension in any practical use, the second step is extremely cheap computationally. Nonetheless, it improves accuracy significantly for a given computational budget across datasets. In this setup, the purpose of the priming is to narrow down the search space, and prepare the data for the second step, an exact calculation. We show formally that pPCA improves upon the priming algorithm under very mild conditions, and we provide experimental validation on both synthetic and real large-scale datasets showing that it systematically translates to improved performance. In our experiments we prime pPCA by several approximate algorithms and report an average speedup by a factor of 7.2 over Oja’s rule, and a factor of 10.5 over EigenGame.

## 1 Introduction

Principal Component Analysis is a widely used tool both within and outside of computer science. Introduced more than a century ago by Karl Pearson [1901], it has been used for compression and feature extraction, and has led to many important works and variants [Turk and Pentland, 1991, Schölkopf et al., 1999].

Let  $X \in \mathbb{R}^{n \times d}$  be a centered dataset of cardinality  $n$  and dimension  $d$ . In terms of linear algebra, the principal directions are the eigenvectors of the covariance matrix  $X^T X$ . Unfortunately, if  $d$  and  $n$  are as high as in modern datasets, then traditional approaches, like computing the full-SVD of the covariance matrix gets computationally challenging. This has led to several attempts trying to circumvent this problem by using approximate/heuristic alternatives to full-SVD for finding the first few principal components of datasets of larger scale. Recently, Gemp et al. [2020] introduced the EigenGame-algorithm and managed to perform approximate PCA on datasets of dimension  $> 10M$ .

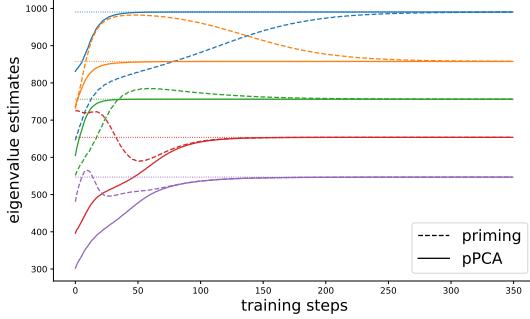
The contributions of this paper are:

In Section 3, we introduce a family of approximate-PCA algorithms that performs a one-time, cheap full-PCA step on the output of the priming algorithm.

In Section 4, we analyse this full-PCA step using elementary linear algebra and derive the mild theoretical conditions under which it improves the performance of the priming algorithm.

In Section 5, we study it empirically on several datasets by priming it by the power rule [Rutishauser, 1970], Oja’s algorithm [Oja, 1982] and EigenGame [Gemp et al., 2020] and demonstrate the improvement in accuracy and convergence speed. Whenever the size of the dataset allows we adopt the metric

of “Longest Correct Eigenvector Streak” of Gemp et al. [2020] that accounts for proper estimation of individual eigenvectors.




---

**Algorithm 1** Primed PCA (pPCA<sub>*l*</sub>)

---

**Input:** Dataset of dimension  $d$ ,  $\{\mathbf{x}_i\}_{i=1}^n$

**Parameter:**  $k > 0$ ,  $l > 0$ , PRIMING

**Output:** First  $k$  principal components

- 1:  $v_1, \dots, v_{k+l} \leftarrow \text{PRIMING}(\{\mathbf{x}_i\}_{i=1}^n, k+l)$
  - 2:  $V \leftarrow \text{span}(v_1, \dots, v_{k+l})$
  - 3:  $\tilde{\mathbf{x}}_i \leftarrow \text{proj}_V(\mathbf{x}_i)$
  - 4:  $e_1, \dots, e_k \leftarrow \text{full-PCA}(\{\tilde{\mathbf{x}}_i\}_{i=1}^n, k)$
  - 5: **return**  $e_1, \dots, e_k$
- 

Figure 1: (Left) The results of an example run of the proposed algorithm on the synthetic data with exponentially decaying spectrum. Thin, dotted lines denote the eigenvalues of the covariance matrix. Dashed lines denote the eigenvalue estimates of the priming algorithm, while solid lines denote eigenvalue estimates of pPCA. (Right) Pseudocode for primed-PCA.

## 2 Related Work

### 2.1 Power method

The power method, introduced by Rutishauser [1970], initialises a random vector  $x_0 \in \mathbb{R}^d$  of unit norm and iteratively computes  $x_{i+1} = \frac{Mx_i}{\|Mx_i\|}$  until the first eigendirection of  $M = X^T X$  dominates. The algorithm terminates when  $\|x_i - x_{i+1}\| < \epsilon$  for some small  $\epsilon$ . To capture multiple principal components, Sharma and Paliwal [2007] propose to simply repeat the above algorithm and orthogonalize after every training step to stay in the complement of the span of the higher order directions.

### 2.2 Oja’s rule

Oja’s learning rule [Oja, 1982] considers the output of a single neuron  $y = \mathbf{x}^T \mathbf{w} = \mathbf{w}^T \mathbf{x}$  with update rule  $\Delta \mathbf{w} = \alpha(\mathbf{x}y - y^2 \mathbf{w}) = \alpha(\mathbf{x}\mathbf{x}^T \mathbf{w} - (\mathbf{w}^T \mathbf{x})(\mathbf{x}^T \mathbf{w})\mathbf{w})$ . Averaging over all points  $C = \frac{1}{n} \sum_i \mathbf{x}_i \mathbf{x}_i^T$ , and plugging in the fixed-point condition ( $\Delta \mathbf{w} = 0$ ) one gets  $C\mathbf{w} - (\mathbf{w}^T C \mathbf{w})\mathbf{w} = 0$ , an eigenvector-eigenvalue equation for the covariance matrix  $C$ , i.e. the weights following this dynamics end up as the coordinates of principal directions. To find multiple principal components, the algorithm initializes  $w_1, \dots, w_m$  weight vectors and uses the update rule  $\Delta \mathbf{w}_m = \alpha y_m (\mathbf{x} - \sum_{l \leq m} y_l \mathbf{w}_l)$ . This generalized version of Oja’s rule is sometimes called Sanger’s rule [Oja, 1992, Sanger, 1989].

### 2.3 EigenGame

EigenGame [Gemp et al., 2020] interprets PCA in a game-theoretical framework, where vectors on the unit sphere  $S^{d-1} \subset \mathbb{R}^d$  correspond to strategies of players, playing the following multiplayer game. Let  $v_1, \dots, v_k \in \mathbb{R}^d$  ( $k < d$ ) denote the players. The utility function of the first player is

$$U_1(v_1) = v_1^T (X^T X) v_1 \quad (1)$$

i.e. player 1 is trying to tune the vector  $v_1 \in S^{d-1}$  to capture the maximum variance of the data. By definition,  $v_1$ ’s goal is to find the first principal component of  $X$ . To make  $v_2$  find the second principal component, the authors set the utility function of  $v_2$  to

$$U_2(v_2) = v_2^T (X^T X) v_2 - \frac{(v_2^T (X^T X) v_1)^2}{v_1^T (X^T X) v_1} \quad (2)$$

---

<sup>1</sup> $S^{d-1}$  denotes the (d-1) dimensional sphere, i.e. the  $d$ -dimensional vectors of unit norm.

where the first term rewards  $v_2$  if it finds a direction of high variance but the second term cancels the reward from its component parallel to  $v_1$ . Similarly, the utility function of  $v_j (1 < j \leq k)$  is

$$U_j(v_j) = v_j^T(X^T X)v_j - \sum_{1 \leq i < j} \frac{(v_j^T(X^T X)v_i)^2}{v_i^T(X^T X)v_i} \quad (3)$$

Finding the Nash-equilibrium of the game defined by the utility functions  $U_1, \dots, U_k$  is equivalent to finding the principal components of  $X$ .

This Gram-Schmidt-like setup defines a hierarchy between the players. Intuitively,  $v_1$  aims to maximise its variance without having to care about the other players, while all other players are also trying to maximise their variance, but have the additional constraint to stay orthogonal to the players with lower indices.

### 3 Priming PCA

#### 3.1 PRIMING

Our algorithm makes use of an already existing approximate PCA algorithm (priming), not necessarily one of those we mentioned above. To avoid the cumbersome "any approximate PCA algorithm" in the rest of the paper, let PRIMING be a placeholder for the priming algorithm to which any PCA algorithm can be assigned.

During the training process of PRIMING, it can happen that the exact principal directions  $e_1, \dots, e_k$  are not yet properly captured by the predicted directions  $v_1, \dots, v_k$ , but they already lie (up to some error term) in  $\text{span}(v_1, \dots, v_k)$ .

#### 3.2 PCA as post-processing

**Question.** Are  $v_1, \dots, v_k$  the best approximations of  $e_1, \dots, e_k$  in  $\text{span}(v_1, \dots, v_k)$ ?

Whenever the answer to this question is not affirmative, it makes sense to further optimize  $v_1, \dots, v_k$  within  $\text{span}(v_1, \dots, v_k)$ . Since usually  $k \ll d$ , full-PCA is feasible after projecting the data to this  $k$ -dimensional subspace. We term the resulting algorithm, the combination of PRIMING and full-PCA, primed-PCA (pPCA), and the algorithm for the initial approximation will be referred to as the priming algorithm or PRIMING.

We expect that the full-PCA will ease numerical issues and inaccuracies related to stochastic gradient descent and speed up convergence. In this paper we investigate how full-PCA improves the accuracy and speeds up PRIMING.

#### 3.3 Extra components

Knowing that after running PRIMING, we will post-process the output and not accept it as the final prediction of the principal components, allows us to modify PRIMING in a way that makes the job of full-PCA step easier.

For instance, if we are interested in the first  $k$  principal components of  $X$ , we could run PRIMING to search for  $k + l$  principal directions, project onto the  $(k + l)$ -dimensional  $\text{span}(v_1, \dots, v_k, \dots, v_{k+l})$  and do full-PCA to extract the first  $k$  components. Since the computational cost of running PRIMING (usually) scales quadratically in the number of directions,  $l$  has to be small. The question, of course, is whether or not the performance gained from a few additional components compensates for increased computational costs. In the rest of the paper, we refer to pPCA with  $l$  additional directions as pPCA <sub>$l$</sub> .

### 4 Theoretical analysis

It is intuitive to expect that the extra full-PCA step on the projected data should improve (or at least not hurt) the accuracy of the predicted principal directions. In this section, we analyse under what conditions this intuitive expectation holds up.

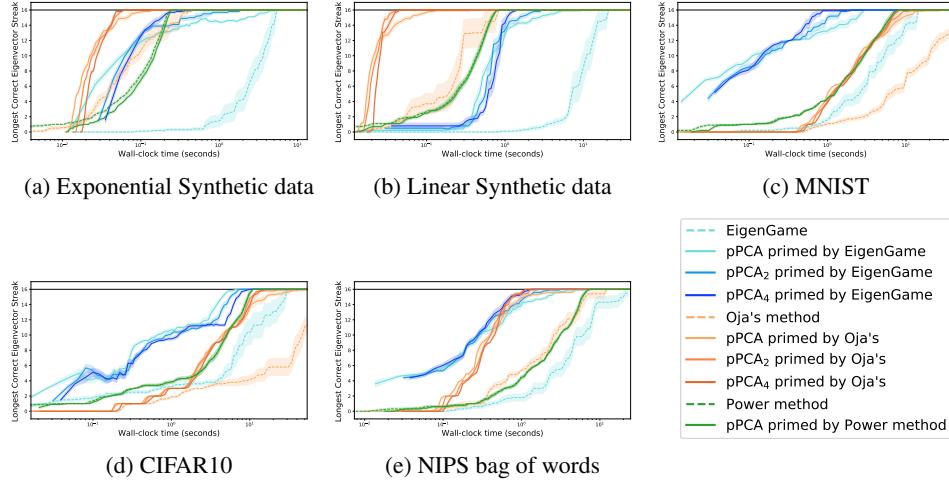


Figure 2: Plots showing the effect of the full-PCA step and the extra components on the small-scale experiments for  $V = \pi/8$ . Solid lines are pPCA runs, dashed lines denote the 3 baselines (Power Method, Oja’s, EigenGame). The pPCA curves have been primed the baseline of the same color. Shaded regions denote  $\pm$  standard error of the mean. Plots for  $V \in \{\pi/16, \dots, \pi/1024\}$  are available in the appendix.

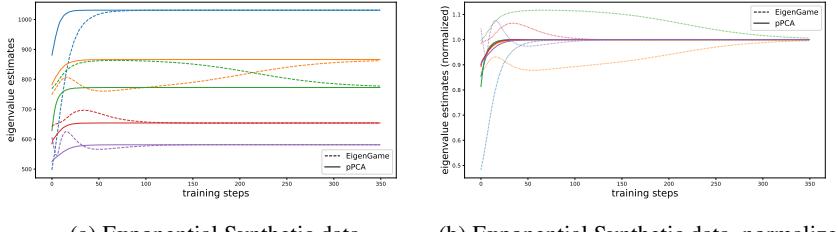


Figure 3: Figure (a) shows the estimates of the eigenvalues by pPCA and EigenGame on the Exponential Synthetic dataset with  $k = 5$ . Figure (b) is obtained by normalizing the curves of (a) by the average value of their respective final 5 datapoints.

#### 4.1 A counterexample

Consider the 3-dimensional, centered dataset of 6 elements,  $X = \{\pm 3(1, 0, 0), \pm 2(0, 1, 0), \pm (0, 0, 1)\}$ . Suppose we are interested in finding the first 2 principal components using the process described above. The covariance matrix is given by

$$\frac{1}{6} X^T X = \frac{1}{3} \begin{pmatrix} 9 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (4)$$

and the first two principal components are  $e_1 = (1, 0, 0)$ ,  $e_2 = (0, 1, 0)$ . Now let us suppose that PRIMING results in the vectors  $e_1^{\text{PRIMING}} = (\epsilon, 0, \sqrt{1 - \epsilon^2})$ ,  $e_2^{\text{PRIMING}} = (0, 1, 0)$ , for some small  $\epsilon \ll 1$ . That is, the second principal direction is perfectly recovered. On the other hand, the first principal component is estimated to be a linear combination of the first and third principal directions almost having no contribution from the first one.

Now, projecting the data onto  $\text{span}(e_1^{\text{PRIMING}}, e_2^{\text{PRIMING}})$  gives us

$$\tilde{X} = \{\pm 3\epsilon(\epsilon, 0, \sqrt{1 - \epsilon^2}), \pm 2(0, 1, 0), \pm \sqrt{1 - \epsilon^2}(\epsilon, 0, \sqrt{1 - \epsilon^2})\} \quad (5)$$

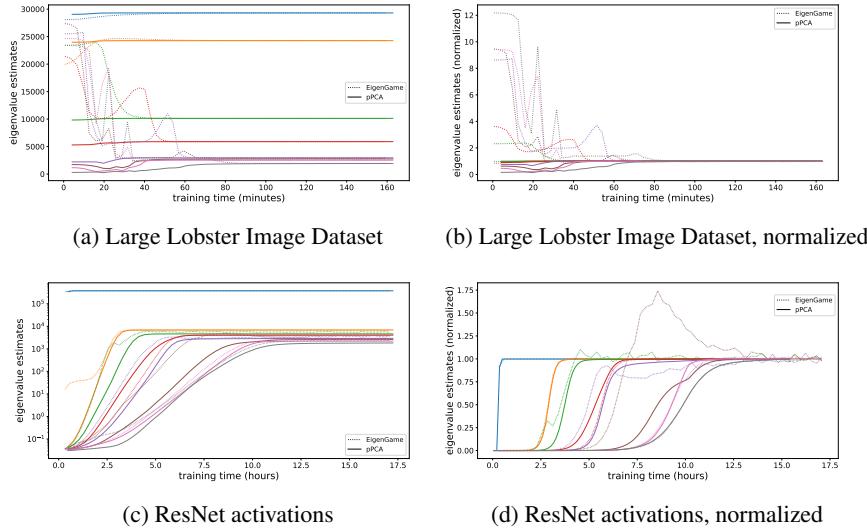


Figure 4: Plots showing the effect of the full-PCA on the large-scale experiments. Figures (a) and (c) show the estimates of the eigenvalues by pPCA and EigenGame. Figures (b) and (d) are obtained by normalizing the curves of (a) and (c).

Or, in the basis of  $\{e_1^{\text{PRIMING}}, e_2^{\text{PRIMING}}\}$ ,  $\tilde{X} = \{\pm 3\epsilon(1, 0), \pm 2(0, 1), \pm \sqrt{1 - \epsilon^2}(1, 0)\}$ . The covariance matrix of the projected data is then

$$\frac{1}{6}\tilde{X}^T\tilde{X} = \frac{1}{3} \begin{pmatrix} 9\epsilon^2 + 2(1 - \epsilon^2) & 0 \\ 0 & 4 \end{pmatrix} \quad (6)$$

If  $\epsilon$  is small enough, the full-PCA after the projection will predict the principal directions to be  $e_1^{pPCA} = e_2^{\text{PRIMING}}$  and  $e_2^{pPCA} = e_1^{\text{PRIMING}}$ . i.e it changes the ordering of the principal directions. Unfortunately,  $\langle e_1^{\text{PRIMING}}, e_1 \rangle^2 = \langle (\epsilon, 0, \sqrt{1 - \epsilon^2}), (1, 0, 0) \rangle^2 = \epsilon^2$  and  $\langle e_1^{pPCA}, e_1 \rangle^2 = \langle (0, 1, 0), (1, 0, 0) \rangle^2 = 0$ . In other words, we had a better approximation of the first principal direction before doing the full-PCA step.

It is important to note the absurdity of this example. The starting assumption that  $e_1^{\text{PRIMING}} = (\epsilon, 0, \sqrt{1 - \epsilon^2})$  would mean that PRIMING almost completely ignored the first two principal components and aligned itself with the third, smallest one. This, of course, can happen but it is not the expected behavior from an algorithm that is designed to find the principal components. The typical scenario is that PRIMING finds  $e_1$  up to some small error, and the remaining variance is picked up by the other players with lower indices. This is the case where the final full-PCA step will be useful.

Nonetheless, the above example demonstrates that we cannot just state that full-PCA step cannot decrease the accuracy of predicted principal components.

The reason why the above example fails is that the subspace spanned by the vectors  $e_i^{\text{PRIMING}}$  is almost orthogonal to the first principal direction. Projecting the data onto this subspace "forgets" that the original data has had high variance along  $e_1$  and  $e_1$  becomes impossible to recover because the lower principal components dominate the projected data.

## 4.2 When does the full-PCA step help?

In this section we derive the theoretical conditions under which the full-PCA step is useful. The main result of the section is Theorem 4.4 which roughly states if the priming algorithm converges then situations like the one in the Section 4.1 can only happen in the early phase of the priming algorithm where the eigenvector estimates are not yet "aligned enough" with the actual eigenvectors.

We begin the discussion with fixing the notation,

notation	meaning
$X, X^T X$	the dataset and its covariance matrix
$\{e_1, \dots, e_k\}$	the first $k$ principal directions of $X$ (eigenvectors of $X^T X$ )
$e_1^{\text{PRIMING}}, \dots, e_k^{\text{PRIMING}}$	PRIMING's estimates of the principal directions
$S, \pi_S$	$\text{span}(e_1^{\text{PRIMING}}, \dots, e_k^{\text{PRIMING}})$ , projection operator onto $S$
$\tilde{X}, \{\tilde{e}_1, \dots, \tilde{e}_k\}$	the projection of $X$ and $\{e_1, \dots, e_k\}$ onto $S$
$\mathbb{S}^{d-1}$	the $(d-1)$ dimensional sphere, vectors of unit norm

The variance of the data  $X$  in a direction  $v \in \mathbb{S}^{d-1}$  is  $\text{Var}_v(X) = v^T(X^T X)v$ . Write now  $\tilde{X}$  as the matrix product  $\tilde{X} = X P_S$ , where  $P_S$  is the projection matrix from  $\mathbb{R}^d$  onto  $S$ . The variance of  $\tilde{X}$  in a given direction  $v \in S \cap \mathbb{S}^{d-1}$  is then

$$\text{Var}_v(\tilde{X}) = v^T(\tilde{X}^T \tilde{X})v = v^T(P_S^T X^T X P_S)v = (P_S v)^T(X^T X)(P_S v) = v^T(X^T X)v$$

i.e. the quadratic form of the variance of  $\tilde{X}$  is just that of  $X$  after restricting its domain to  $S$ ,  $\text{Var}_{(\_)}\tilde{X}) = \text{Var}_{(\_)}(X)|_S$ .

**Proposition 4.1.** *If  $\pi_S(e_1)$  maximises  $\text{Var}_{(\_)}(X)$  on  $S$ , then the full-PCA step on  $S$  can not decrease the accuracy of the predicted first principal component.*

*Proof.* If  $\pi_S(e_1)$  maximises  $\text{Var}_{(\_)}(X)$  on  $S$ , then the full-PCA step on  $S$  returns  $\pi_S(e_1)$  as the first principal component. If PRIMING already output  $\pi_S(e_1)$  as the first principal component, then this has no effect on the accuracy, in all other cases, accuracy is improved.  $\square$

In essence, the only thing that can go wrong for the first principal component is that  $S$  is such that the variance of  $\tilde{X}$  in the direction of  $\pi_S(e_1)$  is smaller than in the direction of, say,  $\pi_S(e_2)$ .

**Proposition 4.2.** *If the assumption of Proposition 4.1 are satisfied, and  $\pi_S(e_2)$  maximises  $\text{Var}_{(\_)}(X)$  on  $\pi_S(e_1)^\perp$ , then the full-PCA step on  $S$  can not decrease the accuracy of the predicted first and second principal components.*

*Proof.* The proof is just a repeated use of Proposition 4.1. As the assumption of Proposition 4.1 is satisfied, full-PCA returns  $\pi_S(e_1)$  as the first principal component, not decreasing the accuracy of the prediction. Let us now consider the second principal direction. Since the eigenvectors of a covariance matrix are orthogonal, and  $\pi_S(e_2)$  maximises  $\text{Var}_{(\_)}(X)$  on  $\pi_S(e_1)^\perp$  by assumption, the repeated use of Proposition 4.1 on  $\pi_S(e_1)^\perp$  implies that the full-PCA step outputs  $\pi_S(e_2)$  as the second eigenvector, not decreasing the accuracy of the prediction.  $\square$

Continuing this line of reasoning, always requiring the projection of the next principal component to maximise the variance on the orthogonal complement of the previous eigenvectors, we arrive at following sequence of conditions

**Proposition 4.3.** *If  $V$  is such that*

- $\pi_S(e_1)$  maximises  $\text{Var}_{(\_)}(X)$  on  $S$ ,
- $\pi_S(e_2)$  maximises  $\text{Var}_{(\_)}(X)$  on  $\pi_S(e_1)^\perp$ ,
- ...
- $\pi_S(e_i)$  maximises  $\text{Var}_{(\_)}(X)$  on the orthogonal complement of  $\text{span}(\pi_S(e_1), \dots, \pi_S(e_{i-1}))$

then the full-PCA step on  $S$  can not decrease the accuracy of the predicted 1<sup>st</sup>, 2<sup>nd</sup>, ...,  $i^{\text{th}}$  principal components.

*Proof.* This is again just a repeated application of Proposition 4.1 as in the proof of Proposition 4.2.  $\square$

It is important to note that even though the conditions of Propositions 4.3 look cumbersome, the outputs of every approximate PCA algorithm satisfies them if trained long enough.

**Theorem 4.4.** Let  $e_i^{\text{PRIMING}}(\tau)$  denote the output of PRIMING at timestep  $\tau$ . If PRIMING converges, i.e. if

$$\lim_{\tau \rightarrow \infty} e_i^{\text{PRIMING}}(\tau) \rightarrow e_i \quad \text{for all } 0 < i \leq k \quad (7)$$

then there exists a  $\tau^*$  such that if  $\tau > \tau^*$  then the conditions of Proposition 4.3 are satisfied and applying the full-PCA step on  $\{e_1^{\text{PRIMING}}(\tau), \dots, e_k^{\text{PRIMING}}(\tau)\}$  improves performance with respect to the metric of the longest correct eigenvector streak for any  $V$ .

*Proof.* If  $e_i^{\text{PRIMING}}(\tau) = e_i$  for all  $i$ , i.e. if PRIMING already found the eigenvectors then the implication of the theorem obviously holds. As the conditions of 4.3 are an open condition (only involving inequalities), they are satisfied on an open set. This means that there exists an open neighborhood  $U$  of  $\{e_1, \dots, e_k\}$  on which these conditions of 4.3 are satisfied. Since

$$\lim_{\tau \rightarrow \infty} e_i^{\text{PRIMING}}(\tau) \rightarrow e_i \quad \text{for all } 0 < i \leq k$$

The vectors  $\{e_1^{\text{PRIMING}}(\tau), \dots, e_k^{\text{PRIMING}}(\tau)\}$  enter and stay in  $U$  if  $\tau$  is large enough.  $\square$

## 5 Experiments and results

We test the proposed algorithm on several dataset of varying size. Here we present them in order of increasing complexity. Doing full-PCA in the original data space is doable for the small-scale datasets (synthetic, MNIST, CIFAR10 and “NIPS bag of words”), but infeasible for the Large Lobster Image Dataset and ResNet activations.

We train with SGD using Nesterov momentum with a factor of 0.9 [Nesterov, 1983]. Everything is implemented in Pytorch, the experiments on the small-scale datasets are executed on a NVIDIA RTX 3090 while the experiments on the large-scale datasets are executed on an NVIDIA A100.

	Exponential Synthetic	Linear Synthetic	MNIST	CIFAR10	NIPS bag of words
dimensions	50	50	768	3092	11 463
points	5000	5000	60 000	50 000	5812
$V = \pi/8$					
EigenGame	$4.55 \pm 0.67$	$12.24 \pm 4.14$	$9.33 \pm 3.64$	$15.79 \pm 8.33$	$7.54 \pm 2.19$
+ pPCA (Ours)	$3.48 \pm 0.87$	$4.04 \pm 2.30$	<b><math>2.21 \pm 0.59</math></b>	<b><math>5.02 \pm 0.73</math></b>	<b><math>3.81 \pm 1.37</math></b>
Oja’s method	$0.23 \pm 0.09$	$0.32 \pm 0.18$	n.a.	n.a.	$5.67 \pm 2.15$
+ pPCA (Ours)	<b><math>0.08 \pm 0.03</math></b>	<b><math>0.11 \pm 0.09</math></b>	$11.53 \pm 5.74$	n.a.	$2.68 \pm 1.71$
Power method	$0.21 \pm 0.01$	$0.67 \pm 0.07$	$6.32 \pm 0.80$	$9.63 \pm 1.13$	$4.47 \pm 0.38$
+ pPCA (Ours)	$0.23 \pm 0.01$	$0.68 \pm 0.07$	$6.33 \pm 0.80$	$9.63 \pm 1.13$	$4.48 \pm 0.38$
$V = \pi/32$					
EigenGame	$5.81 \pm 0.83$	n.a.	n.a.	$28.37 \pm 10.09$	n.a.
+ pPCA (Ours)	$4.58 \pm 0.95$	$7.45 \pm 3.00$	<b><math>3.44 \pm 0.57</math></b>	<b><math>6.92 \pm 0.79</math></b>	$6.57 \pm 2.31$
Oja’s method	$0.34 \pm 0.09$	$0.58 \pm 0.24$	n.a.	n.a.	$10.03 \pm 2.98$
+ pPCA (Ours)	<b><math>0.15 \pm 0.05</math></b>	<b><math>0.19 \pm 0.10</math></b>	$19.74 \pm 7.31$	n.a.	$4.54 \pm 1.98$
Power method	$0.22 \pm 0.01$	n.a.	$6.38 \pm 0.80$	$9.73 \pm 1.13$	<b><math>4.52 \pm 0.38</math></b>
+ pPCA (Ours)	$0.23 \pm 0.01$	$0.69 \pm 0.07$	$6.38 \pm 0.80$	$9.74 \pm 1.13$	$4.54 \pm 0.38$

Table 1: Time to reach an Eigenvalues streak of length 16 with different threshold values  $V \in \{\pi/8, \pi/32\}$  for the algorithms on the small-scale datasets. All experiments were repeated 10 times, mean and variance values are reported. All values have units of seconds, if one the 10 runs did not reach a streak of length 16 during training, then “n.a.” is reported. A more detailed table including PCA<sub>2</sub> and pPCA<sub>4</sub> can be found in the appendix.

## 5.1 Small-scale Datasets

We run EigenGame for each of the datasets 10 times with learning rates  $\{1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$ . We then choose the learning rate with the smallest angular error when summing over runs, timesteps and principal components. The pPCA methods primed by EigenGame are then executed also 10 times with the learning rate that best fitted EigenGame. Similarly, we run Oja’s algorithm for each of the datasets 10 times with learning rates  $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$ . We then choose the learning rate with the smallest angular error when summing over runs, timesteps and principal components. The pPCA methods primed by Oja’s method are then executed with this learning rate. We run the power method with termination conditions  $\epsilon \in \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}\}$ , choose the best  $\epsilon$  and run pPCA on top of it. Due to the sequential nature of the power method, we don’t expect much improvement from the full-PCA step in this setup. In all these experiments, we are interested in finding the first 16 principal components.

**Synthetic data** We generate synthetic datasets along the lines of the synthetic experiments of the EigenGame paper [Gemp et al., 2020]. The data consists of 5000 points in 50 dimensions with a spectrum that decays exponentially (resp. linearly) over 3 orders of magnitudes, from 1000 to 1. We train with a batch size of 1000.

**MNIST and CIFAR10.** We flatten the training images of MNIST [LeCun and Cortes, 2010] and CIFAR10 [Krizhevsky, 2009] that results in a dataset of cardinality 60 000 (resp. 50 000) and dimensionality 784 (resp. 3092). We use a batch size of 1000.

**NIPS bag of words** This dataset contains the frequency of 11 463 words in 5812 NIPS papers published between 1987 and 2015 [Perrone et al., 2016]. Each entry in the 11 463-by-5812 matrix holds the number of occurrences of a given word in the corresponding paper. We train with a batch size of 1000.

## 5.2 Large-scale Datasets

For the large-scale experiments we only train EigenGame and run pPCA on top of it (without additional components).

**ResNet activations** Following the original EigenGame paper, we build a dataset from the activation patterns of a pretrained ResNet-152 [He et al., 2015] evaluated on the validation set of ImageNet [Deng et al., 2009]. Each datapoint is constructed from the outputs of all residual blocks of the ResNet-152. This results in a dataset of dimension  $d \sim 13.1M$  and cardinality  $n = 50000$ . We use a batch size of 1024 and train for 600 training steps ( $\sim 12$  epochs) with a learning rate of  $10^{-6}$  to find the first 8 principal components. The time-cost of the full-PCA step (in particular, of the projection) is comparable to the time cost of 1 training epoch. In this experiment, we only train for 12 epochs, therefore the the cost of full-PCA is not negligible. Circumventing this issue, we only project 10% of the data for computing full-PCA, reducing the overhead for full-PCA from  $\sim 95$  to  $\sim 9.5$  minutes.

**Large Lobster Image Dataset** The Large Lobster Image Dataset [Vo et al., 2020] contains 6654 images of 238 southern rock lobsters taken at a lobster processor in Tasmania over the course of 6 days. The images have a resolution of  $1944 \times 2592$  and 3 color channels. After flattening, we end up with  $d \sim 15.6M$  dimensional vectors. See Figure 5 for examples. We use a batch size of 1024 and train for 300 training steps ( $\sim 43$  epochs) with a learning rate of  $10^{-4}$  to find the first 8 principal components.

## 5.3 Measure of performance

**Small-scale** To evaluate the small-scale experiments we adopt the demanding metric of “Longest Correct Eigenvector Streak” of Gemp et al. [2020] which reflects the hierarchy of principal directions. Given the ground truth principal directions  $\{e_1, \dots, e_k\}$ , the approximate directions coming from the learning algorithm  $\{v_1, \dots, v_k\}$  and a threshold value  $0 < V < \pi/2$ , we compute if the angle between  $v_i$  and  $e_i$  is smaller than  $V$  for all  $i \in \{1, \dots, k\}$ . Then, the number of consecutive pairs from index 1 that are within angle  $V$  of each other is by definition the Longest Correct Eigenvector Streak. For



(a) Lobster #2 on day 2 (b) Lobster #17 on day 5 (c) Lobster #220 on day 6 (d) Average of the dataset

Figure 5: The Large Lobster Image Dataset

instance, if for a given  $V$  the angles are  $[V/2, V/3, 2V, V/2, \dots]$ , then the Longest Correct Eigenvector Streak for this  $V$  is 2. In our experiments, we evaluate all runs with  $V \in \{\pi/8, \pi/16, \dots, \pi/1024\}$ .

**Large-scale** The remaining 2 experiments (Lobsters and ResNet activations) need to evaluated differently, since the real principal directions are not available. This means that we can only evaluate performance indirectly. We plot the variance<sup>2</sup> captured by the result of EigenGame and pPCA. When the algorithm converges to the actual principal directions of the data, the variance will converge to the corresponding eigenvalue of the covariance matrix. Convergence of the variance is therefore a hint (but not evidence!) of convergence of the corresponding eigenvalues/principal directions.

#### 5.4 Results

**Small-scale** Figure 2 shows the plots of the Longest Correct Eigenvector Streak for the small-scale experiments described above. We reuse the threshold from the EigenGame paper [Gemp et al., 2020] and set it to  $V = \pi/8$  for the plots of Figure 2. In the Appendix, plots are available for all threshold values  $V \in \{\pi/16, \dots, \pi/1024\}$ . A qualitative comparison is available in Table 1.

**Large-scale** Figure 4 displays the results for the large-scale experiments. Figure 6 in the Appendix shows the eigenvectors found by both methods after 30 and 300 training steps of the Large Scale Lobster Experiment. After 300 training steps the methods converge to vectors that are indistinguishable by the human eye. Looking at the results after 30 training steps, we can also conclude the pPCA already found the first 4 components, while EigenGame only the first 2. Note that this is exactly what Figure 4 suggests.

## 6 Conclusion

We introduced primed-PCA, a method for computing the first  $k$  principal directions by combining an already existing algorithm (called the priming) with a cheap, post-processing full-PCA step. We have demonstrated on several datasets that pPCA greatly improves upon the priming in terms of convergence speed and worked out the algebraic condition that guarantees that in most cases this is expected to happen. We have also demonstrated on small-scale datasets that using extra components improves performance further when we use Oja’s algorithm or EigenGame for priming.

## Future Work

As discussed in Subsection 3.3, the fact that after running the priming we will post-process the results allows for modifications of the priming algorithm. Since none of the existing approximate-PCA was not designed to be followed by a post-processing step, it is possible that their current form is not the optimal preprocessing step for the full-PCA. It could also be investigated how the optimal value of additional players,  $l$ , changes as the parameters of the dataset,  $d$  and  $n$ , or the number of principal components of interest,  $k$ , vary.

---

<sup>2</sup>The variance captured by a vector  $v$  is  $v^T(X^T X)v$

## 7 Funding disclosure

Bálint Máté was supported by the Swiss National Science Foundation under grant number FNS-193716 “Robust Deep Density Models for High-Energy Particle Physics and Solar Flare Analysis (RODEM)”.

## References

- Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning Big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering, 2018.
- Ian Gemp, Brian McWilliams, Claire Vernade, and Thore Graepel. Eigengame: Pca as a nash equilibrium. *arXiv preprint arXiv:2010.00554*, 2020.
- Ian Gemp, Brian McWilliams, Claire Vernade, and Thore Graepel. EigenGame Unloaded: When playing games is better than optimizing, 2021.
- Mina Ghashami, Edo Liberty, Jeff M. Phillips, and David P. Woodruff. Frequent Directions : Simple and Deterministic Matrix Sketching. *CoRR*, abs/1501.01711, 2015. URL <http://arxiv.org/abs/1501.01711>.
- Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL <https://doi.org/10.1038/s41586-020-2649-2>.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, 2015.
- F.R.S. Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. doi: 10.1080/14786440109462720.
- Alex Krizhevsky. Learning multiple layers of features from tiny images. Master’s thesis, Department of Computer Science, University of Toronto, 2009.
- Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL <http://yann.lecun.com/exdb/mnist/>.
- Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence  $\mathcal{O}(1/k^2)$ . 1983.
- Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, November 1982. ISSN 0303-6812. doi: 10.1007/BF00275687. URL <http://dx.doi.org/10.1007/BF00275687>.
- Erkki Oja. Principal components, minor components, and linear neural networks. *Neural Networks*, 5:927–935, 1992.
- Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.
- Valerio Perrone, Paul A. Jenkins, Dario Spano, and Yee Whye Teh. Poisson Random Fields for Dynamic Feature Models, 2016.

- H. Rutishauser. Simultaneous iteration method for symmetric matrices. *Numerische Mathematik*, 16: 205–223, 1970.
- Terence Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2:459–473, 12 1989. doi: 10.1016/0893-6080(89)90044-0.
- Bernhard Schölkopf, Alexander J. Smola, and Klaus-Robert Müller. *Kernel Principal Component Analysis*, page 327–352. MIT Press, Cambridge, MA, USA, 1999. ISBN 0262194163.
- Alok Sharma and Kuldip Paliwal. Fast principal component analysis using fixed-point algorithm. *Pattern Recognition Letters*, 28:1151–1155, 07 2007. doi: 10.1016/j.patrec.2007.01.012.
- Cheng Tang. Exponentially convergent stochastic k-PCA without variance reduction. *CoRR*, abs/1904.01750, 2019. URL <http://arxiv.org/abs/1904.01750>.
- M.A. Turk and A.P. Pentland. Face recognition using eigenfaces. In *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–591, 1991. doi: 10.1109/CVPR.1991.139758.
- Son Anh Vo, Joel Scanlan, Paul Turner, and Robert Ollington. Convolutional Neural Networks for individual identification in the Southern Rock Lobster supply chain. *Food Control*, 118: 107419, 2020. ISSN 0956-7135. doi: <https://doi.org/10.1016/j.foodcont.2020.107419>. URL <https://www.sciencedirect.com/science/article/pii/S0956713520303352>.

## A Additional numerical results

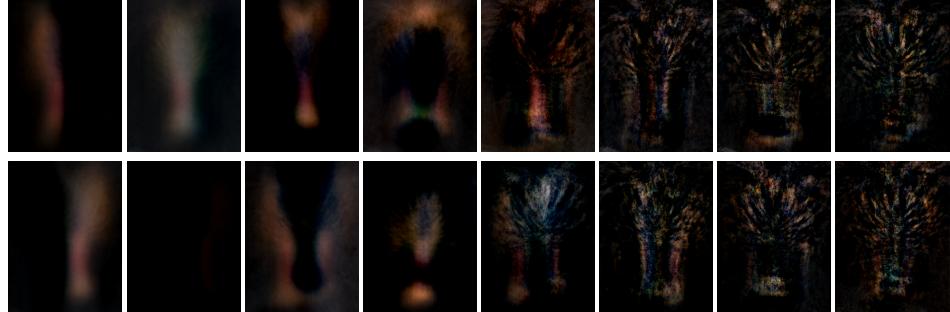
	Exponential Synthetic	Linear Synthetic	MNIST	CIFAR10	NIPS bag of words
dimensions	50	50	768	3092	11 463
points	5000	5000	60 000	50 000	5812
$V = \pi/8$					
EigenGame	4.55 ± 0.67	12.24 ± 4.14	9.33 ± 3.64	15.79 ± 8.33	7.54 ± 2.19
+ pPCA (Ours)	3.48 ± 0.87	4.04 ± 2.30	2.21 ± 0.59	5.02 ± 0.73	3.81 ± 1.37
+ pPCA <sub>2</sub> (Ours)	0.89 ± 0.58	1.64 ± 0.62	1.15 ± 0.37	3.17 ± 1.54	1.66 ± 0.54
+ pPCA <sub>4</sub> (Ours)	0.27 ± 0.11	1.10 ± 0.20	<b>0.78 ± 0.43</b>	<b>2.45 ± 0.91</b>	1.24 ± 0.62
Oja's method	0.23 ± 0.09	0.32 ± 0.18	n.a.	n.a.	5.67 ± 2.15
+ pPCA (Ours)	0.08 ± 0.03	0.11 ± 0.09	11.53 ± 5.74	n.a.	2.68 ± 1.71
+ pPCA <sub>2</sub> (Ours)	<b>0.05 ± 0.01</b>	<b>0.03 ± 0.01</b>	8.03 ± 3.32	14.38 ± 7.24	1.07 ± 0.30
+ pPCA <sub>4</sub> (Ours)	<b>0.05 ± 0.01</b>	0.04 ± 0.00	6.83 ± 0.86	11.58 ± 2.02	<b>0.91 ± 0.14</b>
Power method	0.21 ± 0.01	0.67 ± 0.07	6.32 ± 0.80	9.63 ± 1.13	4.47 ± 0.38
+ pPCA (Ours)	0.23 ± 0.01	0.68 ± 0.07	6.33 ± 0.80	9.63 ± 1.13	4.48 ± 0.38
$V = \pi/32$					
EigenGame	5.81 ± 0.83	n.a.	n.a.	28.37 ± 10.09	n.a.
+ pPCA (Ours)	4.58 ± 0.95	7.45 ± 3.00	3.44 ± 0.57	6.92 ± 0.79	6.57 ± 2.31
+ pPCA <sub>2</sub> (Ours)	2.35 ± 0.61	2.92 ± 0.93	2.18 ± 0.57	5.09 ± 1.42	3.18 ± 0.38
+ pPCA <sub>4</sub> (Ours)	0.80 ± 0.34	1.94 ± 0.44	<b>1.58 ± 0.57</b>	<b>4.61 ± 1.10</b>	2.59 ± 0.59
Oja's method	0.34 ± 0.09	0.58 ± 0.24	n.a.	n.a.	10.03 ± 2.98
+ pPCA (Ours)	0.15 ± 0.05	0.19 ± 0.10	19.74 ± 7.31	n.a.	4.54 ± 1.98
+ pPCA <sub>2</sub> (Ours)	0.09 ± 0.02	<b>0.05 ± 0.01</b>	13.07 ± 4.45	26.32 ± 9.60	1.90 ± 0.43
+ pPCA <sub>4</sub> (Ours)	<b>0.07 ± 0.01</b>	<b>0.05 ± 0.01</b>	11.96 ± 1.44	18.97 ± 3.80	<b>1.61 ± 0.21</b>
Power method	0.22 ± 0.01	n.a.	6.38 ± 0.80	9.73 ± 1.13	4.52 ± 0.38
+ pPCA (Ours)	0.23 ± 0.01	0.69 ± 0.07	6.38 ± 0.80	9.74 ± 1.13	4.54 ± 0.38
$V = \pi/128$					
EigenGame	n.a.	n.a.	n.a.	n.a.	n.a.
+ pPCA (Ours)	n.a.	n.a.	n.a.	n.a.	n.a.
+ pPCA <sub>2</sub> (Ours)	3.05 ± 0.71	4.84 ± 1.57	n.a.	n.a.	n.a.
+ pPCA <sub>4</sub> (Ours)	1.60 ± 0.35	2.93 ± 0.63	n.a.	n.a.	n.a.
Oja's method	0.46 ± 0.10	0.92 ± 0.29	n.a.	n.a.	n.a.
+ pPCA (Ours)	0.23 ± 0.06	0.29 ± 0.11	29.88 ± 8.00	n.a.	n.a.
+ pPCA <sub>2</sub> (Ours)	0.14 ± 0.03	0.07 ± 0.02	19.45 ± 4.69	40.01 ± 11.90	n.a.
+ pPCA <sub>4</sub> (Ours)	<b>0.09 ± 0.01</b>	<b>0.06 ± 0.01</b>	17.60 ± 1.71	27.88 ± 4.82	n.a.
Power method	0.22 ± 0.01	n.a.	n.a.	n.a.	n.a.
+ pPCA (Ours)	0.23 ± 0.01	0.69 ± 0.07	<b>6.46 ± 0.80</b>	<b>9.80 ± 0.94</b>	<b>6.39 ± 0.58</b>

Table 2: Time to reach an Eigenvalues streak of length 16 with different threshold values  $V \in \{\pi/8, \pi/32, \pi/128\}$  for the algorithms on the small-scale datasets. All experiments were repeated 10 times, mean and variance values are reported. All values have units of seconds, if one the 10 runs did not reach a streak of length 16 during training, then “n.a.” is reported.

## **B Additional plots**



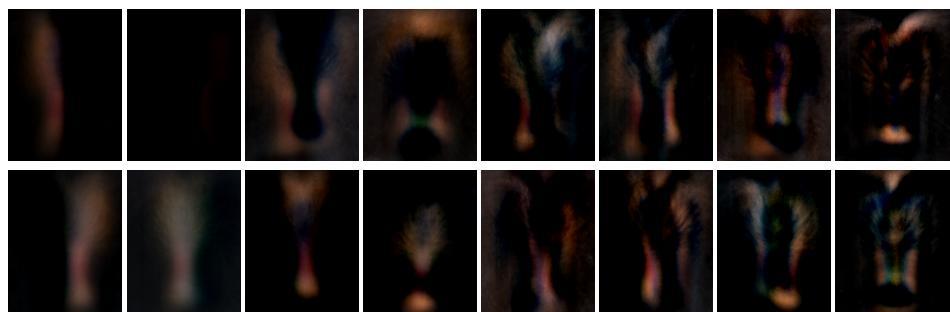
(a) EigenGame players after 30 training steps



(b) Principal directions predicted by pPCA after 30 training steps



(c) EigenGame players after 300 training steps



(d) Principal directions predicted by pPCA after 300 training steps

Figure 6: Each of the subplots show the first 8 principal components of the Large Lobster Image Dataset organised into 8 columns found by the respective algorithms. The two lines show the positive and negative parts of the vectors. After 30 training steps pPCA already finds the first 4 vectors, while EigenGame only predicts the first 2 correctly. After 300 training steps, the methods have the same output for all 8 principal directions.

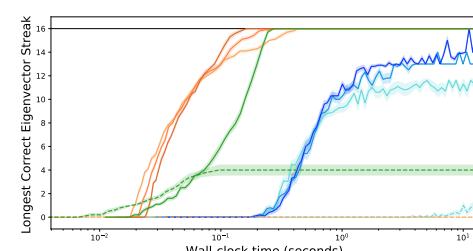
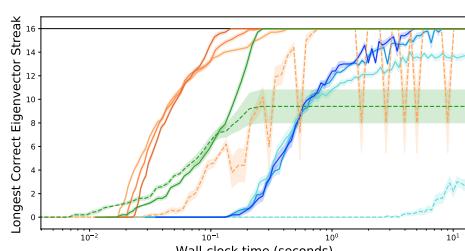
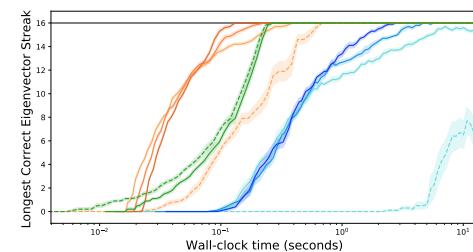
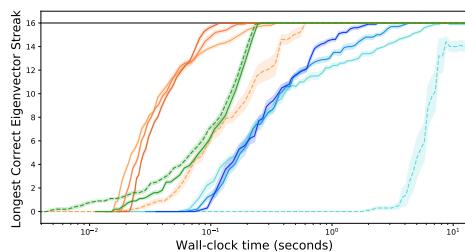
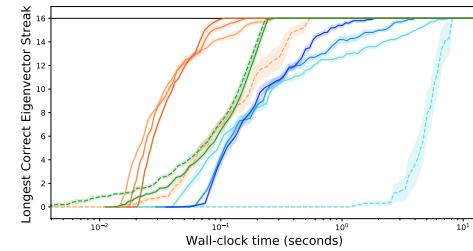
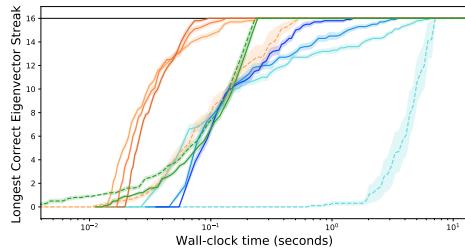
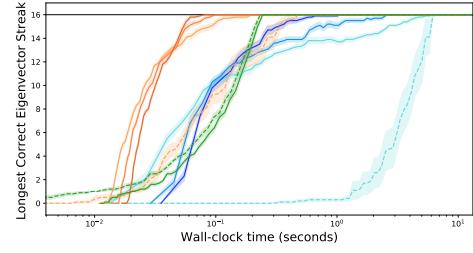
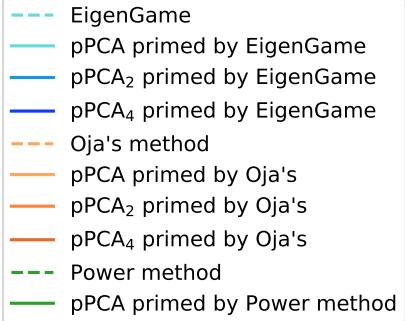


Figure 7: Synthetic data with exponential spectrum

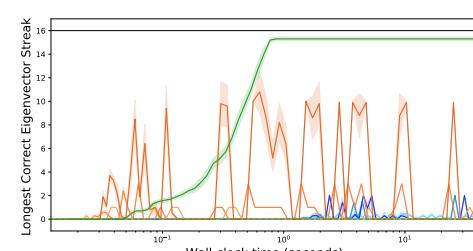
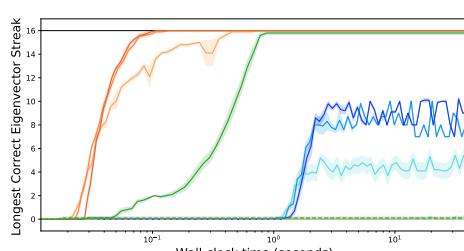
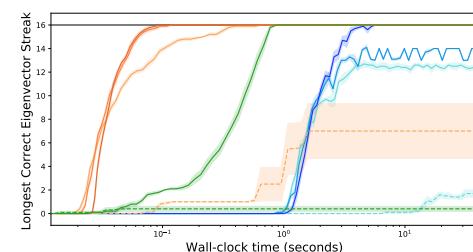
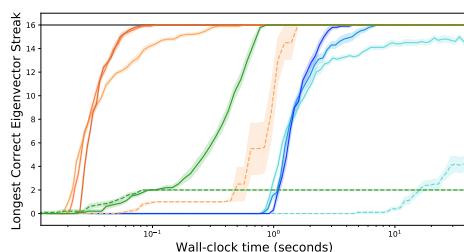
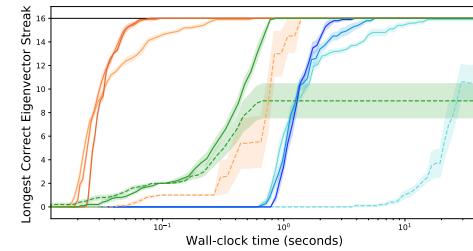
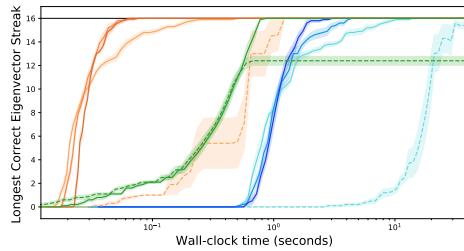
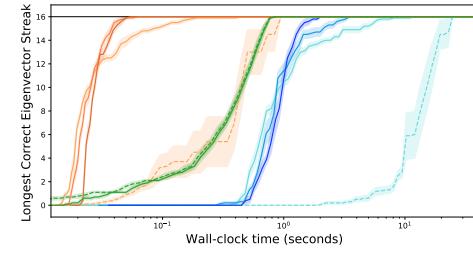
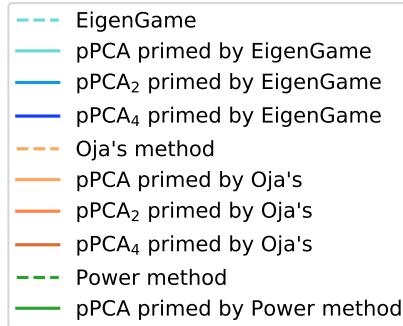
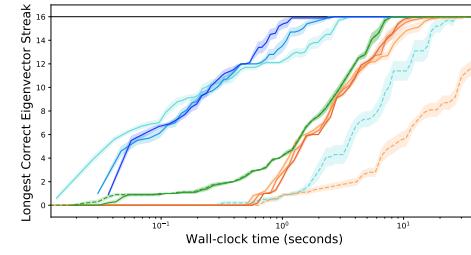
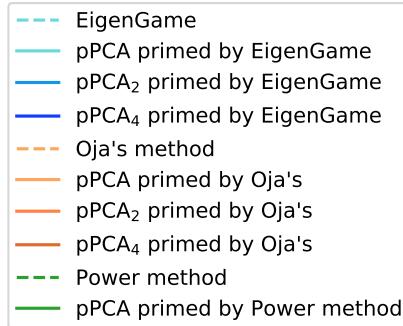
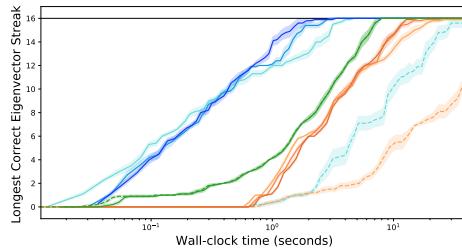


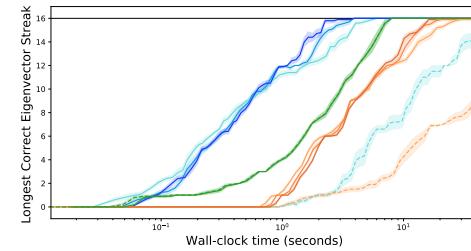
Figure 8: Synthetic data with linear spectrum



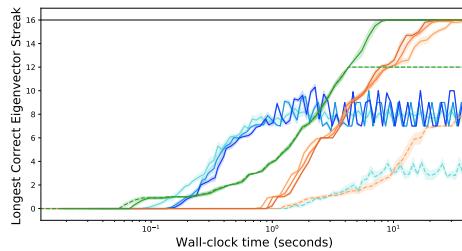
(a)  $V = \pi/16$



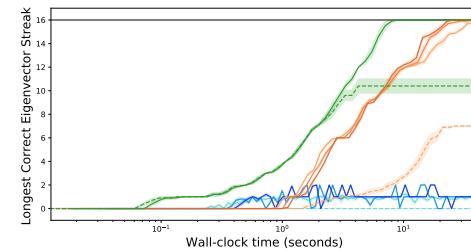
(b)  $V = \pi/32$



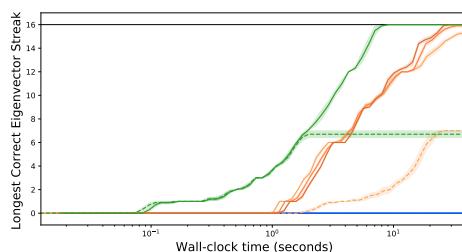
(c)  $V = \pi/64$



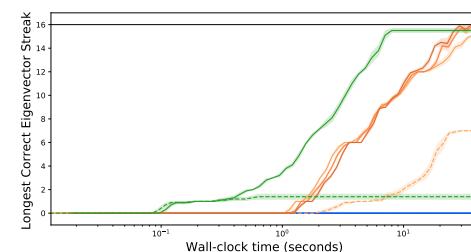
(d)  $V = \pi/128$



(e)  $V = \pi/256$



(f)  $V = \pi/512$



(g)  $V = \pi/1024$

Figure 9: MNIST

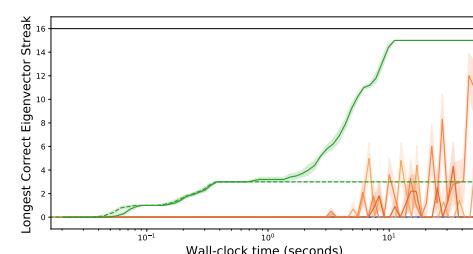
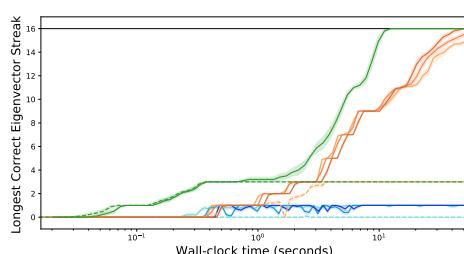
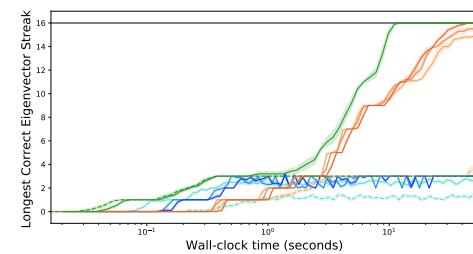
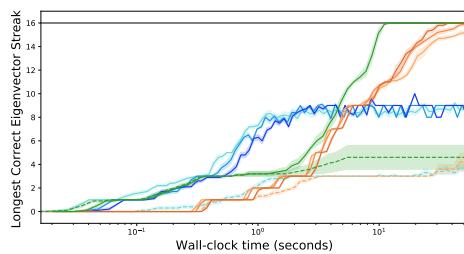
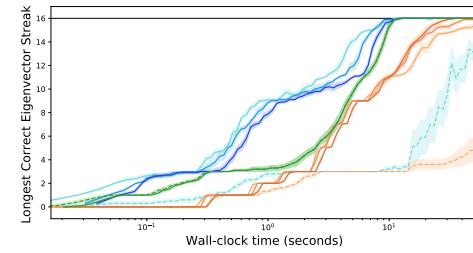
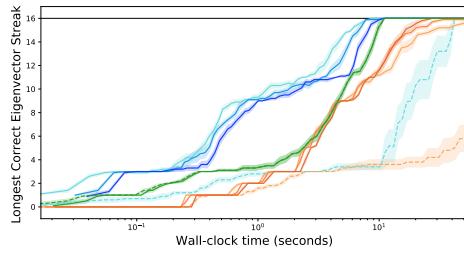
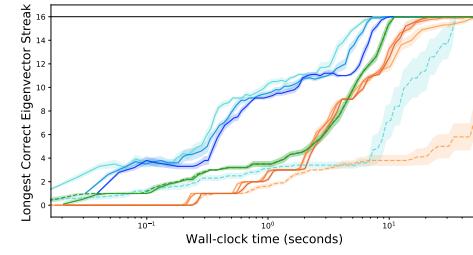
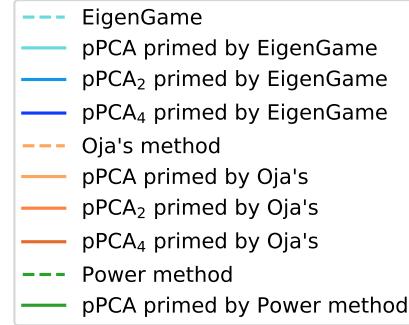
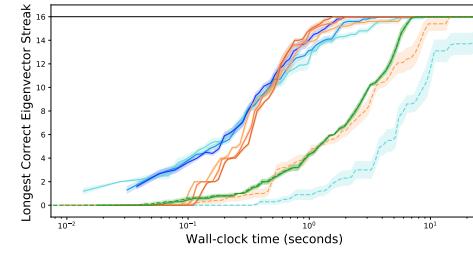
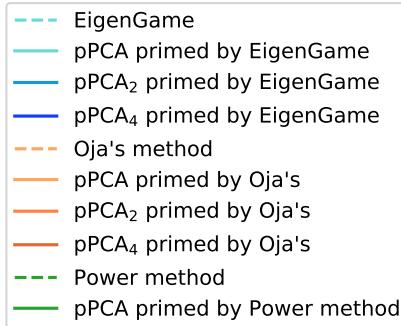
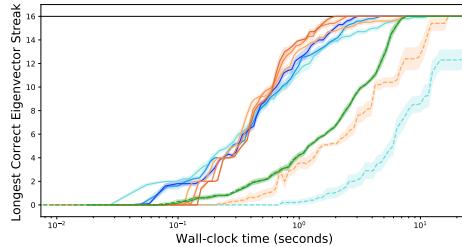


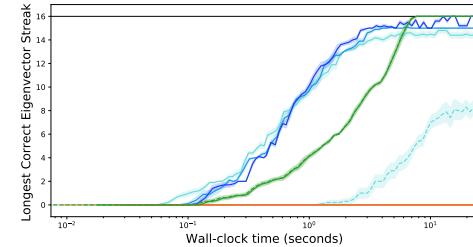
Figure 10: CIFAR10



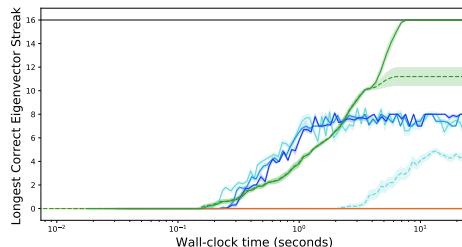
(a)  $V = \pi/16$



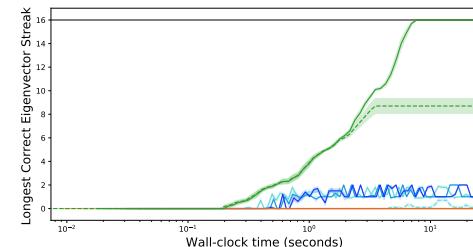
(b)  $V = \pi/32$



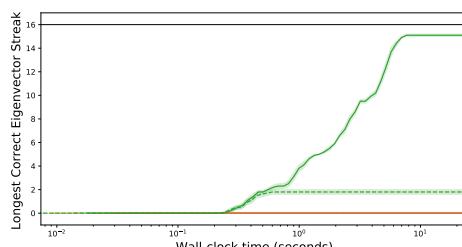
(c)  $V = \pi/64$



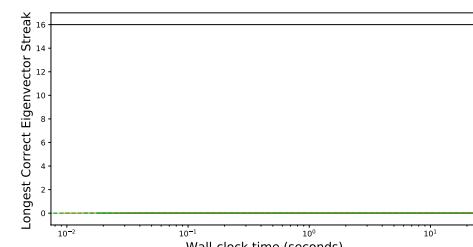
(d)  $V = \pi/128$



(e)  $V = \pi/256$



(f)  $V = \pi/512$



(g)  $V = \pi/1024$

Figure 11: NIPS bag of words