A Novel Stochastic Gradient Descent Algorithm for Learning Principal Subspaces

author names withheld

Under Review for OPT 2022

Abstract

In this paper, we derive an algorithm that learns a principal subspace from sample entries, can be applied when the approximate subspace is represented by a neural network, and hence can be scaled to datasets with an effectively infinite number of rows and columns. Our method consists in defining a loss function whose minimizer is the desired principal subspace, and constructing a gradient estimate of this loss whose bias can be controlled.

1. INTRODUCTION

Learning compact representations of data while minimizing information loss is at the heart of machine learning. A common approach for doing so is to learn a d-dimensional principal subspace that explains most of the variation in the data, what is known as principal component analysis (PCA). For small datasets, PCA can be accomplished by computing the singular value decomposition of the relevant data matrix. For sufficiently large datasets, however, this approach is impractical and one must instead turn to a stochastic or sample-based procedure.

Streaming PCA algorithms learn an approximate principal subspace by sampling columns from the data matrix Ψ and performing an incremental update that moves their approximation closer to the true subspace [e.g. 18, 19, 26, 31]. Central to these methods is the computation of the inner product between a full matrix column and the approximate subspace as well as a step to normalize the basis vectors parametrizing this subspace, making these methods most suited to problems where there are relatively few matrix rows. Another line of work learns the principal subspace as the by-product of a low-rank linear regression problem. In this case, the learner forms a product Φw_t where Φ encodes the approximate subspace and w_t is a per-column weight vector; the aim is to minimize the Euclidean distance between Φw_t and the column Ψ_t [22, 34, 35]. This approach has been effective for learning state representations in reinforcement learning [7, 10, 17], but can only handle a small number of columns, owing to the need to store an explicit weight vector for each.

In this paper, we propose a fully sample-based algorithm which exhibits the best of these two classes of approaches. Rather than maintain the weight vector w_t in memory, we instead estimate it on-the-fly from samples – effectively making the weight vector implicit. We use the weight vector estimate to construct a gradient of a suitable loss function, on which we perform stochastic gradient descent in order to determine an approximation to the d-dimensional principal subspace. Key to our approach is the derivation of the gradient in terms of Danskin's theorem. Although the naive plug-in gradient fails to be an unbiased estimate and can perform quite poorly in practice, an unbiased estimate is obtained by constructing two independent weight vector estimates. These estimates

are derived from a technique known as the LISSA [2] that produces a sequence of asymptotically-unbiased estimators of the inverse covariance matrix $(\Phi^\top \Phi)^\dagger$. Based on its origins, we call the result the *Danskin-LISSA* algorithm.

In Section 4, we show that our algorithm can recover the principal subspace of synthetic matrices and of MNIST images, while only observing a small subset of the data matrix at each update. We further demonstrate the effectiveness of our method for representation learning in reinforcement learning, specifically by learning a neural network-based approximation to the principal subspace of the successor measure [8] in the Puddle World domain [36].

2. BACKGROUND

2.1. Problem Statement

We consider a collection of column functions $\{\psi_t \in \mathbb{R}^S\}_{t\in\mathcal{T}}$ where $\mathcal{T}$ is an index set, and where each ψ_t maps row indices to real values. We assume that the column indices and the row indices are drawn i.i.d from a distribution λ on $\mathcal{T}$ and ξ on $\mathcal{S}$ respectively [1]. For a given integer $d \in \mathbb{N}$ and a *row representation* $\phi : \mathcal{S} \to \mathbb{R}^d$, we define the *representation loss*

$$L(\phi) = \mathop{\mathbb{E}}_{t\sim\lambda}\left[\min_{w_t\in\mathbb{R}^d} \mathop{\mathbb{E}}_{s\sim\xi}\left(\phi(s)^\top w_t - \psi_t(s)\right)^2\right]. \tag{1}$$

The representation loss describes the approximation error incurred by fitting the column function ψ_t with the d-dimensional linear approximation $\phi(s)^\top w_t$, on average over draws from λ. Here, we are interested in determining a d-dimensional representation ϕ that minimises $L(\phi)$ among all such representations.

For now, let us consider the case in which $\mathcal{S}$ and $\mathcal{T}$ are of finite sizes S and T, respectively. In this case, we may write $\Phi \in \mathbb{R}^{S\times d}$ for the *feature matrix* whose rows are $\{\phi(s)\}_{s\in\mathcal{S}}$ and $\Psi \in \mathbb{R}^{S\times T}$ for the data matrix whose columns are $\{\psi_t\}_{t\in\mathcal{T}}$. If additionally $W \in \mathbb{R}^{d\times T}$ is a weight matrix, then finding the function ϕ that minimizes Equation 1 is equivalent to jointly minimizing the loss $L(\Phi, W)$ over Φ and W, where

$$L(\Phi, W) = \|\Xi^{1/2}(\Phi W - \Psi)\Lambda^{1/2}\|_F^2. \tag{2}$$

Here, $\Xi \in \mathbb{R}^{S\times S}$ (resp. $\Lambda \in \mathbb{R}^{T\times T}$) is a diagonal matrix with entries $\{\xi(s) : s \in \mathcal{S}\}$ (resp. $\{\lambda(t) : t \in \mathcal{T}\}$) on the diagonal. For a given Φ, we write

$$W^* \in \mathop{\arg\min}_{W\in\mathbb{R}^{d\times T}} L(\Phi, W) \qquad L(\phi) = L(\Phi, W^*). \tag{3}$$

From standard linear algebra, in closed form we have

$$W^* = (\Phi^\top \Xi \Phi)^\dagger \Phi^\top \Xi \Psi. \tag{4}$$

Note that this expression does not depend on the column distribution Λ. We will use this matrix form to derive a gradient-based algorithm in the next section.

Equation 2 describes a weighted low-rank approximation problem [34]. Its solutions are the set of matrices Φ whose columns span the d-dimensional subspace of left singular vectors of Ψ. If in addition the columns of Ψ have mean zero, this corresponds to determining the subspace spanned by the d principal components of Ψ. Consequently, in the finite case our objective is to find a state representation whose implied feature matrix has columns that span this subspace.

1. We assume that $\xi(s) > 0$ for all row indices $s \in \mathcal{S}$ and that $\lambda(t) > 0$ for all column indices $t \in \mathcal{T}$.

3. PCA FROM SAMPLES

We assume access to a model from which we may repeatedly sample row indices according to the distribution ξ and the values taken on at those row indices by column functions sampled from λ. We are interested in the setting in which it is undesirable or impossible to sample the entire collection of column functions for a given state, or an entire column function all at once. This is different from the setting that approaches such as Oja's method [31] or the recent EigenGame [18] have considered for their experiments, which in matrix terms assume that it is possible to sample entire rows or columns from Ψ (for a longer discussion on prior work, see Section C).

Let us begin by expressing the gradient of the loss function $L(\Phi, W)$. In matrix form, this is

$$r \quad L(\Phi, W) = 2\Xi(\Phi W \quad \Psi)\Lambda W^\top, \qquad r_W L(\Phi, W) = 2\Phi^\top \Xi(\Phi W \quad \Psi)\Lambda \tag{5}$$

When the number of columns T is small, finding an optimal ϕ can be accomplished by optimizing the loss function $L(\Phi, W)$ using a nested or two-timescale optimization procedure based on unbiased estimates of these gradients. For example, the pair of update rules

$$\phi(s) \quad \phi(s) \quad \alpha(\phi(s)^\top w_t \quad \psi_i(s))w_t, \qquad w_t \quad w_t \quad \beta\phi(s)(\phi(s)^\top w_t \quad \psi_i(s)) \tag{6}$$

finds an optimal representation ϕ under suitable conditions on the step-sizes α and β. This is because the loss $L(\Phi, W)$ is convex in W when Φ is fixed and the two-timescale algorithm allows us to approximately run gradient descent on the objective we care about.

When T is large (or infinite), however, it is may be expensive (or impossible) to store a separate weight vector for each column. Instead, we rely on a form of the gradient of the loss $L(\phi)$ in which the weight vector is implicit.

Lemma 1 *Let $\beta > 0$ and $W^* = (\Phi^\top \Xi \Phi + \beta I)^{-1}\Phi^\top \Xi \Psi$. The loss $L : \mathbb{R}^{S \times d} \; ! \; \mathbb{R}$ defined by*

$$L(\Phi) = \min_{W \in \mathbb{R}^d \;\; T} \quad k\Xi^{1/2}(\Phi W \quad \Psi)\Lambda^{1/2}k_F^2 + \beta k W k_F^2$$

is continuously differentiable, with gradient $r \quad L(\Phi) = 2\Xi(\Phi W^* \quad \Psi)\Lambda W^{*\top}$

The idea is to use an instantaneous estimate of W^* to update the row representation in the negative direction of the (estimated) gradient of $L(\phi)$. As we will see, such an estimate can be obtained by sampling as little as a single column and a small number of rows. In effect, given a sample row index s our goal is to obtain a gradient estimate $\hat{g}(s)$ such that

$$\phi(s) \quad \phi(s) \quad \alpha\hat{g}(s) \tag{7}$$

should converge to an optimal representation under suitable conditions on the time-varying step-size α. In Subsection 4.3, we will discuss how Equation 7 can be applied to learn parametrized row representations such as those described by neural networks.

Before describing our approach, it is worth noting that the procedure that naively estimates W^* from a subset of rows and columns results in a biased gradient estimate. That is, suppose we are given the sample row indices $s, s', s_1, \ldots s_n$ and sample column t. If we write $\hat{\Phi}$ for the matrix whose rows are $\phi(s_1), \ldots, \phi(s_n)$ and construct the empirical covariance matrix $\hat{C} = \hat{\Phi}^\top \hat{\Phi}$, then we find that the estimate

$$\hat{g}_{\text{NAIVE}}(s) = \hat{w}_t \;\; \phi(s)^\top \hat{w}_t \quad \psi_t(s) \qquad \hat{w}_t = \hat{C}^\dagger \phi(s')\psi_t(s') \tag{8}$$

is not an unbiased estimate of $r_{\phi(s)} L(\Phi)$. In fact, the bias can be quite substantial when n is small, as we empirically show in Section 4.

3.1. An Improved Gradient Estimate

One issue with the estimate of Equation 8 is that the estimated weight vector $\hat{w}_t$ is itself a largely biased estimate of the optimal weight vector for column t (that is, the t^{th} column of W^*, $W^*_{:,t}$). Conversely, unbiasedness is obtained if $\hat{w}_t$ satisfies

$$\mathbb{E}[\hat{w}_t] = W^*_{:,t};$$

and if the term $w_t^\top$ is an independent, also unbiased estimate of $W^{*\top}_{:,t}$ in Lemma 1. To reduce the bias of the naive estimate, we will construct two low-biased estimates of the inverse covariance matrix $(\Phi^\top \Phi)^{-1}$, $\hat{C}$ and $\hat{C}'$, from which we derive two independent weight estimates $\hat{w}_t$ and $\hat{w}'_t$.

Before we explain how to obtain these estimates, let us describe our algorithm at a high level. We begin by drawing three row indices s, s', s'' and a column index t. We then construct the two weight estimates and then the gradient estimate

$$\hat{w}_t = \hat{C}(s', s)\,\delta_t(s') \qquad \hat{w}'_t = \hat{C}'(s'', s)\,\delta_t(s''); \qquad \hat{g}_{DL}(s) = \hat{w}'_t(s)\,\hat{w}_t - \delta_t(s) \tag{9}$$

which uses two LISSA estimators [2] to construct independent weight estimates by application of Danskin's theorem. In effect, using two separate weight estimates effectively allows us to estimate the outer product $W^*_{:,t} W^{*\top}_{:,t}$ appearing in Lemma 1 with a very low bias and hence obtain a gradient estimate that is overall low-biased, up to a multiplicative factor that we fold into the step-size parameter.

Theorem 2 Let $e_s \in \mathbb{R}^S$ denote a basis vector. Given two independent unbiased estimates $\hat{C}$ and $\hat{C}'$ of the inverse covariance, for $\eta < \frac{1}{\lambda_1^2}$, the gradient estimate $\hat{g}_{DL}(s)$ given in Equation (9) satisfies

$$\mathbb{E}[e_s \hat{g}_{DL}(s)^\top] = (I - \eta W^* W^{*\top}) W^*.$$

Note that the estimate $\hat{g}_{DL}(s)$ does not require the set of columns T to be finite. As such, our procedure can also be used to learn the principal components of infinite sets of columns; we will demonstrate this point in Subsection 4.3.

We derive the procedure which, given access to a stream of sample row representations $(s_j)_{j=1}^{\infty}$, asymptotically produces an unbiased estimate of the optimal weight vector for a given column in Appendix A and provide a detailed presentation of our algorithm in Appendix B.

4. EXPERIMENTS

We now conduct an empirical evaluation demonstrating that the Danskin-LISSA algorithm described in Section 3 recovers the d-dimensional principal subspace of different types of data: synthetic matrices, MNIST images [27] and the successor measure for the modified PuddleWorld domain [36]. In all cases, we measure convergence using the normalized subspace distance [37] between $\hat{W}$ and the principal subspace of W^*, $1 - \frac{1}{d}\,\mathrm{Tr}\,(F_d F_d^\top P) \in [0, 1]$. Here, F_d are the top d left singular vectors of W^* and $P = \Phi(\Phi^\top\Phi)^{-1}\Phi^\top$ is the orthogonal projector onto the column space of Φ. For simplicity, we take $M = N = J$ in all experiments. The parameter $\eta = \eta_0 / \max_{s \in s_{1:J}} \|\phi(s)\|_2^2$, where η_0 is a hyperparameter, is computed from the sampled feature vectors but we note that it can also be estimated online by a running average.

Figure 1: Subspace distance over the course of training LISSA for different dimensions (Left, $L = 25$) and for different total number of samples per update (Right, $d = 10$) on synthetic matrices with a spectrum decaying linearly and exponentially, averaged over 30 seeds. Shaded areas represent 95% confidence intervals.

Figure 2: (Left) Training curves for LISSA on MNIST ($d = 16$) that updates only a subset of pixels at a time. : see main text. (Right) Reconstruction on MNIST test images. First row show samples from test images. Second are images reconstructed from the true principal components of and third row are images reconstructed from the principal components learnt by Danskin-LISSA ($N = 64$). Reconstruction MSE errors for true components and Danskin-LISSA are 21.46 and 21.53 respectively.

4.1. Synthetic Matrices

To begin, we consider a random matrix $2 \mathbb{R}^{50 \times 50}$ whose entries are sampled from a standard normal distribution and We follow the experimental protocol from Gemp et al. [18] (See Appendix).

Figure 1, left illustrates that the Danskin-LISSA algorithm successfully recovers the d-dimensional principal subspace given suf ciently many training steps, with smaller values of d being easier to learn for the exponentially decaying spectrum (results for $d > 25$ are given in the appendix). However, we see that learning the subspace spanned by a representation of dimension $d = 25$ is easier than $d = 1$ for linearly decaying spectrum. Figure 1 right demonstrates that empirically, it is possible to obtain a reasonable approximation of the principal subspace even for a very smaller number of samples ($L = 1$ being the extreme), despite our theoretical expectation of a biased covariance estimate. As described in Section 3, the Danskin-LISSA approach stems from a combination of several algorithmic concepts (two independent estimates of the weight vectors, LISSA procedure). To understand better their relative importance in the performance of the Danskin-LISSA algorithm, we compare it to two sample-based baselines which have access to the same amount of information and memory (See Appendix). Figure 4 illustrates the bias-reducing advantage of the LISSA covari-

Figure 3: (Left) The Puddle World domain [36], with the shaded area indicating regions where the agent moves slowly. In our experiments, each grid cell is associated with a column of the implied data matrix. (Right) Subspace distance as a function of the dimension d after 10^8 gradient steps for three methods: Danskin-LISSA, Explicit, and the Large Batch baseline.

ance estimator, in particular in the low-sample regime. The naive method, which constructs a single weight estimate, has high bias and underperforms compared to both of these methods.

4.2. MNIST Dataset

We now consider learning the principal subspace from the MNIST training dataset with the Danskin-LISSA algorithm. Figure 2 shows that it is possible to effectively learn the principal subspace of this data even while updating as few as 32 pixels (rows) at a time; naturally, using more samples per step results in improved learning speed. As a point of comparison, we provide the subspace distance obtained by Eigengame [18], a state-of-the-art method that performs PCA by sampling full columns (images) at a time. To quantify the goodness of the representation learnt on the MNIST training set, we use it to reconstruct MNIST images on the test set. Figure 2, right, shows that the MNIST digits reconstructed from the subspace learnt by Danskin-LISSA qualitatively look similar to the images reconstructed from the true principal components of the training set and achieve a similar reconstruction error.

4.3. Learning the Successor Measure

In reinforcement learning (RL), the successor representation [13] encodes an agent's future trajectories from any given state in terms of the vistation frequency to various states. Of immediate relevance, it is often used as a building block in representation learning for RL, in particular by directly learning its principal subspace [6, 29, 30]. Its extension to continuous state spaces is called the successor measure [8], and is naturally described by an in nite dimensional matrix. Our last experiment illustrates how the Danskin-LISSA algorithm can be used to approximate the principal subspace of the successor measure of the Puddle World domain [36].

Compared to the experiments of the previous sections, we parametrize the representation by a neural network. We are interested in understanding the degree to which this neural network can be trained to approximate the d-dimensional principal subspace of the successor measure.

To gain an understanding of the effectiveness or our method, we compare it with two other gradient-based methods commonly used in reinforcement learning. As the name indicates, the Explicit method maintains a weight vector w_i for each column and relies on the pair of updates from Equation 6, similar to the method used by Bellemare et al. [7]. Note that we present this

method only for completeness, as it is not applicable to an in nite number of columns and may otherwise carry an impractically large memory cost. The Large Batch method, on the other hand, estimates the weight vector w_i using and evaluated at center of each of the 10,000 grid cells (close in spirit to the Naive method of Section 4.1).

Figure 3, right compares the nal subspace distance of these three algorithms for various values of d. We nd that the performance of the Danskin-LISSA algorithm degrades gracefully as d is increased, while the Large Batch method is only practical for small values of d. In part, this is explained by the fact that even with such a large batch, there is a residual bias in the latter method's covariance estimate. The poor performance of the Explicit method is explained by the fact that a single column is updated at any given time, resulting in stale weight vectors w_i. Although in practice this can be mitigated by updating multiple columns at once, the result illustrates an important pitfall with the use of an explicit weight vector.

5. DISCUSSION & CONCLUSION

In this paper, we presented an algorithm that learns principal components of very large or in nite dimensional matrices by stochastic gradient descent. Our experiments on synthetic matrices and MNIST images demonstrate that indeed the method converges towards their top principal subspace. Our analysis on the Puddle World domain also reveals that our algorithm works well when representing states with a neural network. In deep reinforcement learning (RL), training a network on supervised auxiliary predictions results in its representation corresponding to the principal components of this set of tasks, assuming the network is other unconstrained [7]. Incorporating the Danskin-LISSA procedure within a deep RL architecture may provide performance improvements by incorporating more knowledge about the world into the network's representation.

References

[1] Mart´n Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Zheng Xiaoqiang. Tensor ow: A system for large-scale machine learning. In USENIX Symposium on Operating Systems Design and Implementation, 2016.

[2] Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization in linear time. Journal of Machine Learning Research, 18:1–40, 2017.

[3] Ehsan Amid and Manfred K Warmuth. An implicit form of krasulina's k-PCA update without the orthonormality constraint. In Proceedings of the AAAI Conference on Arti cial Intelligence, 2020.

[4] Laura Balzano. On the equivalence of oja's algorithm and grouse. In Proceedings of the International Conference on Arti cial Intelligence and Statistics, 2022.

[5] Laura Balzano, Robert Nowak, and Benjamin Recht. Online identi cation and tracking of subspaces from highly incomplete information. In Annual Allerton Conference on Communication, Control, and Computing. IEEE, 2010.

[6] Bahram Behzadian and Marek Petrik. Low-rank feature selection for reinforcement learning. In ISAIM, 2018.

[7] Marc Bellemare, Will Dabney, Robert Dadashi, Adrien Ali Taiga, Pablo Samuel Castro, Nicolas Le Roux, Dale Schuurmans, Tor Lattimore, and Clare Lyle. A geometric perspective on optimal representations for reinforcement learning. Advances in Neural Information Processing Systems, 2019.

[8] Léonard Blier, Corentin Tallec, and Yann Ollivier. Learning successor states and goal-dependent values: A mathematical viewpoint. arXiv preprint arXiv:2101.07123, 2021.

[9] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, Skye Wanderman-Milne, and Qiao Zhang. Jax: composable transformations of python+ numpy programs. 2018. URL http://github.com/google/jax .

[10] Will Dabney, André Barreto, Mark Rowland, Robert Dadashi, John Quan, Marc G Bellemare, and David Silver. The value-improvement path: Towards better representations for reinforcement learning. In Proceedings of the AAAI Conference on Arti cial Intelligence, 2021.

[11] Wei Dai and Olgica Milenkovic. Set: An algorithm for consistent matrix completion. 2010 IEEE International Conference on Acoustics, Speech and Signal Processing, 2010.

[12] John M Danskin. The theory of max-min and its application to weapons allocation problems volume 5. Springer Science & Business Media, 2012.

[13] Peter Dayan. Improving generalization for temporal difference learning: The successor representation. Neural Computation, 5(4):613–624, 1993.

[14] Christopher De Sa, Christopher Re, and Kunle Olukotun. Global convergence of stochastic gradient descent for some non-convex matrix problems. In Proceedings of the International conference on Machine Learning, 2015.

[15] Zhijie Deng, Jiaxin Shi, and Jun Zhu. Neuralef: Deconstructing kernels by deep neural networks. In Proceedings of the International Conference on Machine Learning, 2022.

[16] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In Proceedings of the Conference on Learning Theory, 2015.

[17] Carles Gelada, Saurabh Kumar, Jacob Buckman, O r Nachum, and Marc G Bellemare. DeepMDP: Learning continuous latent space models for representation learning. In Proceedings of the International Conference on Machine Learning, 2019.

[18] Ian Gemp, Brian McWilliams, Claire Vernade, and Thore Graepel. EigenGame: Pca as a nash equilibrium. In Proceedings of the International Conference on Learning Representations, 2021.

[19] Ian Gemp, Brian McWilliams, Claire Vernade, and Thore Graepel. EigenGame unloaded: When playing games is better than optimizing. In Proceedings of the International Conference on Learning Representations, 2022.

[20] Moritz Hardt. Understanding alternating minimization for matrix completion. In 2014 IEEE 55th Annual Symposium on Foundations of Computer Science, pages 651–660. IEEE, 2014.

[21] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. Low-rank matrix completion using alternating minimization. In Proceedings of the ACM Symposium on Theory of Computing, 2013.

[22] Chi Jin, Sham M Kakade, and Praneeth Netrapalli. Provable efficient online matrix completion via non-convex stochastic gradient descent. Advances in Neural Information Processing Systems, 2016.

[23] Raghunandan H Keshavan and Sewoong Oh. A gradient descent algorithm on the grassman manifold for matrix completion. arXiv preprint arXiv:0910.5260, 2009.

[24] Raghunandan H Keshavan, Andrea Montanari, and Sewoong Oh. Matrix completion from a few entries. IEEE Transactions on Information Theory, 56(6):2980–2998, 2010.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Proceedings of the International Conference on Learning Representations, 2015.

[26] T Krasulina. Method of stochastic approximation in the determination of the largest eigenvalue of the mathematical expectation of random matrices. Automation and remote control, 2:50–56, 1970.

[27] Yann LeCun. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/, 1998.

[28] Xingguo Li, Junwei Lu, Raman Arora, Jarvis Haupt, Han Liu, Zhaoran Wang, and Tuo Zhao. Symmetry, saddle points, and global optimization landscape of nonconvex matrix factorization. IEEE Transactions on Information Theory, 65(6):3489–3514, 2019.

[29] Marlos C. Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. Eigenoption discovery through the deep successor representation. In Proceedings of the International Conference on Learning Representations, 2018.

[30] Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. Journal of Machine Learning Research, 8(10), 2007.

[31] Erkki Oja. Simplified neuron model as a principal component analyzer. Journal of mathematical biology, 15(3):267–273, 1982.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems 2019.

[33] David Pfau, Stig Petersen, Ashish Agarwal, David GT Barrett, and Kimberly L Stachenfeld. Spectral inference networks: Unifying deep and spectral learning. In *Proceedings of the International Conference on Learning Representations*, 2019.

[34] Nathan Srebro and Tommi Jaakkola. Weighted low-rank approximations. In *Proceedings of the International Conference on Machine Learning*, 2003.

[35] Ruoyu Sun and Zhi-Quan Luo. Guaranteed matrix completion via non-convex factorization. *IEEE Transactions on Information Theory*, 62(11):6535–6579, 2016.

[36] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 8, 1995.

[37] Cheng Tang. Exponentially convergent stochastic k-pca without variance reduction. *Advances in Neural Information Processing Systems*, 2019.

[38] Laurenz Wiskott and Terrence J Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural computation*, 14(4):715–770, 2002.

[39] Tian Ye and Simon S Du. Global convergence of gradient descent for asymmetric low-rank matrix factorization. In *Advances in Neural Information Processing Systems*, 2021.

Appendix A. Estimate of the Weight Vector $W_{;t}$

Central to our procedure is an estimate $\hat{A}$ of the inverse covariance matrix $(xx^\top)^\dagger$. We construct this estimate by embedding what is known as the LISSA estimator [2, originally used to estimate the Hessian inverse]. Our algorithm is parameterised by two scalars α and J, which trade off estimator variance with sample complexity. All proofs can be found in Appendix E

To begin, consider an arbitrary matrix $A \in \mathbb{R}^{S \times d}$ and denote $\|\cdot\|_{op}$ the spectral norm. For any $\alpha < \|A^\top\|_{op}^{-1}$, the Moore-Penrose pseudo-inverse $(AA^\top)^\dagger$ has a Neumann series expansion of the form

$$(AA^\top)^\dagger = \alpha \sum_{i=0}^{\infty} (I - \alpha AA^\top)^i. \tag{10}$$

Here, α is a scaling parameter that ensures the convergence of the series. Denoting the first j terms of the above series, we have that

$$S_j = \alpha I + (I - \alpha AA^\top) S_{j-1}.$$

We use this observation to build an estimator $(AA^\top)^\dagger$ with access to a finite number of samples from S.

Definition 3 (LISSA estimator) Let $A \in \mathbb{R}^{S \times d}$ be a feature matrix. Let $s_{1:J} = \{s_1, s_2, \ldots, s_J\}$ be J i.i.d. row indices sampled from A. Let $\alpha_0 \in (0, 2)$ and $\alpha = \alpha_0 \sup_{s_{1:J}} \|A(s_i)\|_2^{-2}$. The j-LISSA estimator $\hat{b}_j$ is recursively given by

$$\begin{aligned} \hat{b}_0 &= \alpha I \\ \hat{b}_j &= \alpha I + (I - \alpha A(s_j) A(s_j)^\top) \hat{b}_{j-1}, \quad 0 < j \leq J. \end{aligned} \tag{11}$$

Lemma 4 (Bias of LISSA) For $\alpha < \sup_{s_{1:J}} 2\|A(s_i)\|_2^{-2}$, the bias of $\hat{b}_j$ with respect to $(AA^\top)^\dagger$ is given by

$$\text{bias}(\hat{b}_j) = (AA^\top)^\dagger (I - \alpha AA^\top)^{j+1}$$

In particular, this bias asymptotically vanishes, in the sense that

$$\lim_{j \to \infty} \text{bias}(\hat{b}_j) = 0.$$

Appendix B. Algorithm Based on LISSA

Provided that we use the LISSA procedure twice to construct two independent estimates $\hat{w}_{t,1}^0$ and $\hat{w}_{t,2}^0$ of the optimal weight vector $W_{;t}$, it is straightforward to demonstrate that $\hat{g}_{DL}(s)$ (Equation (9)) becomes an unbiased estimate of the gradient of the loss as $J \to \infty$; furthermore, for finite J its biased is controlled in the sense of Theorem 2. We may then perform gradient descent with this estimate, adjusting the s^{th} row of the matrix W according to

$$W(s) \leftarrow W(s) - \eta \hat{g}_{DL}(s), \tag{12}$$

where $\in [0, 1)$ is a suitable step size. Based on our derivation, we call this procedure the Danskin-LISSA algorithm. In practice, it is usually desirable to update $N > 1$ rows at once and use $M > 1$ samples to estimate $\hat{w}_i$ and $\hat{w}_i'$; we give this more general form in Algorithm 1. Note that while larger values of J are desirable in order to reduce estimation bias, larger values of M and N contribute to reducing the variance of the gradient estimate $\hat{g}_{DL}$ and speeding up the learning process.

An important case is when the row representation is given by a mapping that is parametrized by a collection of weights , for example a neural network. In this case, Equation 12 should be replaced by an update rule that adjusts the weights . In practice, this can be done by determining the Jacobian $\frac{\partial}{\partial}$ of with respect to the weights , and applying the update

$$\frac{\partial}{\partial} \hat{g}_{DL}(s).$$

An alternative particularly suited to automatic differentiation frameworks [1, 9, 32], is to de ne a loss function whose gradient corresponds to $\frac{\partial}{\partial} \hat{g}_{DL}(s)$. One can verify that the sample loss function

$$\ell(w) = \frac{1}{2} \ell(\hat{w}_t + \hat{w}_t') \quad \ell(\hat{w}_t) \quad \ell(\hat{w}_t') \quad (s)^\top \, \mathrm{SG}(w) \quad {}_t(s)^2$$

satis es this requirement, where SG denotes the stop-gradient operation (in the sense that $\mathrm{SG}(w) = 0$). Additionally, the recursion in Equation 11 can be implemented ef ciently by rst computing the vector-matrix product $(s_j)^\top) \, b_{j-1}$ and then taking the outer product of the result with (s_j).

Algorithm 1 Danskin-LISSA

Parameters Dimension $d \in \mathbb{N}^+$, $J, M, N \in \mathbb{N}^+$, , $_0 \in (0, 2)$
repeat
  Sample independent rows $s_{1:N}$; $s'_{1:M}$; $s''_{1:M}$
  Sample a column
  $\hat{C}$ LISSA($_0$; J)
  $\hat{C}'$ LISSA($_0$; J)
  $w_t = \hat{C} \sum_{k=1}^{M} (s'_k) \, {}_t(s'_k)$
  $\hat{w}_t' = \hat{C}' \sum_{k=1}^{M} (s''_k) \, {}_t(s''_k)$
  $\hat{g}_{2\text{LISSA}}(s_k) = \hat{w}_t' \, (s_k)^\top \hat{w}_t \quad {}_t(s_k)$
  $(s_k) \quad (s_k) \quad \hat{g}_{DL}(s_k)$ for $k = 1, \ldots, N$
until satis ed

While for any nite value of J, the LISSA estimator b_j is not an unbiased estimate, Theorem 4 establishes that its bias can be made arbitrarily small with enough samples. In our experiments, we will show that this results in substantially better convergence compared to a naive estimate of the covariance matrix.

In Theorem 3, the parameter controls the rate of convergence of the full Neumann series: larger values of result in faster convergence, requiring fewer samples to obtain an estimate that has little bias with regards to the inverse covariance matrix. However, larger values of (which is bounded above as per Theorem 3) also produce estimators that have higher variance. Although here we consider the simplest setting in which a single sample is used at each iteration of Equation (11), the variance of the estimator can of course be reduced by using several samples per iteration.

Appendix C. RELATED WORK

Streaming PCA. Oja [31] and Krasulina [26] proposed the original streaming PCA algorithms. They approximate the top eigenvector of a matrix through a stochastic approximation of the power method. Tang [37] extends this method to other principal components but requires explicit normalization. Amid and Warmuth [3] extends it without the need to explicitly performing orthonormalization after each gradient step at the cost of a batch having to be of size d.

Pfau et al. [33] recovers the subspace spanned by the top eigenfunctions of symmetric infinite dimensional matrices by parametrizing them with neural networks and performing gradient descent on a kernel-based loss. It is itself a generalization of slow feature analysis [38] in the tabular setting. Deng et al. [15] extends the objective from Gemp et al. [18] to the function space and propose an algorithm to learn the top d-eigenfunctions of symmetric matrices by representing them with d neural networks. To find the principal subspace of a general infinite dimensional matrix, the approaches above require computing eigenfunctions of $\bar{F}$, which requires full row access to $\bar{F}$. By contrast, our method can recover the principal subspace of any infinite dimensional matrix using samples entries from rows of $\bar{F}$.

Low-rank matrix completion. In this setting, we observe a subset of entries from a data matrix and aim to find a low-rank matrix that matches these observations [34]. Matrix factorization is a common technique to solve this problem where the matrix of interest is expressed as a product W. It can be solved efficiently by standard optimization algorithms [35]. Hardt [20], Jain et al. [21] rely on alternating minimization over the representation and weight matrices and guarantee convergence towards the true matrix. Other methods perform gradient descent [28, 39] or stochastic gradient descent [14, 16, 22]. Keshavan and Oh [23], Keshavan et al. [24] minimize simultaneously over the representation and the weights W by gradient descent. Dai and Milenkovic [11] first solves the inner optimization problem and find the optimal weight matrix W and then takes a gradient step on the outer optimization problem, with respect to the representation matrix T. The Grassmannian Rank-One Subspace Estimation (GROUSE) algorithm [5] is a stochastic manifold gradient descent algorithm for tracking subspaces from incomplete data which was recently shown to be equivalent to Oja's algorithm [4]. In comparison, we propose an optimization procedure which performs gradient descent on the representation matrix only and where the weight matrix W is expressed implicitly, as a function of T.

Appendix D. Proofs for Section 2

Proposition 5 Denote $GL_d(R)$ the set of $d \times d$ invertible matrices. Assume $\operatorname{rank}(\bar{F}) < \infty$. Write $\bar{F} = F \Sigma B$ as the SVD of $\bar{F}$. For an integer $\ell \in \{1, \dots, S\}$, let $F_\ell \in \mathbb{R}^{S \times \ell}$ be the matrix containing the first ℓ columns of F (sorted by decreasing singular value). For any $d \in \{1, \dots, S\}$,

$$\arg\min_{T \in \mathbb{R}^{S \times d}} \min_{W \in \mathbb{R}^{d \times T}} \| T W - \bar{F} \|_F^2 = \{ T \in \mathbb{R}^{S \times d} \mid \exists M \in GL_d(\mathbb{R}),\ T = F_d M \} \tag{13}$$

Proof For a fixed $T \in \mathbb{R}^{S \times d}$ and if T is full rank, the unique solution of $\min_W \| TW - \bar{F} \|_F^2$ is given by $W = (T^\top T)^{-1} T^\top \bar{F}$. When T is orthonormal, we have $TW = T T^\top \bar{F}$. Moreover, $\operatorname{rank}(T T^\top \bar{F}) \leq \min(\operatorname{rank}(T), \operatorname{rank}(\bar{F})) = \min(d, \min(d, r)) = d$. By the Eckart-Young Theorem, the best rank d approximation of $\bar{F}$ with respect to the Frobenius norm is given by $F_d \Sigma_d B_d^\top = F_d (F_d^\top \bar{F})$. By identification, $T (T^\top \bar{F}) = F_d (F_d^\top \bar{F})$ and $T = F_d$ is a solution.

As we can turn the basis for (F_d) into any other basis $^0 = \quad R$ with $R \ 2 \ R^{d \ d}$ an invertible matrix, the set of solutions for is $f \ F_d R; R \ 2 \ R^{d \ d}$ invertible$g$ ■

Appendix E. Proofs for Section 3

Lemma 6 Let $> \ 0$ and $W \ = (\ ^{>} + \ I \ )^{\ 1} \ ^{>}$. The loss $L : R^{S \ d} ! \ R$ de ned by

$$L( ) = \min_{W \ 2 R^{d \ T}} k \ ^{1=2}(\ W \ ) \ ^{1=2} k_F^2 + \ kW k_F^2$$

is continuously differentiable, with gradient $L( ) = 2 (\quad W \quad) \ W^{>}$

Proof The proof is similar to the one of Danskin's theorem [12]. By linear algebra, the unique minimizer W in Equation (??) is given by Equation (??), which is itself differentiable with respect to . By the chain rule, we have

$$r \ L( ) = \ r \ L(\ ; W \ ) + \frac{@W}{@} \ ^{>} \frac{@}{@W} L(\ ; W \ ): \qquad (14)$$

Now, since W is de ned as the (unconstrained) minimizer of $L(\ ; W \ )$, its gradient with respect to the second argument vanishes at W, and so second term is zero. The result then follows from the de nition of $r \ L(\ ; W)$ in Equation (5). ■

Let $= \ E_s \ [e_s e_s^T]$ and $= \ E_t \ [e_t e_t^T]$.

Lemma 7 The j-LISSA estimator b_j is an unbiased estimator of the partial Neumann series de- ned in Equation(10). That is, given j samples $s_{1:j} = f s_1; s_2; :::; s_j g$ drawn i.i.d. from , we have that

$$E_{s_{1:j}} \ [b_j] = \sum_{i=0}^{j} (I \quad ^{>}) \ ^i$$

Proof By induction.

$E[b_0] = E[I \ ] = I$ and $\sum_{i=0}^{0} (I \quad ^T) \ ^i = I$

$E_{s_1} \ [b_1] = E_{s_1} \ [I \ + (I \quad _{s_1} \ ^T_{s_1}) I \ ] = I \ + \ (I \quad ^T) \ \text{as} E[\ _i \ ^T_i] = \ ^T$

and $\sum_{i=0}^{1} (I \quad ^T) \ ^i = I \ + \ (I \quad ^T)$

Let's suppose that $\mathbb{E}_{s_{1:j-1}}[b_{j-1}] = \sum_{i=0}^{j-1}(I - \eta N)^i$. Then,

$$\begin{aligned}
\mathbb{E}_{s_{1:j}}[b_j] &= \mathbb{E}_{s_{1:j}}[I + (I - \eta s_j s_j^T) b_{j-1}] \\
&= I + \mathbb{E}_{s_{1:j}}[(I - \eta s_j s_j^T) b_{j-1}] \\
&= I + \mathbb{E}_{s_j}[I - \eta s_j s_j^T]\mathbb{E}_{s_{1:j-1}}[b_{j-1}] \\
&= I + (I - \eta N) \sum_{i=0}^{j-1}(I - \eta N)^i \\
&= \sum_{i=0}^{j}(I - \eta N)^i
\end{aligned}$$

Hence, the conclusion. ■

Lemma 8 (Bias of LISSA) For $\eta < \sup_{s_{1:J}} 2\|\phi(s_i)\|_2^2$, the bias of b_j with respect to $(\lambda + \gamma)^{-1}$ is given by

$$\text{bias}(b_j) = (\lambda + \gamma)^{-1}(I - \eta(\lambda + \gamma))^{j+1}$$

In particular, this bias asymptotically vanishes, in the sense that

$$\lim_{j \to \infty} \text{bias}(b_j) = 0.$$

Proof

$$\begin{aligned}
\text{bias}(b_j) &= \mathbb{E}(b_j) - (\lambda + \gamma)^{-1} \\
&= \sum_{i=0}^{j}(I - \eta(\lambda + \gamma))^i - (\lambda + \gamma)^{-1} \text{ by Theorem 7} \\
&= (I - (I - \eta(\lambda + \gamma)))^{-1}(I - (I - \eta(\lambda + \gamma))^{j+1}) - (\lambda + \gamma)^{-1} \text{ using the closed form of a geometric se} \\
&= -(\lambda + \gamma)^{-1}(I - \eta(\lambda + \gamma))^{j+1}
\end{aligned}$$

■

Theorem 2 Let $e_s \in \mathbb{R}^S$ denote a basis vector. Given two independent unbiased estimates $\hat{C}$ and $\hat{C}'$ of the inverse covariance, for $s \in S$, the gradient estimate $\hat{g}_{DL}(s)$ given in Equation (9) satisfies

$$\mathbb{E}[e_s \hat{g}_{DL}(s)^T] = (\lambda + W)W^T.$$

Proof By definition,

$$\hat{g}_{DL}(s) = \sum_i \hat{w}_i' \phi(s)^T w_i \phi_i(s)$$

Plugging in $\hat{w}_i = \hat{C}\phi(s')\phi_i(s')$ and $\hat{w}_i' = \hat{C}'\phi(s'')\phi_i(s'')$, we have

$$\hat{g}_{DL}(s)^T = \phi(s)^T \hat{C}\phi(s')\phi_i(s)\phi_i(s)(\hat{C}'\phi(s'')\phi_i(s''))^T$$

Now taking the expectation,

$$
\begin{aligned}
E_{s;s^0;s^{00};s_{1:n};s^0_{1:n};i}[e_s \hat{g}_{DL}(s)^\top] &= E_{s;s^0;s^{00};s_{1:n};s^0_{1:n};i}\Big[e_s \quad (s)^\top \hat{C} \quad (s^0) \quad _i(s^0) \quad \quad _i(s) \quad (\hat{C}^0 \quad (s^{00}) \quad _i(s^{00}))^\top\Big] \\
&= E_{s;i}\Big[e_s \quad (s)^\top E_{s_{1:n}}[\hat{C}]E_{s^0}[\quad (s^0) \quad _i(s^0)] \quad \quad _i(s) \quad (E_{s^0_{1:n}}[\hat{C}^0]E_{s^{00}}[\quad (s^{00}) \quad _i(s^{00})])^\top\Big] \\
&= E_{s;i}\Big[e_s \; e_s^\top \quad E_{s_{1:n}}[\hat{C}]E_{s^0}[\quad ^\top e_{s^0}e_{s^0}^\top \quad e_i] \quad e_s^\top \quad e_i \quad (E_{s^0_{1:n}}[\hat{C}^0]E_{s^{00}}[\quad ^\top e_{s^{00}}e_{s^{00}}^\top \quad e_i])^\top\Big] \\
&= E_{s;i}\Big[e_s e_s^\top \quad E_{s_{1:n}}[\hat{C}]E_{s^0}[\quad ^\top e_{s^0}e_{s^0}^\top] \quad \quad e_i e_i^\top E_{s^{00}}[\quad ^\top e_{s^{00}}e_{s^{00}}^\top](\quad E_{s^0_{1:n}}[\hat{C}^0])^\top\Big] \\
&= E_s e_s e_s^\top \quad E_{s_{1:n}}[\hat{C}] \quad ^\top E_{s^0}[e_{s^0}e_{s^0}^\top] \quad \quad E_i[e_i e_i^\top] \quad ^\top E_{s^{00}}[e_{s^{00}}e_{s^{00}}^\top] \quad E_{s^0_{1:n}}[\hat{C}^0] \\
&= \quad E_{s_{1:n}}[\hat{C}] \quad ^\top \quad \quad (\quad ^\top (\quad E_{s^0_{1:n}}[\hat{C}^0])^\top)
\end{aligned}
$$

where in the last line, we used the fact that $E_s[e_s e_s^\top] = E_t[e_t e_t^\top]$. Now, given two unbiased estimators $\hat{C}$ and $\hat{C}^0$, we have

$$E_{s_{1:n}}[\hat{C}] = (\quad ^\top)^y \text{ and } E_{s^0_{1:n}}[\hat{C}^0] = (\quad ^\top)^y$$

It then follows that

$$
\begin{aligned}
E_{s;s^0;s^{00};s_{1:n};s^0_{1:n};i}[e_s \hat{g}_{DL}^\top(s)] &= \quad (\quad ^\top)^y \quad ^\top \quad \quad (\quad ^\top)(\quad ^\top)^y \\
&= \quad (\quad ^\top)^y \quad ^\top \quad \quad ((\quad ^\top)^y \quad ^\top) \quad ^\top \\
&= (\quad W \quad) \quad W^\top \\
&/ r \quad L(\quad)
\end{aligned}
$$

■

Appendix F. Additional Experimental Results

F.1. Synthetic matrices

In order to study our algorithm's behaviour under different conditions, we follow Gemp et al. [18] and set the matrix's singular values from 1000 to 1 linearly or exponentially We initialize $\in R^{50 \quad 50}$ randomly from a normal distribution. We compute its SVD such that $= F \quad B$. Let $_{linear} = (1; :::; 1000)$ and $_{exp} = (10^0; :::; 10^3)$. We rescale the matrix such that $_{linear} = F \quad _{linear}B$ and $_{exp} = F \quad _{exp}B$. The matrix $\in R^{S \quad d}$ is also initialized randomly from a standard normal distribution. We sweeped over the step size and chose $= 0:001$ which was working well in all the synthetic experiments. We used the SGD otpimizer but found that there was not a big performance difference with the Adam otpimizer [25] in most of these synthetic experiments. In Figure 6, we also sweeped over the hyperparameter and found that $_0 = 1:9$ was performing well across dimensions and for both linear and exponential spectra. We selected the parameter $_0 = 1:9$ from a hyperparameter sweep (Figure 6 compares performance for different values of $_0$, in particular illustrating how $_0 > 2$ fares poorly, according to our theory). We trained the Danskin-LISSA method for 10^6 time steps. As a complement to Figure 1, we show in Figure 5 the training curves of the Danskin-LISSA algorithm for a broader range of dimensions. For the

Figure 4: Subspace distance ($d = 10$) after 10^6 training steps according to the method used to estimate the loss gradient. Here, the x-axis represents the total number of row samples from the A matrix ($L = 2J + 2M + N$ for the Danskin methods, $J + M + N$ for the naive method). Shaded areas represent 95% confidence intervals. Note that because we are sampling with replacement, $L = 250$ still differs from the gradient given in Lemma 1.

Figure 5: Subspace distance over the course of training LISSA for different dimensions on synthetic matrices with a spectrum decaying linearly and exponentially, averaged over 30 seeds. The total number of samples used is 50. Shaded areas represent 95% confidence intervals.

exponential spectrum, when $d > 25$, larger dimensions are easier to learn. This is the opposite trend to the behavior found when $d \leq 25$ where smaller dimensions are easier to learn. For the linearly decaying spectrum, when $d > 25$, larger dimensions are easier to learn which is also the same trend as what we observed for $d \leq 25$.

In Figure 4, the first baseline uses the naive gradient estimator described in Section 3. The second uses two separate weight estimates, following the derivation from Danskin's theorem, but uses the inverse of the empirical covariance matrix rather than the LISSA procedure used in the Danskin-LISSA method – accordingly, we call this the Danskin-Empirical method.

F.2. MNIST

We represent the data as a matrix $A \in \mathbb{R}^{784 \times 60000}$ where each column is a 28×28 sample image (flattened to size 784) of one of the ten possible digits and from which the mean image has been subtracted. To accelerate learning speed we use the second-order Adam optimizer [25]. We per-

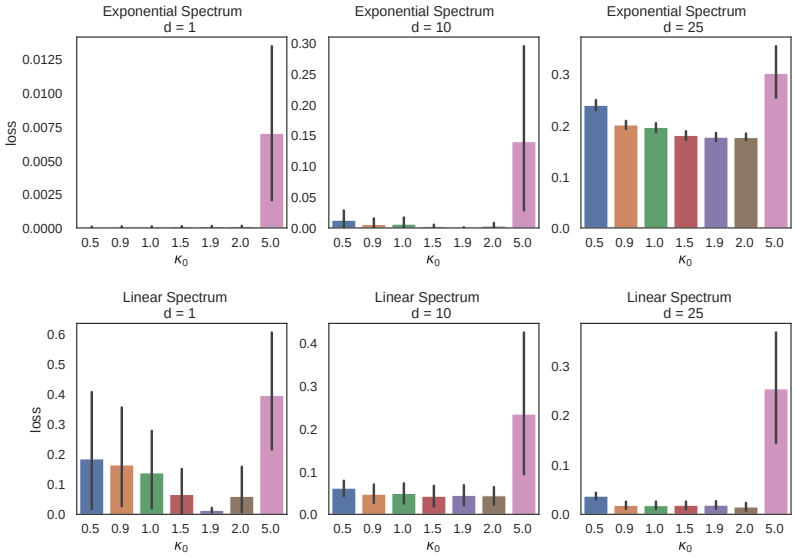

Figure 6: Subspace distance after 10^6 training steps of the LISSA algorithm for different κ_0

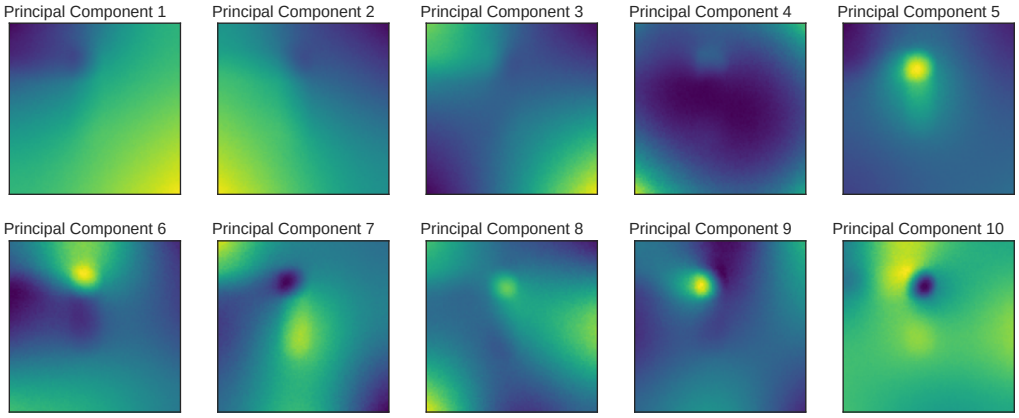

Figure 7: First 10 principal components of the successor measure of the Puddle World domain.

formed a sweep over the step-size α and found that $\alpha = 0.005$ worked best for 128 and 64 pixels. $\alpha = 0.01$ performed best for 32 pixels. We trained the Danskin-LISSA algorithm for $2.5 \quad 10^6$ steps.

In Figure 2 right, denoting Ψ_{test} *2* $\mathsf{R}^{784\times 10000}$ the test dataset and Φ *2* $\mathsf{R}^{784\times d}$ a representation learnt from the training set, the reconstructed images on the test set are given by $P \ \Psi_{\text{test}}$ where P denotes the orthogonal projector onto the column space of Φ.

F.3. Puddle World

In our version of this environment, traversing puddles requires more time, resulting in asymmetric successor measure; details of the environment and the reinforcement learning framework are given

below. Here, $s \in [0,1]^2$ corresponds to a particular two-dimensional state in the environment. For a collection of sets $X = \{X \subseteq [0,1]^2\}$ to be described below, we define the successor measure as

$$\Psi(s, X) = \sum_{t>0} \gamma^t \, \mathbb{P}\left(S_t \in X \mid S_0 = s\right), \quad \gamma \in (0,1)$$

The successor measure describes the expected, discounted number of visits to the set X when the agent begins in state s and moves randomly. We take $\gamma = 0.99$. We take the collection X to be the set of non-overlapping cells of a 100×100 grid (illustrated by Figure 3). For computational reasons, we assign the same value of $\Psi(\cdot, X)$ to all states within a grid cell; this value is computed by $1,000$ truncated Monte-Carlo rollouts from a start state sampled uniformly at random within a cell. This produces a $10,000 \times 10,000$ matrix which we treat as ground truth for measuring the accuracy of our predicted subspace. All three methods use Adam [25] to optimize a two-layer MLP with 512 hidden units and ReLU activations. We take $J = M = N = 50$ for Danskin-LISSA and $N = 250$ for the two other methods. The step size α was tuned for each method according to a small hyperparameter sweep and after 10^8 gradient steps averaged across 5 runs.

A Puddle World [36] is a square arena, with x, y both in [0, 1]. It has a continuous state space and a discrete action space. There are four actions (up, down, left, right) that move the agent by 0.05 in each of the corresponding directions. A random gaussian noise with standard deviation 0.01 is also added to transitions in both directions. For our experiments, we used the same puddle configuration found in [36]. This configuration contains two puddles. The first puddle lies between the points $(0.1, 0.75)$ and $(0.45, 0.75)$ with a radius of 0.1. The second puddle lies between the points $(0.45, 0.4)$ and $(0.45, 0.8)$, also with a radius of 0.1. While the original Puddle World gives negative rewards for being in a puddle, our puddles instead cause a slowing affect by a factor of 0.5. That is, when in a puddle, the agent only moves by 0.025 in each direction. The puddles compound, meaning that in the area where the two puddles overlap the agent will only move a distance of 0.0125. We chose to use slowing puddles because our task is reward-agnostic, and the successor measure task that we chose would capture the dynamics of the slowing puddles. We visualize in Figure 7 the top-10 principal components of the successor measure of Puddle World, demonstrating that they are non-trivial.

The successor measure was computed using 1000 Monte Carlo rollouts from each starting grid cell, truncated after 700 steps. We used a discount factor $\gamma = 0.99$. We subtracted the row sums to center-mean each column of the ground truth matrix $\Psi \in \mathbb{R}^{10^4 \times 10^4}$.

For each of the methods, we performed a sweep of learning rates and optimizers (between Adam and SGD) and found that Adam with a learning rate of 10^{-4} worked well across the board. We ran each method for 100 million gradient steps. For Danskin-LISSA, we kept κ fixed at 1.9, which we found worked well in our previous experiments. Danskin-LISSA used a batch size of 50 for each of its 5 batches, while Large Batch and Explicit used a main batch size of 250 to ensure that each method saw the same number of samples. To compute $\phi(s)$ we used a two hidden-layer MLP with 512 hidden units per layer.

Appendix G. Limitations

For simplicity, in this paper we assumed that all samples used in computing a given gradient estimate are drawn independently. In practice, samples are naturally expensive and it may appear

undesirable to require a total of $N + 2J + 2M$ for a single gradient estimate. However, one can improve on this state of affairs by permuting the order in which samples from the batch are presented, constructing different gradient estimates from these permutations, and noting that the average of multiple unbiased estimates remains unbiased (and generally has lower variance).