⚃ EigenGame Unloaded ⚁
When playing games is better than optimizing

Ian Gemp [* 1] **Brian McWilliams** [* 1] **Claire Vernade** [1] **Thore Graepel** [1]

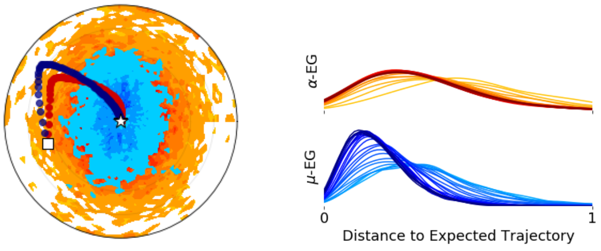
Abstract

We build on the recently proposed EigenGame that views eigendecomposition as a competitive game. EigenGame's updates are biased if computed using minibatches of data, which hinders convergence and more sophisticated parallelism in the stochastic setting. In this work, we propose an unbiased stochastic update that is asymptotically equivalent to EigenGame, enjoys greater parallelism allowing computation on datasets of larger sample sizes, and outperforms EigenGame in experiments. We present applications to finding the principal components of massive datasets and performing spectral clustering of graphs. We analyze and discuss our proposed update in the context of EigenGame and the shift in perspective from optimization to games.

1. Introduction

Large, high-dimensional datasets containing billions of samples are commonplace. Dimensionality reduction to extract the most informative features is an important step in the data processing pipeline which enables faster learning of classifiers and regressors (Dhillon et al., 2013), clustering (Kannan & Vempala, 2009), and interpretable visualizations. Many dimensionality reduction and clustering techniques rely on eigendecomposition at their core including principal component analysis (Jolliffe, 2002), locally linear embedding (Roweis & Saul, 2000), multidimensional scaling (Mead, 1992), Isomap (Tenenbaum et al., 2000), and graph spectral clustering (Von Luxburg, 2007).

Numerical solutions to the eigenvalue problem have been approached from a variety of angles for centuries: Jacobi's method, Rayleigh quotient, power (von Mises) iteration (Golub & Van der Vorst, 2000). For large datasets that do not fit in memory, approaches that access only subsets—or *minibatches*—of the data at a time have been proposed.



(a) (b)

Figure 1. Comparing α-EigenGame (Gemp et al., 2021) and μ-EigenGame (this work) over 1000 trials with a batch size of 1. **(a)** The expected trajectory[2] of each algorithm from initialization (□) to the true value of the third eigenvector ($\star$). The density of the shaded region shows the distribution of steps taken by the stochastic variant of each algorithm after 100 burn-in steps. Although the expected path of α-EG is slightly more direct, its stochastic variant has much larger variance. **(b)** The distribution of distances between stochastic update trajectories and the expected trajectory of each algorithm as a function of iteration count (**bolder** lines are later iterations and modes further left are more desirable). With increasing iterations, the stochastic μ-EG trajectory approaches its expected value whereas α-EG exhibits larger bias.

Recently, EigenGame (Gemp et al., 2021) was introduced with the novel perspective of viewing the set of eigenvectors as the Nash strategy of a suitably defined game. While this work demonstrated an algorithm that was empirically competitive given access to only subsets of the data, its performance degraded with smaller minibatch sizes.

One path towards circumventing EigenGame's need for large minibatch sizes is parallelization. In a data parallel approach, updates are computed in parallel on partitions of the data and then combined such that the aggregate update is equivalent to a single large-batch update. The technical obstacle preventing such an approach for EigenGame lies in the bias of its updates, i.e., the divide-and-conquer EigenGame update is not equivalent to the large-batch update. Biased updates are not just a theoretical nuisance; they can slow and even prevent convergence to the solution.

[*] Equal contribution [1] DeepMind, London, UK. Correspondence to: Ian Gemp <imgemp@google.com>, Brian McWilliams <bmcw@google.com>.

[2] The trajectory when updating with $\mathbb{E}[X_t^\top X_t]$.

In this work we introduce a formulation of EigenGame which admits unbiased updates which we term μ-EigenGame. When necessary we will refer to the original formulation of EigenGame as α-EigenGame.[3]

The difference between μ-EigenGame and α-EigenGame is illustrated in Figure 1. Unbiased updates allow us to increase the effective batch size using data parallelism. Lower variance updates mean that μ-EigenGame should converge faster and to more accurate solutions than α-EigenGame regardless of batch size.

Our contributions: In the rest of the paper, we present our new formulation of the EigenGame problem, analyze its bias and propose a **novel unbiased parallel variant, μ-EigenGame**. We demonstrate its performance with extensive experiments including applications to massive data sets and clustering a large social network graph. We conclude with discussions of the algorithm's design and context within optimization, game theory, and neuroscience.

2. Preliminaries

In this work, we aim to compute the top-k right singular vectors of data X, which is either represented as a matrix, $X \in \mathbb{R}^{n\times d}$, of n d-dimensional samples, or as a d-dimensional random variable. In either case, we assume we can repeatedly sample a minibatch X_t from the data of size $n' < n$, $X_t \in \mathbb{R}^{n'\times d}$. The top-$k$ right singular vectors of the dataset are then given by the top-k eigenvectors of the (sample) covariance matrix, $\mathbb{E}[\frac{1}{n'}X_t^\top X_t]$.

For small datasets, SVD is appropriate. However, SVD's time, $\mathcal{O}(\min\{nd^2, n^2d\})$, and space, $\mathcal{O}(nd)$, complexity prohibit its use for larger datasets (Shamir, 2015) including when X is a random variable. For larger datasets, stochastic, randomized, or sketching algorithms are better suited. Stochastic algorithms such as Oja's algorithm (Oja, 1982; Allen-Zhu & Li, 2017) perform power iteration (Rutishauser, 1971) to iteratively improve an approximation, maintaining orthogonality of the learned eigenvectors typically through repeated QR decompositions. Alternatively, randomized algorithms (Halko et al., 2011; Sarlos, 2006; Cohen et al., 2017) first compute a random projection of the data onto a $(k+p)$-subspace approximately containing the top-k subspace. This is done using techniques similar to Krylov subspace iteration methods (Musco & Musco, 2015). After projecting, a call to SVD is then made on this reduced-dimensionality data matrix. Sketching algorithms (Feldman et al., 2020) such as Frequent Directions (Ghashami et al., 2016) also target learning the top-k subspace by maintaining an overcomplete sketch matrix of size $(k+p)\times d$ and maintaining a span of the top subspace with repeated calls to SVD. In both the randomized and sketching approaches, a final SVD of the $n\times(k+p)$ dataset is required to recover the desired singular vectors. Although the SVD scales linearly in the number of samples, some datasets are too large to fit in memory; in this case, an out-of-memory SVD may suffice (Haidar et al., 2017). For this reason, the direct approach of stochastic algorithms, which avoid an SVD call altogether, is appealing when processing very large datasets.

Notation: We follow the same notation as Gemp et al. (2021). Variables returned by an approximation algorithm are distinguished from the true solutions with hats, e.g., the column-wise matrix of eigenvectors $\hat{V}$ approximates V. We order the columns of V such that the ith column, v_i, is the eigenvector with the ith largest eigenvalue λ_i. The set of all eigenvectors $\{\hat{v}_j\}$ with λ_j larger than λ_i, namely v_i's *parents*, will be denoted by $v_{j<i}$. Similarly, sums over subsets of indices may be abbreviated as $\sum_{j<i} = \sum_{j=1}^{i-1}$. The set of all parents *and* children of v_i are denoted by v_{-i}. We assume the standard Euclidean inner product $\langle u, v\rangle = u^\top v$ and denote the unit-sphere and simplex in ambient space $\mathbb{R}^d$ with $\mathcal{S}^{d-1}$ and Δ^{d-1} respectively.

3. EigenGame

We build on the algorithm introduced by Gemp et al. (2021), which we refer to here as α-EigenGame. This algorithm is derived by formulating the eigendecomposition of a symmetric positive definite matrix as the Nash equilibrium of a game among k players, each player i owning the approximate eigenvector $\hat{v}_i$. Each player is also assigned a utility function, $u_i^\alpha(\hat{v}_i|\hat{v}_{j<i})$, that they must maximize:

$$u_i^\alpha(\hat{v}_i|\hat{v}_{j<i}) = \overbrace{\hat{v}_i^\top \Sigma \hat{v}_i}^{\text{Var}} - \sum_{j<i} \overbrace{\frac{\langle \hat{v}_i, \Sigma \hat{v}_j\rangle^2}{\langle \hat{v}_j, \Sigma \hat{v}_j\rangle}}^{\perp\text{-penalty}}. \tag{1}$$

These utilities balance two terms, one that rewards a $\hat{v}_i$ that captures more variance in the data and a second term that penalizes $\hat{v}_i$ for failing to be orthogonal to each of its parents $\hat{v}_{j<i}$ (the gradients derived from these terms are indicated with Var and $\perp$-penalty in equation (2)). In α-EigenGame, each player simultaneously updates $\hat{v}_i$ with gradient ascent, and it is shown that this process converges to the Nash equilibrium. We are interested in extending this approach to the data parallel setting where each player i may distribute its update computation over multiple devices.

3.1. Biased updates

Consider partitioning the sample covariance matrix into a sum of $\frac{n}{n'}$ matrices as $\Sigma = \frac{1}{n}X^\top X = \frac{n'}{n}\sum_t \frac{1}{n'}X_t^\top X_t = \frac{n'}{n}\sum_t \Sigma_t$. We would like α-EigenGame to parallelize over these partitions. However, the gradient direction for $\hat{v}_i$ does

[3] μ signifies unbiased or *un*loaded and α denotes original.

not decompose cleanly over the data partitions:

$$\nabla_i^\alpha \propto \overbrace{\Sigma\hat{v}_i}^{\text{Var}} - \sum_{j<i} \overbrace{\frac{\hat{v}_i^\top \Sigma \hat{v}_j}{v_j^\top \Sigma \hat{v}_j} \Sigma \hat{v}_j}^{\perp\text{-penalty}} \tag{2}$$

$$= \frac{n'}{n} \sum_t \Big[\Sigma_t \hat{v}_i - \sum_{j<i} \boxed{\frac{\hat{v}_i^\top \Sigma \hat{v}_j}{\hat{v}_j^\top \Sigma \hat{v}_j}} \Sigma_t \hat{v}_j \Big]. \tag{3}$$

We include the superscript α on the EigenGame gradient to differentiate it from the μ-EigenGame direction later. The nonlinear appearance of Σ in the penalty terms makes obtaining an unbiased gradient difficult. The quadratic term in the numerator of equation (2) could be made unbiased by using two sample estimates of Σ, one for each term. But the appearance of the term in the denominator does not have an easy solution. Σ_t is likely singular for small n' ($n' < d$) which increases the likelihood of a small denominator, i.e., a large penalty coefficient (boxed), if we were to estimate the denominator with samples. The result is an update that emphasizes penalizing orthogonality over capturing data variance. Techniques exist to reduce the bias of samples of ratios of random variables, but to our knowledge, techniques to obtain unbiased estimates are not available. This was conjectured by Gemp et al. (2021) as the reason for why α-EigenGame performed worse with small minibatches.

4. *Unbiased* EigenGame

It is helpful to rearrange equation (3) to shift perspective from estimating a penalty coefficient (in red) to estimating a penalty direction (in blue):

$$\nabla_i^\alpha \propto \frac{n'}{n} \sum_t \Big[\Sigma_t \hat{v}_i - \sum_{j<i} \hat{v}_i^\top \Sigma_t \hat{v}_j \frac{\Sigma \hat{v}_j}{\hat{v}_j^\top \Sigma \hat{v}_j} \Big]. \tag{4}$$

The penalty direction in equation (4) is still difficult to estimate. However, consider the case where $\hat{v}_j$ is any eigenvector of Σ with associated (unknown) eigenvalue λ'. In this case, $\Sigma \hat{v}_j = \lambda' \hat{v}_j$ and the penalty direction (in blue) simplifies to $\hat{v}_j$ because $||\hat{v}_j|| = 1$. While this assumption is certainly not met at initialization, the goal of α-EigenGame is to lead each $\hat{v}_j$ towards v_j, so we expect this assumption might be met approximately after some iterations.

This intuition motivates the following μ-EigenGame update direction for $\hat{v}_i$ with inexact parents $\hat{v}_j$ (compare orange in equation (5) to blue in equation (4)):

$$\Delta_i^\mu = \Sigma \hat{v}_i - \sum_{j<i} (\hat{v}_i^\top \Sigma \hat{v}_j) \hat{v}_j \tag{5}$$

$$= \frac{n'}{n} \sum_t \Big[\Sigma_t \hat{v}_i - \sum_{j<i} (\hat{v}_i^\top \Sigma_t \hat{v}_j) \hat{v}_j \Big]. \tag{6}$$

We use Δ instead of ∇ because the direction is not a gradient (discussed later). Notice how the strictly linear appearance of Σ in μ-EigenGame allows the update to easily decompose over the data partitions in equation (6).

The μ-EigenGame update satisfies two important properties.

Lemma 1 (Asymptotic Equivalence)**.** *The μ-EigenGame direction, Δ_i^μ, with exact parents ($\hat{v}_j = v_j \; \forall \; j < i$) is equivalent to α-EigenGame.*

Proof. We start with α-EigenGame and add a superscript e to its gradient to emphasize this is the gradient computed with exact parents ($\hat{v}_j = v_j$). Then simplifying, we find

$$\nabla_i^{\alpha,e} \propto \Sigma \hat{v}_i - \sum_{j<i} \frac{\hat{v}_i^\top \Sigma v_j}{v_j^\top \Sigma v_j} \Sigma v_j \tag{7}$$

$$= \Sigma \hat{v}_i - \sum_{j<i} \frac{\hat{v}_i^\top \Sigma v_j}{v_j^\top \cancel{\lambda_j} v_j} \cancel{\lambda_j} v_j \tag{8}$$

$$= \Sigma \hat{v}_i - \sum_{j<i} (\hat{v}_i^\top \Sigma v_j) v_j = \Delta_i^\mu. \tag{9}$$

□

Therefore, once the first $(i-1)$ eigenvectors are learned, learning the ith eigenvector with μ-EigenGame is equivalent to learning with α-EigenGame.

Lemma 2 (Zero Bias)**.** *Monte Carlo approximation of Δ_i^μ does not introduce any bias.*

Proof. Let $X \sim p(X)$ where $X \in \mathbb{R}^d$ and $p(X)$ is the uniform distribution over the dataset. Then

$$\mathbb{E}[\Delta_i^\mu] = \mathbb{E}[(XX^\top)\hat{v}_i - \sum_{j<i} (\hat{v}_i^\top (XX^\top) \hat{v}_j) \hat{v}_j] \tag{10}$$

$$= \mathbb{E}[XX^\top] \hat{v}_i - \sum_{j<i} (\hat{v}_i^\top \mathbb{E}[XX^\top] \hat{v}_j) \hat{v}_j \tag{11}$$

$$= \Sigma \hat{v}_i - \sum_{j<i} (\hat{v}_i^\top \Sigma \hat{v}_j) \hat{v}_j. \tag{12}$$

where all expectations are with respect to $p(X)$. □

Figure 2 illustrates μ-EigenGame's reduced bias when estimating the utility function (and resulting optimum) from an average over minibatches.

These two lemmas provide the foundation for a performant algorithm. The first enables convergence to the desired solution, while the second facilitates scaling to larger datasets.

4.1. Convergence to PCA

We begin by leveraging the first, asymptotic equivalence.

[5] Overestimation is expected by Jensen's: $\mathbb{E}[\frac{1}{X}] \geq \frac{1}{\mathbb{E}[X]}$.

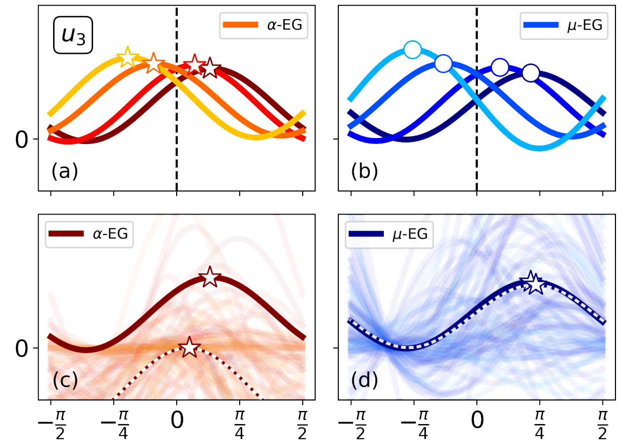


Figure 2. Bias. In the top row, player 3's utility is given for parents mis-specified by an angular distance along the sphere of $\angle(\hat{v}_{j<i}, v_{j<i}) \in [-20°, -10°, 10°, 20°]$ moving from light to dark. Player 3's mis-specification, $\angle(\hat{v}_i, v_i)$, is given by the x-axis (optimum is at 0 radians). α-EigenGame (a) exhibits slightly lower sensitivity than μ-EigenGame (b) to mis-specified parents. However, when the utilities are estimated using samples $X_t \sim p(X)$ (samples shown in background), μ-EigenGame remains accurate (d), while α-EigenGame (c) returns a utility (dotted line) with an optimum that is shifted to the left and down. The downward shift occurs because of the random variable in the denominator of the penalty terms (see equation (4)).[5]

Pseudo-Utility We arrived at μ-EigenGame by analyzing and improving properties of the α-EigenGame update. However, the μ-EigenGame update direction is linear in each $\hat{v}_i$. This suggests we may be able to design a *pseudo*-utility function for it. Rearranging the update direction from equation (5) as

$$\Delta_i^\mu = \Sigma\hat{v}_i - \sum_{j<i} \hat{v}_j(\hat{v}_j^\top \Sigma \hat{v}_i) \tag{13}$$

$$= \Big[I - \sum_{j<i} \hat{v}_j \hat{v}_j^\top\Big]\Sigma\hat{v}_i = \tilde{\nabla}_i^\mu \tag{14}$$

reveals that we can reverse-engineer the following objective

$$u_i^\mu = \hat{v}_i^\top \Big[\overbrace{I - \sum_{j<i} \hat{v}_j \hat{v}_j^\top}^{\text{deflation}}\Big]\Sigma \bullet[\hat{v}_i] \tag{15}$$

where ● is the stop gradient operator commonly used in deep learning packages. As the name implies, ● stops gradients from flowing through its argument so that equation (15) appears linear in $\hat{v}_i$ instead of quadratic when differentiating the expression. In light of this, we have renamed Δ_i^μ to $\tilde{\nabla}_i^\mu$ to emphasize that it is a *pseudo*-gradient of u_i^μ. Note that without the stop gradient, the true gradient of u_i^μ would be $\frac{1}{2}[A + A^\top]\hat{v}_i$ where $A = [I - \sum_{j<i} \hat{v}^\top \hat{v}_j]\Sigma$.

The utility function u_i^μ has an intuitive meaning. It is the Rayleigh quotient for the matrix $\Sigma_i = [I - \sum_{j<i} \hat{v}^\top \hat{v}_j]\Sigma$, which represents the covariance after the subspace spanned by $\hat{v}_{j<i}$ has been removed. In other words, player i is directed to find the largest eigenvalue in the orthogonal complement of the approximate top-$(i-1)$ subspace. This approach is known as "deflating" the matrix Σ.

Theorem 1. *PCA is the unique Nash of μ-EigenGame given symmetric Σ with distinct eigenvalues.*

Proof. We will show by induction that each v_i is the unique best response to v_{-i}, which implies they constitute the unique Nash equilibrium. First, consider player 1's utility. It is simply the Rayleigh quotient of Σ because $\hat{v}_1$ is constrained to the unit-sphere, i.e., $u_1^\mu = \hat{v}_1^\top \Sigma \hat{v}_1 = \frac{\hat{v}_1^\top \Sigma \hat{v}_1}{\hat{v}_1^\top \hat{v}_1}$. Therefore, we know v_1 maximizes u_1^μ and the maximizer is unique because the eigenvalues are distinct. In game theory parlance, v_1 is a *best response* to all other v_{-1}. The proof then continues by induction. The utility of player i is $u_i^\mu = \hat{v}_i^\top [I - \sum_{j<i} v_j v_j^\top]\Sigma v_1$, which is the Rayleigh quotient of Σ with the subspace spanned by the top $(i-1)$ eigenvectors removed. Therefore, the maximizer of u_i^μ is the largest eigenvector in the remaining subspace, i.e., v_i. As before, the eigenvalues are assumed distinct, so this maximizer is unique. This shows that each v_i is a best response to v_{-i}, therefore, the set of v_i forms the unique Nash. □

Notice how the induction proof of Theorem 1 relies on a) the hierarchy of vectors (v_1 does not depend on v_{-1}) and b) the fact that u_i^μ need only be a sensible utility when all player i's parents are eigenvectors. We revisit this in conjunction with Figure 8 later in discussion section 6.2 to aid researchers in the design of future approaches.

The Nash property is important because it enables the use of any black-box procedure for computing best responses. Like prior work, we develop a Riemannian gradient method for optimizing each utility, however, that is not a requirement. Any optimization oracle suffices if it can efficiently compute a best response.

Riemannian Manifolds. Before introducing an algorithm for μ-EigenGame, we first briefly review necessary terminology for learning on Riemannian manifolds (Absil et al., 2009), specifically for the sphere. The notation $\mathcal{T}_{\hat{v}_i}\mathcal{S}^{d-1}$ denotes the set of vectors tangent to the sphere at a point $\hat{v}_i$ (i.e., any vector orthogonal to $\hat{v}_i$). $R_{\hat{v}_i}(z) = \frac{\hat{v}_i + z}{||\hat{v}_i + z||}$ is the commonly used restriction of the retraction on $\mathcal{S}^{d-1}$ to the tangent bundle at $\hat{v}_i$ (i.e., step in tangent direction z and then unit-normalize the result). The operator $\Pi_{\hat{v}_i}(y) = (I - \hat{v}_i^\top \hat{v}_i)y$ projects the direction y onto $\mathcal{T}_{\hat{v}_i}\mathcal{S}^{d-1}$. Combining these tools together results in a movement along the Riemannian manifold: $\hat{v}_i^{(t+1)} \leftarrow R_{\hat{v}_i}\big(\Pi_{\hat{v}_i}(y)\big)$.

We present pseudocode for μ-EigenGame below where computation is parallelized both over the k players and over M

Algorithm 1 μ-EigenGame^R

```
Given: data stream $X_t \in \mathbb{R}^{n' \times d}$, number of parallel machines $M$ per player (minibatch size per machine $n'' = \frac{n'}{M}$), initial vectors $\hat{v}_i^0 \in \mathcal{S}^{d-1}$, step size sequence $\eta_t$, and number of iterations $T$.
$\hat{v}_i \leftarrow \hat{v}_i^0$ for all $i$
for $t = 1 : T$ do
    parfor $i = 1 : k$ do
        parfor $m = 1 : M$ do
            rewards $\leftarrow X_{tm}^\top X_{tm} \hat{v}_i$
            penalties $\leftarrow \sum_{j<i} \langle X_{tm} \hat{v}_i, X_{tm} \hat{v}_j \rangle \hat{v}_j$
            $\tilde{\nabla}_{im}^{\mu} \leftarrow$ rewards $-$ penalties
            $\tilde{\nabla}_{im}^{\mu,R} \leftarrow \tilde{\nabla}_{im}^{\mu} - \langle \tilde{\nabla}_{im}^{\mu}, \hat{v}_i \rangle \hat{v}_i$
        end parfor
        $\tilde{\nabla}_{i}^{\mu,R} \leftarrow \frac{n''}{n'} \sum_m [\tilde{\nabla}_{im}^{\mu,R}]$
        $\hat{v}_i' \leftarrow \hat{v}_i + \eta_t \tilde{\nabla}_{i}^{\mu,R}$
        $\hat{v}_i \leftarrow \frac{\hat{v}_i'}{||\hat{v}_i'||}$
    end parfor
end for
return all $\hat{v}_i$
```

machines per player.

Before proceeding to a convergence proof, we strengthen Lemma 1 with an approximate asymptotic equivalence result.

Lemma 3. *An $\mathcal{O}(\epsilon)$ angular error of parent $\hat{v}_{j<i}$ implies an $\mathcal{O}(\epsilon)$ angular error in the location of the solution for $\hat{v}_i$.*

Proof. Proof in Appendix A. □

The asymptotic equivalence of μ-EigenGame to α-EigenGame ensures μ-EigenGame is globally, asymptotically convergent and its unbiased updates ensure it is scalable.

Theorem 2 (Global Convergence)**.** *Given a positive definite matrix Σ with distinct eigenvalues, full-batch updates ($n' = n$), and a square-summable, not summable step size sequence η_t (e.g., $1/t$), Algorithm 1 converges to the top-k eigenvectors asymptotically ($\lim_{T\to\infty}$) with probability* 1.

Proof. Assume none of the $\hat{v}_i$ are initialized to an angle exactly 90° away from the true eigenvector: $\langle \hat{v}_i, v_i \rangle \neq 0$. The set of vectors $\{\hat{v}_i : \langle \hat{v}_i, v_i \rangle = 0\}$ has Lebesgue measure 0, therefore, the above assumption holds w.p.1. The update direction for the top eigenvector $\hat{v}_1$ is exactly equal to that of α-EigenGame ($\tilde{\nabla}_1^{\mu} = \nabla_1^{\alpha}$), therefore, they have the same limit points for $\hat{v}_1$. The proof then proceeds by induction. As $\hat{v}_{j<i}$ approach their limit points, the update for the ith eigenvector $\hat{v}_i$ approaches that of α-EigenGame ($\tilde{\nabla}_i^{\mu} = \nabla_i^{\alpha}$) and, by Lemma 3, the stable limit point of μ-EigenGame also approaches the top-k eigenvectors. The result is then obtained by applying Theorem 7 of Shah (2019) with the following information: a) the unit-sphere is a compact manifold with an injectivity radius of π, b) the update field is a polynomial and therefore smooth (analytic), and c) there is zero noise because we have assumed full-batch updates ($n' = n$). □

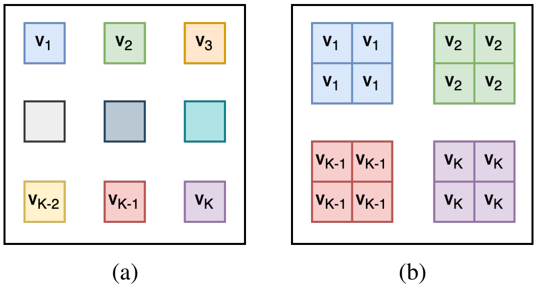


Figure 3. (a) Extreme model parallelism as proposed in α-EigenGame. (b) An approach to model and data parallelism enabled by μ-EigenGame. Each coloured square is a separate device (in this example $M = 4$). Copies of estimates are colour-coded. Updates are summed or averaged across copies to allow for a larger effective batch size.

This asymptotic convergence result is complimentary to the result in Gemp et al. (2021) where each $\hat{v}_i$ is learned in sequence. In contrast, the proof above applies when learning all $\hat{v}_i$ in parallel.

In the ensuing discussion and later experiments, it is not always possible to set $n' = n$ due to limited computational resources. We leave a formal convergence proof of the stochastic setting where $n' < n$ to future work, but address this gap empirically in Section 5.

4.2. Model and Data Parallelism

With the fixed point of μ-EigenGame and convergence of Algorithm 1 established, we consider the implications of Lemma 2 for scaling and parallelism.

In our setting we have a number of connected devices. Specifically we consider the parallel framework specified by TPUv3 available in Google Cloud, however our setup is general and applicable to any multi-host, multi-device system. The α-EigenGame formulation (Gemp et al., 2021) considers an extreme form of model parallelism (Figure 3a) where each device has its own unique set of eigenvectors.

In this work we further consider a different form of model and data parallelism which is directly enabled by having unbiased updates (Figure 3b). This enables μ-EigenGame to deal with both high-dimensional problems as well as massive sample sizes. Here each set of eigenvectors are copied on M devices. Update directions are computed on each device individually using a different data stream and then combined by summing or averaging. Updates are applied to a single copy and this is duplicated across the $M-1$ remaining devices. In this way updates are computed using a $M\times$ larger effective batch size while still allowing device-wise model parallelism. This setting is particularly useful when the number of samples is very large. This form of parallelism is not possible using the original EigenGame formulation since it relies on combining *unbiased* updates. In this sense, the parallelism discussed in this work generalizes that introduced in Gemp et al. (2021).

Note that we also allow for within-device parallelism. That is, each v_i in Figure 3 is a contiguous collection of eigenvectors which are updated independently, in parallel, on a given device (for example using `vmap` in Jax). We provide detailed Jax pseudo-code for parallel μ-EigenGame in Appendix B. We compare the empirical scaling performance of μ-EigenGame against α-EigenGame on a 14 billion sample dataset in section 5.4.

5. Experiments

As in EigenGame, we omit the projection of gradients onto the tangent space of the sphere; specifically, we omit line 9 in Algorithm 1. As discussed in Gemp et al. (2021), this has the effect of intelligently adapting the step size to use smaller learning rates near the fixed point.

5.1. Metrics

To ease comparison with previous work, we count the *longest correct eigenvector streak* as introduced in Gemp et al. (2021), which measures the number of eigenvectors that have been learned, in order, to within an angular threshold (e.g., $\frac{\pi}{8}$) of the true eigenvectors. We also measure how well the set of $\hat{v}_i$ captures the top-k subspace with a normalized subspace distance: $1-\frac{1}{k}\cdot\text{Tr}(U^*P)\in[0,1]$ where $U^*=VV^\dagger$ and $P=\hat{V}\hat{V}^\dagger$ (Tang, 2019). Shading in plots indicates ± standard error of the mean.

5.2. Synthetic

We first validate μ-EigenGame in a full-batch setting on two synthetic datasets: one with exponentially decaying spectrum; the other with a linearly decaying spectrum. Figure 4 shows μ-EigenGame outperforms α-EigenGame on the former and matches its performance on the latter. We discuss possible reasons for this gap in the discussion in Section 6.

5.3. MNIST

We compare μ-EigenGame against α-EigenGame, GHA (Sanger, 1989), Matrix Krasulina (Tang, 2019), and Oja's algorithm (Allen-Zhu & Li, 2017) on the MNIST dataset. We flatten each image in the training set to obtain a $60,000\times 784$ dimensional matrix X.

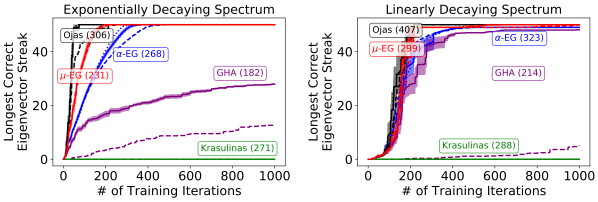


Figure 4. Synthetic Experiment. Runtime (milliseconds) in legend.

Figure 5 demonstrates μ-EigenGame's robustness to minibatch size. It performs best in the longest streak metric and better than α-EigenGame in subspace distance. We attribute this performance boost to the unbiased updates of μ-EigenGame.

5.4. Meena Conversational Model Dataset

This dataset consists of embeddings computed using the Meena language model (Adiwardana et al., 2020). The embedded data consists of a subset of the 40 billion words used to train the transformer-based model. The subset was preprocessed to remove duplicates and then embedded using the trained model. Full details of the dataset and model can be found in Adiwardana et al. (2020). The dataset consists of $n\approx 14$ billion embeddings each with dimensionality $d=2560$; its total size is 131TB. Due to its moderate dimensionality we can exactly compute the ground truth solution by stochastically accumulating the covariance matrix of the data and computing its eigendecomposition. On a single machine this takes approximately 1.5 days (but is embarrassingly parallelizable with MapReduce).

We use minibatches of size 4,096 in each TPU. We do model parallelism across 8 TPUs so we see 32,768 samples per iteration per set of eigenvectors. We test two additional degrees of data parallelism with 4× (16 TPUs with 131,072 samples) and 8× (32 TPUs with 262,144 samples) the amount of data per iteration respectively. We compute and apply updates in Optax using SGD with a learning rate of 5×10^{-5} and Nesterov momentum with a factor of 0.9.

We compare the performance of μ-EigenGame against α-EigenGame as a function of the degree of parallelism in computing the top $k=256$ eigenvectors. Each TPU is tasked with learning 32 contiguous eigenvectors. Figure 6 reports on the longest correct streak of eigenvectors learned to an angular tolerance of $\pi/8$ (with standard error computed over five runs). We see that increasing the degree of parallelism has no effect on the performance of α-EigenGame. As expected, α-EigenGame is unable to take advantage of the higher data throughput since its updates are biased and cannot be meaningfully linearly combined across copies.

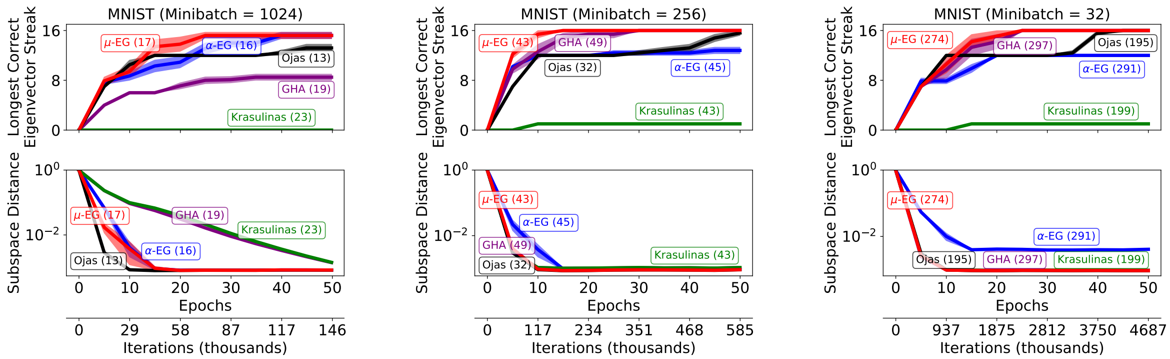


Figure 5. MNIST Experiment. Runtime (seconds) in legend. Each column evaluates a different minibatch size $\in \{1024, 256, 32\}$.

The performance of μ-EigenGame scales with the effective batch size achieved through parallelism. μ-EigenGame ($8\times$) is able to recover 256 eigenvectors after 40,000 iterations (approximately 0.75 epochs) in 2 hours 45 minutes.

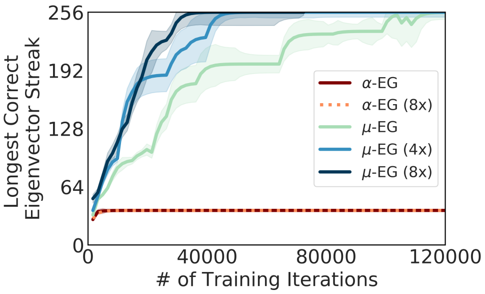


Figure 6. Comparison between μ-EigenGame and α-EigenGame with different degrees of data parallelism (in parentheses) on the Meena dataset.

5.5. Spectral clustering on Graphs

We conducted an experiment on learning the eigenvectors of the Laplacian of a social network graph (Leskovec & McAuley, 2012) for the purpose of spectral clustering. The eigenvalues of the graph Laplacian reveal several interesting properties as well such as the number of connected components, an approximation to the sparsest cut, and the diameter of a connected graph (Chung et al., 1994).

Given a graph with a set of nodes $\mathcal{V}$ and set of edges $\mathcal{E}$, the graph Laplacian can be written as $\mathcal{L} = X^\top X$ where each row of the incidence matrix $X \in \mathbb{R}^{|\mathcal{E}| \times |\mathcal{V}|}$ represents a distinct edge; $X_{e=(i,j)\in\mathcal{E}}$ is a vector containing only 2 nonzero entries, a 1 at index i and a -1 at index j (Horaud, 2009). In this setting, the eigenvectors of primary interest are the bottom-k $(\lambda_{|\mathcal{V}|}, \lambda_{|\mathcal{V}|-1}, \ldots)$ rather than the top-k $(\lambda_1, \lambda_2, \ldots)$, however, a simple algebraic manipulation allows us to reuse a top-k solver. By defining the matrix $\mathcal{L}^- = \lambda^* I - \mathcal{L}$ with $\lambda^* > \lambda_1$, we ensure $\mathcal{L}^- \succ 0$ and the top-k eigenvectors of $\mathcal{L}^-$ are the bottom-k of $\mathcal{L}$.

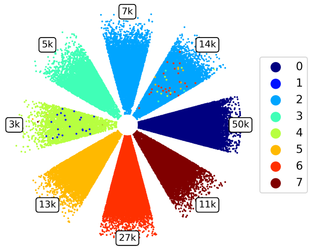


Figure 7. Facebook Page Networks. Petals differentiate ground truth clusters; colors differentiate learned clusters. Petals are ideally colored according to the color bar starting with the rightmost petal and proceeding counterclockwise. Numbers indicate ground truth cluster size. Clusters are extracted by running k-means clustering on the learned eigenvectors $\hat{V} \in \mathbb{R}^{|\mathcal{V}|\times k}$ (samples on rows).

The update in equation (5) is transformed into

$$\tilde{\nabla}^{\mu}_i = (\lambda^* I - \mathcal{L})\hat{v}_i - \sum_{j<i} \left(\hat{v}_i^\top (\lambda^* I - \mathcal{L})\hat{v}_j\right)\hat{v}_j. \quad (16)$$

We provide efficient pseudo-code in Appendix C.

The Facebook graph consists of $134,833$ nodes, $1,380,293$ edges, and 8 connected components where each component is a network formed by a set of Facebook pages belonging to a distinct category, e.g., Government, TV shows, etc. (Rozemberczki et al., 2019). By projecting the graph onto the bottom 8 eigenvectors of the graph Laplacian and then running k-means clustering (Pedregosa et al., 2011), we are able to approximately recover 7/8 ground truth clusters (see Figure 7) obtaining a V-measure[6] of $0.967 \in [0, 1]$.

6. Discussion

6.1. Acceleration

We conjecture that μ-EigenGame converges more quickly than α-EigenGame because of the following two claims.

[6] Harmonic mean of homogeneity and completeness.

Claim 1 The penalty terms of $\tilde{\nabla}^{\mu}_i$ are all within 90° of those of ∇^{α}_i because

$$\Big\langle \frac{M\hat{v}_j}{\hat{v}_j^\top M\hat{v}_j}, \hat{v}_j \Big\rangle = 1 > 0. \quad \square \tag{17}$$

Claim 2 The penalty terms of $\tilde{\nabla}^{\mu}_i$ are all smaller in magnitude than those of ∇^{α}_i:

$$||\hat{v}_j|| \leq \Big|\Big| \frac{M\hat{v}_j}{\hat{v}_j^\top M\hat{v}_j} \Big|\Big|. \tag{18}$$

Indeed, consider the direction $M\hat{v}_j$. By properties of the vector rejection, we know the rejection of this direction onto the tangent space of the unit sphere has magnitude less than or equal to that of the original vector, $||M\hat{v}_j||$. The projection is $(I - \hat{v}_j\hat{v}_j^\top)(M\hat{v}_j)$. Therefore, the rejection is $\hat{v}_j\hat{v}_j^\top(M\hat{v}_j)$ and, by the preceding argument, we know its magnitude $|\hat{v}_j^\top M\hat{v}_j|||\hat{v}_j||$ is less than or equal to $||M\hat{v}_j||$. Rearranging the inequality completes the proof. □

By **Claim 1**, the penalty directions of μ-EG and α-EG approximately agree. And by **Claim 2**, α-EG's penalty direction is shorter. Consider a scenario where a parent of $\hat{v}_i$ has not converged and transiently occupies space along $\hat{v}_i$'s geodesic to its true endpoint $\hat{v}_i$, a strong penalty term will force $\hat{v}_i$ to take a roundabout trajectory, thereby slowing its convergence. A weaker penalty term allows $\hat{v}_i$ to pass through regions occupied by its parent **as long as** its parent is not an eigenvector. Recall from Section 4 that the two utilities are equivalent when the parents are eigenvectors.

6.2. Utilities to Updates and Back

Figure 8 summarizes the relationships advising the designs of the various EigenGame algorithms. Starting from the α-EigenGame utility, its update is arrived at by simply following the standard gradient ascent paradigm. In noticing that stochastic estimates of the gradient are biased, we arrive at the μ-EigenGame update by considering how to remove this bias in a principled manner.

Sacrificing the exact steepest decent direction for a direction that allows unbiased estimates is a tradeoff that in this case has benefits. Also, while $\tilde{\nabla}^{\mu}_i$ is not a gradient (except with exact parents), the new penalties have properties (above) that make them intuitively more desirable than the originals; they are adaptive to the state of the system.

From this new update we can derive a pseudo-utility using the stop gradient operator which has the desired theoretical properties. However, it is unlikely that this utility would be developed independently of these steps to solve the problem at hand (see Appendix D.1 for more details).

This suggests an alternative approach to algorithm design complementary to the optimization perspective: directly designing updates themselves which converge to the desired solution, reminiscent of previous paradigms that drove neuro-inspired learning rules.

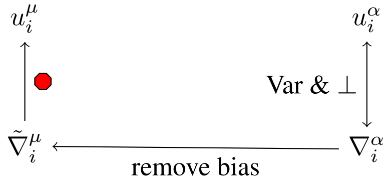


Figure 8. This diagram presents the relationships between utilities and updates. An arrow indicates the endpoint is reasonably derived from the origin; the lack of an arrow indicates the direction is unlikely.

6.3. Bridging Hebbian and Optimization Approaches

The Generalized Hebbian Algorithm (GHA) (Sanger, 1989; Gang et al., 2019; Chen et al., 2019) update direction for $\hat{v}_i$ with inexact parents $\hat{v}_j$ is similar to μ-EigenGame:

$$\Delta^{gha}_i = \Sigma\hat{v}_i - \sum_{j \leq i} (\hat{v}_i^\top \Sigma \hat{v}_j)\hat{v}_j. \tag{19}$$

Σ appears linearly in this update so GHA parallelizes as well. In contrast to μ-EigenGame, GHA additionally penalizes the alignment of $\hat{v}_i$ to itself and removes the unit norm constraint on $\hat{v}_i$ (not shown). Without any constraints, GHA overflows in experiments. We take the approach of Gemp et al. (2021) and constrain $\hat{v}_i$ to the unit-ball ($||\hat{v}_i|| \leq 1$) rather than the unit-sphere ($||\hat{v}_i|| = 1$).

The connection between GHA and μ-EigenGame is interesting because GHA is a Hebbian learning algorithm inspired by neuroscience whereas μ-EigenGame is inspired by game theory. Game formulations of classical machine learning problems may provide a bridge between statistical and biologically inspired viewpoints.

7. Conclusion

We introduced μ-EigenGame, an unbiased, globally convergent, parallelizable algorithm that recovers the top-k eigenvectors of a symmetric positive definite matrix. We demonstrated this feat empirically on several different datasets of varying size and application. We discussed technical components of this algorithm and its place within the wider context of game theory meets machine learning meets neuroscience.

μ-EigenGame's improved robustness to smaller minibatches makes it more amenable to being used within popular deep learning frameworks and as part of optimization (Krummenacher et al., 2016) and regularization (Miyato et al., 2018) techniques which leverage spectral information of gradient covariances or Hessians.

Acknowledgements. We would like to thank Trevor Cai, Rosalia Schneider, Dimitrios Vytiniotis for invaluable help with optimizing algorithm performance on TPU. We thank Maribeth Rauh, Zonglin Li, Daniel Adiwardana and the Meena team for providing us with data and assistance.

References

Absil, P.-A., Mahony, R., and Sepulchre, R. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2009.

Adiwardana, D., Luong, M.-T., So, D. R., Hall, J., Fiedel, N., Thoppilan, R., Yang, Z., Kulshreshtha, A., Nemade, G., Lu, Y., et al. Towards a human-like open-domain chatbot. *arXiv preprint arXiv:2001.09977*, 2020.

Allen-Zhu, Z. and Li, Y. 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)*, pp. 487–492. IEEE, 2017.

Chen, Z., Li, X., Yang, L., Haupt, J., and Zhao, T. On constrained nonconvex stochastic optimization: A case study for generalized eigenvalue decomposition. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 916–925. PMLR, 2019.

Chung, F. R., Faber, V., and Manteuffel, T. A. An upper bound on the diameter of a graph from eigenvalues associated with its Laplacian. *SIAM Journal on Discrete Mathematics*, 7(3):443–457, 1994.

Cohen, M. B., Musco, C., and Musco, C. Input sparsity time low-rank approximation via ridge leverage score sampling. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1758–1777. SIAM, 2017.

Dhillon, P. S., Foster, D. P., Kakade, S. M., and Ungar, L. H. A risk comparison of ordinary least squares vs ridge regression. *The Journal of Machine Learning Research*, 14(1):1505–1511, 2013.

Feldman, D., Schmidt, M., and Sohler, C. Turning big data into tiny data: Constant-size coresets for k-means, PCA, and projective clustering. *SIAM Journal on Computing*, 49(3):601–657, 2020.

Gang, A., Raja, H., and Bajwa, W. U. Fast and communication-efficient distributed PCA. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7450–7454. IEEE, 2019.

Gemp, I., McWilliams, B., Vernade, C., and Graepel, T. Eigengame: PCA as a Nash equilibrium. In *International Conference for Learning Representations*, 2021.

Ghashami, M., Liberty, E., Phillips, J. M., and Woodruff, D. P. Frequent directions: simple and deterministic matrix sketching. *SIAM Journal on Computing*, 45(5):1762–1792, 2016.

Golub, G. H. and Van der Vorst, H. A. Eigenvalue computation in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1-2):35–65, 2000.

Haidar, A., Kabir, K., Fayad, D., Tomov, S., and Dongarra, J. Out of memory SVD solver for big data. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7. IEEE, 2017.

Halko, N., Martinsson, P.-G., and Tropp, J. A. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

Hessel, M., Budden, D., Viola, F., Rosca, M., Sezener, E., and Hennigan, T. Optax: composable gradient transformation and optimisation, in JAX!, 2020. URL http://github.com/deepmind/optax.

Horaud, R. A short tutorial on graph Laplacians, Laplacian embedding, and spectral clustering, 2009. URL http://https://csustan.csustan.edu/~tom/Clustering/GraphLaplacian-tutorial.pdf.

Jolliffe, I. T. Principal components in regression analysis. In *Principal Component Analysis*. Springer, 2002.

Kannan, R. and Vempala, S. *Spectral algorithms*. Now Publishers Inc, 2009.

Krummenacher, G., McWilliams, B., Kilcher, Y., Buhmann, J. M., and Meinshausen, N. Scalable adaptive stochastic optimization using random projections. In *Advances in Neural Information Processing Systems*, pp. 1750–1758, 2016.

Leskovec, J. and McAuley, J. Learning to discover social circles in ego networks. *Advances in Neural Information Processing Systems*, 25:539–547, 2012.

Mead, A. Review of the development of multidimensional scaling methods. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 41(1):27–39, 1992.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Musco, C. and Musco, C. Randomized block Krylov methods for stronger and faster approximate singular value decomposition. In *Advances in Neural Information Processing Systems*, 2015.

Oja, E. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, 1982.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Roweis, S. T. and Saul, L. K. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290 (5500):2323–2326, 2000.

Rozemberczki, B., Davies, R., Sarkar, R., and Sutton, C. Gemsec: Graph embedding with self clustering. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2019*, pp. 65–72. ACM, 2019.

Rutishauser, H. Simultaneous iteration method for symmetric matrices. In *Handbook for Automatic Computation*, pp. 284–302. Springer, 1971.

Sanger, T. D. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2(6):459–473, 1989.

Sarlos, T. Improved approximation algorithms for large matrices via random projections. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pp. 143–152. IEEE, 2006.

Shah, S. M. Stochastic approximation on Riemannian manifolds. *Applied Mathematics & Optimization*, pp. 1–29, 2019.

Shamir, O. A stochastic PCA and SVD algorithm with an exponential convergence rate. In *Proceedings of the International Conference on Machine Learning*, pp. 144–152, 2015.

Tang, C. Exponentially convergent stochastic k-PCA without variance reduction. In *Advances in Neural Information Processing Systems*, pp. 12393–12404, 2019.

Tenenbaum, J. B., De Silva, V., and Langford, J. C. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

Von Luxburg, U. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

Wang, Y., Xiu, N., and Han, J. On cone of nonsymmetric positive semidefinite matrices. *Linear Algebra and its Applications*, 433(4):718–736, 2010.

A. Error Propagation / Sensitivity Analysis

Lemma 3. *An $\mathcal{O}(\epsilon)$ angular error of parent $\hat{v}_{j<i}$ implies an $\mathcal{O}(\epsilon)$ angular error in the location of the solution for $\hat{v}_i$.*

Proof. The proof proceeds in three steps:

1. $\mathcal{O}(\epsilon)$ angular error of parent $\implies$ $\mathcal{O}(\epsilon)$ Euclidean error of parent
2. $\mathcal{O}(\epsilon)$ Euclidean error of parent $\implies$ $\mathcal{O}(\epsilon)$ Euclidean error of norm of child gradient
3. $\mathcal{O}(\epsilon)$ Euclidean error of norm child gradient + instability of minimum at $\pm\frac{\pi}{2}$ $\implies$ $\mathcal{O}(\epsilon)$ angular error of child's solution.

Angular error in the parent can be converted to Euclidean error by considering the chord length between the mis-specified parent and the true parent direction. The two vectors plus the chord form an isoceles triangle with the relation that chord length $l = 2\sin(\frac{\epsilon}{2})$ is $\mathcal{O}(\epsilon)$ for $\epsilon \ll 1$.

Next, write the mis-specified parents as $\hat{v}_j = v_j + w_j$ where $||w_j||$ is $\mathcal{O}(\epsilon)$ as we have just shown. Let b equal the difference between the Riemannian update direction $\tilde{\nabla}^{\mu}_i$ with approximate parents and that with exact parents. All directions we consider here are the Riemannian directions, i.e., they have been projected onto the tangent space of the sphere. Then

$$b = \tilde{\nabla}^{\mu,e}_i - \tilde{\nabla}^{\mu}_i = (\underbrace{I - \hat{v}_i\hat{v}_i^\top}_{\text{projection onto sphere}})\sum_{j<i}\Big[(\hat{v}_i^\top\Sigma\hat{v}_j)\hat{v}_j - (\hat{v}_i^\top\Sigma v_j)v_j\Big] \tag{20}$$

and the norm of the difference is

$$\tag{21}$$

$$||b|| = ||(I - \hat{v}_i\hat{v}_i^\top)\sum_{j<i}\Big[(\hat{v}_i^\top\Sigma\hat{v}_j)\hat{v}_j - (\hat{v}_i^\top\Sigma v_j)v_j\Big]|| \tag{22}$$

$$\leq ||I - \hat{v}_i\hat{v}_i^\top|| \cdot ||\sum_{j<i}\Big[(\hat{v}_i^\top\Sigma\hat{v}_j)\hat{v}_j - (\hat{v}_i^\top\Sigma v_j)v_j\Big]|| \tag{23}$$

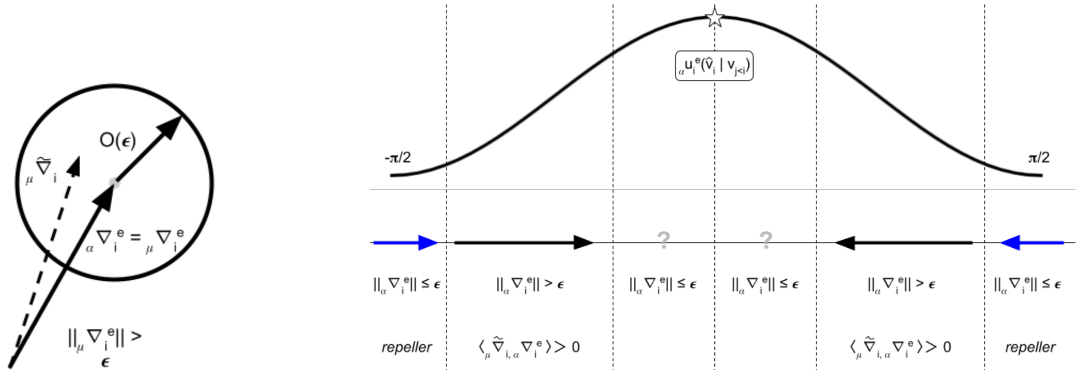
$$\leq ||\sum_{j<i}\Big[(\hat{v}_i^\top\Sigma\hat{v}_j)\hat{v}_j - (\hat{v}_i^\top\Sigma v_j)v_j\Big]||. \tag{24}$$

We can further bound the summands with

$$\begin{aligned}
||(\hat{v}_i^\top\Sigma\hat{v}_j)\hat{v}_j - (\hat{v}_i^\top\Sigma v_j)v_j|| &= ||(\hat{v}_j\hat{v}_j^\top - v_jv_j^\top)\Sigma\hat{v}_i|| && (25)\\
&\leq ||\hat{v}_j\hat{v}_j^\top - v_jv_j^\top||||\Sigma\hat{v}_i|| && (26)\\
&\leq \lambda_1||\hat{v}_j\hat{v}_j^\top - v_jv_j^\top|| && (27)\\
&= \lambda_1||(v_j + w_j)(v_j + w_j)^\top - v_jv_j^\top|| && (28)\\
&= \lambda_1||w_jv_j^\top + v_jw_j^\top + w_jw_j^\top|| && (29)\\
&\leq \lambda_1(||w_jv_j^\top|| + ||v_jw_j^\top|| + ||w_jw_j^\top||) && (30)\\
&= \mathcal{O}(\epsilon). && (31)
\end{aligned}$$

This upper bound on the norm of the difference between the two directions translates to a lower bound on the inner product of the two directions wherever $||\tilde{\nabla}^{\mu,e}_i|| > \epsilon$, specifically $\langle\tilde{\nabla}^{\mu,e}_i, \tilde{\nabla}^{\mu}_i\rangle > 0$ (see Figure 9a). And recall that the direction with exact parents is equivalent to the gradient of α-EigenGame with exact parents, $\nabla^{\alpha,e}_i$.

Therefore, by a Lyapunov argument, the $\tilde{\nabla}^{\mu}_i$ direction is an ascent direction on the α-EigenGame utility where it forms an acute angle (positive inner product) with $\nabla^{\alpha,e}_i$. Furthermore, $\nabla^{\alpha,e}_i$ is the gradient of a utility function that is sinusoidal along the sphere manifold; specifically, it is a cosine with period π and positive amplitude dependent on the spectrum of Σ (c.f. equation (8) of Gemp et al. (2021)). We can derive an upper bound on the size of the angular region for which $\tilde{\nabla}^{\mu}_i$ is


O(ϵ)
$_\mu\tilde{\nabla}_i$
$_\alpha\nabla_i^e = {}_\mu\nabla_i^e$
$||_\mu\nabla_i^e|| >$ ϵ
$_\alpha u_i^e(\hat{v}_i \mid v_{j<i})$
-π/2
π/2
?
?
$||_\alpha\nabla_i^e|| \leq \epsilon$
$||_\alpha\nabla_i^e|| > \epsilon$
$||_\alpha\nabla_i^e|| \leq \epsilon$
$||_\alpha\nabla_i^e|| \leq \epsilon$
$||_\alpha\nabla_i^e|| > \epsilon$
$||_\alpha\nabla_i^e|| \leq \epsilon$
repeller
$\langle {}_\mu\tilde{\nabla}_{i,\alpha}\nabla_i^e \rangle > 0$
$\langle {}_\mu\tilde{\nabla}_{i,\alpha}\nabla_i^e \rangle > 0$
repeller

(a)

(b)

Figure 9. (a) Close in Euclidean distance can imply close in angular distance if the vectors are long enough. (b) The stable region for μ-EigenGame consists of an $\mathcal{O}(\epsilon)$ ball around the true optimum as $\epsilon \to 0$.

not necessarily an ascent direction (the "?" marks in Figure 9). This region is defined as the set of angles for which the norm of the utility's derivative is small, i.e., $||\nabla_i^{\alpha,e}|| \leq \epsilon$. The derivative of cosine is sine, which depends linearly on its argument (angle) for small values, therefore, $|\theta| \leq \mathcal{O}(\epsilon)$ or $|\frac{\pi}{2} - \theta| \leq \mathcal{O}(\epsilon)$. As long as $\hat{v}_i$ does not lie within the $|\frac{\pi}{2} - \mathcal{O}(\epsilon)|$ region, μ-EigenGame will ascend the utility landscape to within $\mathcal{O}(\epsilon)$ angular error of the true eigenvector v_i. In the limit as $\epsilon \to 0$, the size of the $|\frac{\pi}{2} - \mathcal{O}(\epsilon)|$ region vanishes to a point, $v_i^\perp$. To understand the stability of this point, we can again appeal to the analysis from (Gemp et al., 2021). The Jacobian of $\tilde{\nabla}_i^\mu$ and the Hessian of u_i^α are equal with exact parents, and we know that the Riemannian Hessian is positive definite for distinct eigenvalues: $H_{\hat{v}_i}^R[u_i^\alpha] \succeq \min_j(\lambda_j - \lambda_{j+1})I$. This means that the point $v_i^\perp$ is a repeller for α-EigenGame. Similarly to before, we can show more formally that an $\mathcal{O}(\epsilon)$ perturbation to parents results in an $\mathcal{O}(\epsilon)$ perturbation to the Jacobian of $\tilde{\nabla}_i^\mu$ from $H[u_i^\alpha]$:

$$J = [I - \sum_{j<i} \hat{v}_j \hat{v}_j^\top]\Sigma \tag{32}$$

$$= [I - \sum_{j<i} (v_j + w_j)(v_j + w_j)^\top]\Sigma \tag{33}$$

$$= [I - \sum_{j<i} v_j v_j^\top]\Sigma - \sum_{j<i}[w_j v_j^\top + v_j w_j^\top + w_j w_j^\top]\Sigma \tag{34}$$

$$= [I - \sum_{j<i} v_j v_j^\top]\Sigma - \mathcal{O}(\epsilon)W \tag{35}$$

$$= H[u_i^\alpha] - \mathcal{O}(\epsilon)W \tag{36}$$

where W is some matrix with $\mathcal{O}(1)$ entries (w.r.t. ϵ). For the sphere, the Riemannian Jacobian is a linear function of the Jacobian ($J_{\hat{v}_i}^R = (I - \hat{v}_i\hat{v}_i^\top)J - (\hat{v}_i^\top J \hat{v}_i)I = H_{\hat{v}_i}^R[u_i^\alpha] - \mathcal{O}(\epsilon)$) and therefore the error remains $\mathcal{O}(\epsilon)$. The set of (non)symmetric, positive semidefinite matrices (A is p.s.d. iff $y^\top Ay \geq 0 \ \forall \ y$) forms a closed convex cone, the interior of which contains positive definite matrices (Wang et al., 2010). Therefore, $J_{\hat{v}_i}^R$ remains in this set after a small enough $\mathcal{O}(\epsilon)$ perturbation. Therefore, in the limit $\epsilon \to 0$, the spectrum of the Jacobian will also be positive definite indicating the point $v_i^\perp$ is a repeller. This is indicated by the blue arrows in Figure 9b.

Figure 9b summarizes the results that the stable region for α-EigenGame consists of an $\mathcal{O}(\epsilon)$ ball around the true optimum for parents with $\mathcal{O}(\epsilon)$ angular error. □

B. Jax pseudocode

For the sake of reproducibility we have included pseudocode in Jax. We use the Optax[7] optimization library (Hessel et al., 2020) and the Jaxline training framework[8]. Our graph algorithm is a straightforward modification of the provided

[7] https://github.com/deepmind/optax

[8] https://github.com/deepmind/jaxline

pseudo-code. See section C for details.

```
"""
Copyright 2020 DeepMind Technologies Limited.


Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

https://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
"""

import jax
import optax
import jax.numpy as jnp

def eg_grads(vi: jnp.ndarray,
                weights: jnp.ndarray,
                eigs: jnp.ndarray,
                data: jnp.ndarray) -> jnp.ndarray:
    """
    Args:
     vi: shape (d,), eigenvector to be updated
     weights:  shape (k,), mask for penalty coefficients,
     eigs: shape (k, d), i.e., vectors on rows
     data: shape (N, d), minibatch X_t
    Returns:
     grads: shape (d,), gradient for vi
    """
  weights_ij = (jnp.sign(weights + 0.5) - 1.) / 2.  # maps -1 to -1 else to 0
  data_vi = jnp.dot(data, vi)
  data_eigs = jnp.transpose(jnp.dot(data,
                            jnp.transpose(eigs)))  # Xvj on row j
  vi_m_vj = jnp.dot(data_eigs, data_vi)
  penalty_grads = vi_m_vj * jnp.transpose(eigs)
  penalty_grads = jnp.dot(penalty_grads, weights_ij)
  grads = jnp.dot(jnp.transpose(data), data_vi) + penalty_grads
  return grads

def utility(vi, weights, eigs, data):
    """Compute Eigengame utilities.
    util: shape (1,), utility for vi
    """
  data_vi = jnp.dot(data, vi)
  data_eigs = jnp.transpose(jnp.dot(data, jnp.transpose(eigs)))  # Xvj on row j
  vi_m_vj2 = jnp.dot(data_eigs, data_vi)**2.
  vj_m_vj = jnp.sum(data_eigs * data_eigs, axis=1)
  r_ij = vi_m_vj2 / vj_m_vj
  util = jnp.dot(jnp.array(r_ij), weights)
  return util
```

Listing 1. Gradient and utility functions.

```
def _grads_and_update(vi, weights, eigs, input, opt_state, axis_index_groups):
    """Compute utilities and update directions, psum and apply.
    Args:
     vi: shape (d,), eigenvector to be updated
     weights:  shape (k_per_device, k,), mask for penalty coefficients,
     eigs: shape (k, d), i.e., vectors on rows
     input: shape (N, d), minibatch X_t
     opt_state: optax state
     axis_index_groups: For multi-host parallelism https://jax.readthedocs.io/en/latest/_modules/jax/_src/lax/parallel.html
    Returns:
     vi_new: shape (d,), eigenvector to be updated
     opt_state: new optax state
     utilities: shape (1,), utilities
    """
    grads, utilities = _grads_and_utils(vi, weights, V, input)
    avg_grads = jax.lax.psum(
        grads, axis_name=’i’, axis_index_groups=axis_index_groups)
    vi_new, opt_state, lr = _update_with_grads(vi, avg_grads, opt_state)
    return vi_new, opt_state, utilities

def _grads_and_utils(vi, weights, V, inputs):
    """Compute utiltiies and update directions ("grads").
        Wrap in jax.vmap for k_per_device dimension."""
    utilities = utility(vi, weights, V, inputs)
    grads = eg_grads(vi, weights, V, inputs)
    return grads, utilities

def _update_with_grads(vi, grads, opt_state):
    """Compute and apply updates with optax optimizer.
        Wrap in jax.vmap for k_per_device dimension."""
    updates, opt_state = self._optimizer.update(-grads, opt_state)
    vi_new = optax.apply_updates(vi, updates)
    vi_new /= jnp.linalg.norm(vi_new)
    return vi_new, opt_state
```

Listing 2. EigenGame Update functions.

```
def init(self, *):
    """Initialization function for a Jaxline experiment."""
    weights = np.eye(self._total_k) * 2 - np.ones((self._total_k, self._total_k))
    weights[np.triu_indices(self._total_k, 1)] = 0.
    self._weights = jnp.reshape(weights, [self._num_devices,
                                          self._k_per_device,
                                          self._total_k])

    local_rng = jax.random.fold_in(jax.random.PRNGkey(seed), jax.host_id())
    keys = jax.random.split(local_rng, self._num_devices)
    V = jax.pmap(lambda key: jax.random.normal(key, (self._k_per_device, self._dims)))(keys)
    self._V = jax.pmap(lambda V: V / jnp.linalg.norm(V, axis=1, keepdims=True))(V)

    # Define parallel update function. If k_per_device is not None, wrap individual functions with vmap here.
    self._partial_grad_update = functools.partial(
        self._grads_and_update, axis_groups=self._axis_index_groups)
    self._par_grad_update = jax.pmap(
        self._partial_grad_update, in_axes=(0, 0, None, 0, 0, 0), axis_name=’i’)

    self._optimizer = optax.sgd(learning_rate=1e-4, momentum=0.9, nesterov=True)

def step(self, *):
    """Step function for a Jaxline experiment"""
    inputs = next(input_data_iterator)
    self._local_V = jnp.reshape(self._V, (self._total_k, self._dims))
    self._V, self._opt_state, utilities, lr = self._par_grad_update(
        self._V, self._weights_jnp, self._local_V, inputs, self._opt_state,
        global_step)
```

Listing 3. Skeleton for Jaxline experiment.

C. μ-EigenGame on Graphs

Algorithm 2 receives a stream of edges represented as a matrix with edges on the rows and outgoing node id (out) and incoming node id (in) as nonegative integers on the columns. The method $\texttt{zeros\_like}(z)$ returns an array of zeros with the same dimensions as z. The method $\texttt{index\_add}(z, idx, val)$ adds the values in array val to z at the corresponding indices in array idx with threadsafe locking so that indices in idx may be duplicated. Both methods are available in JAX. The largest eigenvector $\hat{v}_1$ is learned to estimate λ_1 and may be discarded. The bottom-k eigenvectors are returned by the algorithm in increasing order. Algorithm 2 expects $k+1$ random unit vectors as input rather than k in order to additionally estimate the top eigenvector necessary for the computation; otherwise, the inputs are the same as Algorithm 1.

Algorithm 2 μ-EigenGame for Graphs

```
Given: Edge stream $\mathcal{E}_t \in \mathbb{R}^{n' \times 2}$, minibatch size per partition $n''$, initial vectors $\hat{v}_i^0 \in \mathcal{S}^{d-1}$, step size sequence $\eta_t$, and iterations $T$.
$\hat{v}_i \leftarrow \hat{v}_i^0$ for all $i \in \{1, \ldots, k+1\}$
$\lambda_1 \leftarrow 2|\mathcal{V}|$ *upper bound on top eigenvalue*
for $t = 1 : T$ do
  parfor $i = 1 : k+1$ do
    parfor $t' = 1 : \frac{n'}{n''}$ do
      $[Xv]_i = \hat{v}_i(out_{tt'}) - \hat{v}_i(in_{tt'})$
      $[X^\top Xv]_i \leftarrow \texttt{zeros\_like}(\hat{v}_i)$
      $[X^\top Xv]_i \leftarrow \texttt{index\_add}([X^\top Xv], out_{tt'}, [Xv]_i)$
      $[X^\top Xv]_i \leftarrow \texttt{index\_add}([X^\top Xv], in_{tt'}, -[Xv]_i)$
      if $i = 1$ then
        $\lambda_1 \leftarrow ||[Xv]_i||^2$
        $\tilde{\nabla}^\mu_{it'} \leftarrow [X^\top Xv]_i$
      else
        $\tilde{\nabla}^\mu_{it'} \leftarrow \lambda_1[\hat{v}_i - \sum_{1<j<i}(\hat{v}_i^\top \hat{v}_j)\hat{v}_j]$
        $\tilde{\nabla}^\mu_{it'} -= [X^\top Xv]_i - \sum_{1<j<i}(\hat{v}_j^\top [Xv]_i)\hat{v}_j$
      end if
    end parfor
    $\tilde{\nabla}^\mu_i \leftarrow \frac{n''}{n'} \sum_{t'} [\tilde{\nabla}^\mu_{it'}]$
    $\hat{v}_i' \leftarrow \hat{v}_i + \eta_t \tilde{\nabla}^\mu_i$
    $\hat{v}_i \leftarrow \frac{\hat{v}_i'}{||\hat{v}_i'||}$
  end parfor
end for
return $\{\hat{v}_i | i \in \{2, \ldots, k+1\}\}$
```

D. Algorithm Design Process

In section 4.1, we presented u_i^μ as the Rayleigh quotient of a deflated matrix (repeated in equation (15) for convencience):

$$u_i^\mu = \hat{v}_i^\top \Big[\overbrace{I - \sum_{j<i} \hat{v}_j \hat{v}_j^\top}^{\text{deflation}} \Big] \Sigma \bullet[\hat{v}_i] \tag{37}$$

$$= \hat{v}_i^\top \Sigma \bullet[\hat{v}_i] - \sum_{j<i} (\hat{v}_i^\top \Sigma \hat{v}_j)(\bullet[\hat{v}_i]^\top \hat{v}_j) \tag{38}$$

$$u_i^\alpha = \overbrace{\hat{v}_i^\top \Sigma \hat{v}_i}^{\text{Var}} - \sum_{j<i} \overbrace{\frac{\langle \hat{v}_i, \Sigma \hat{v}_j \rangle^2}{\langle \hat{v}_j, \Sigma \hat{v}_j \rangle}}^{\perp\text{-penalty}}. \tag{39}$$

Alternatively, we can consider u_i^μ as equation (38) in light of the derivation for u_i^α by Gemp et al. (2021). In that case, utilities are constructed from entries in the matrix

$$\hat{V}^\top \Sigma \hat{V} = \begin{bmatrix} \langle \hat{v}_1, \Sigma \hat{v}_1 \rangle & \langle \hat{v}_1, \Sigma \hat{v}_2 \rangle & \dots & \langle \hat{v}_1, \Sigma \hat{v}_d \rangle \\ \langle \hat{v}_2, \Sigma \hat{v}_1 \rangle & \langle \hat{v}_2, \Sigma \hat{v}_2 \rangle & \dots & \langle \hat{v}_2, \Sigma \hat{v}_d \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \hat{v}_d, \Sigma \hat{v}_1 \rangle & \langle \hat{v}_d, \Sigma \hat{v}_2 \rangle & \dots & \langle \hat{v}_d, \Sigma \hat{v}_d \rangle \end{bmatrix}. \tag{40}$$

It is argued that if $\hat{V}$ diagonalizes M and captures maximum variance, then the diagonal $\langle \hat{v}_i, \Sigma \hat{v}_i \rangle$ terms must be maximized and the off-diagonal $\langle \hat{v}_i, \Sigma \hat{v}_j \rangle$ terms must be zero. As the latter mixed terms may be negative, the authors square the mixed terms to form "minimizable utilities" and divide them by $\langle \hat{v}_j, \Sigma \hat{v}_j \rangle$ so that they have similar "units" to the terms $\langle \hat{v}_i, \Sigma \hat{v}_i \rangle$ of the first type. In contrast, the u_i^μ utilities *could* be arrived at by instead multiplying the mixed terms by $\langle \hat{v}_i, \hat{v}_j \rangle$. While this ensures the mixed terms are positive with exact parents (because $\langle \hat{v}_i, \Sigma v_j \rangle = \lambda_j \langle \hat{v}_i, v_j \rangle$), it does not ensure they are always positive in general[9]. In other words, u_i^μ is defined in way such that the $\perp$-penalties actually encourage vectors to align at times when they should in fact do the opposite! We therefore consider it unlikely that anyone would pose equation (38) as a utility if coming from the perspective of α-EigenGame.

We could have extended the diagram in Figure 8 to include this dead end link. We have also included the true gradient of u_i^μ as a logical endpoint. We present these extensions in Figure 10.

$$\begin{tikzcd}
& u_i^\mu \arrow[dl, <->] & Eq.~(38) \arrow[d, -|, "\times" marking] & u_i^\alpha \arrow[d, <->, "\text{Var } \& \perp"'] \\
\nabla_i^\mu & \tilde{\nabla}_i^\mu \arrow[u, "🛑"'] & \square \arrow[l, "\text{remove}"] & \nabla_i^\alpha \arrow[l, "\text{bias}"]
\end{tikzcd}$$

Figure 10. This diagram presents the relationships between utilities and updates. An arrow indicates the endpoint is reasonably derived from the origin; the lack of an arrow indicates the direction is unlikely. The link from equation (38) is explicitly crossed out with a hard stop for emphasis.

D.1. Gradient Ascent on u_i^μ

If we remove the stop gradient 🛑 from equation (37), we are left with equation (41):

$$u_i^\mu = \hat{v}_i^\top [\overbrace{I - \sum_{j<i} \hat{v}_j \hat{v}_j^\top}^{\text{deflation}}] \Sigma \hat{v}_i. \tag{41}$$

If we then differentiate this utility, we find its gradient is

$$\nabla_i^\mu = \Sigma \hat{v}_i - \frac{1}{2} \sum_{j<i} [(\hat{v}_i^\top \Sigma \hat{v}_j) \hat{v}_j + (\hat{v}_i^\top \hat{v}_j) \Sigma \hat{v}_j]. \tag{42}$$

We also reran experiments with this update direction, ∇_i^μ on the synthetic and MNIST domains. The update is unbiased, so it would be expected to scale well, however, it (in orange) appears to scale more poorly than μ-EigenGame with smaller minibatches. Further research is needed to understand what makes the difference here.

[9] e.g., let $\Sigma = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$ and place $\hat{v}_1$ at -30° and $\hat{v}_2$ at 90°.

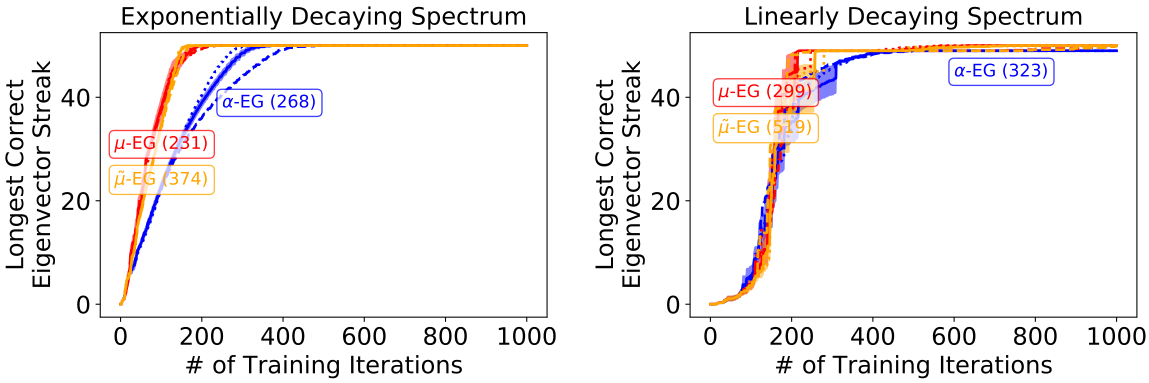


Figure 11. Synthetic Experiment. Runtime (milliseconds) in legend.

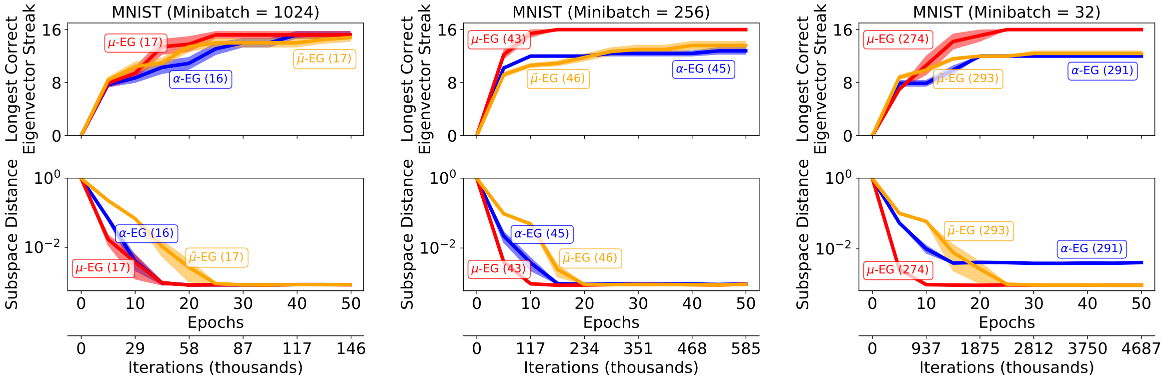


Figure 12. MNIST Experiment. Runtime (seconds) in legend. Each column evaluates a different minibatch size $\in \{1024, 256, 32\}$.