Spectral Inference Networks: Unifying Deep and Spectral Learning

David Pfau[1], Stig Petersen[1], Ashish Agarwal[2], David G. T. Barrett[1] and Kimberly L. Stachenfeld[1]

[1]DeepMind, London, United Kingdom [2]Google Brain, Mountain View, California pfau@google.com

Background: Spectral Learning

Spectral learning is any form of learning that uses a spectral decomposition (SVD, eig) to fit parameters, rather than gradient descent, EM, etc...

- Many classic algorithms in **manifold learning** are cases of spectral learning: Isomap (Tenenbaum et all 2000), LLE (Roweis and Saul 2000), Laplacian Eigenmaps (Belkin and Niyogi 2002), and Spectral Clustering (Ng et al 2000)
- Inference by **Nystrom approximation**, O(N) in size of training data (Bengio et al 2004)
- Spectral learning can also be used to learn **parametric** models like LDA, HMM, and mixture models (Anandkumar et al 2012, Hsu et al 2012, Hsu and Kakade 2013)
- Spectral manifold learning is similar in spirit to **self-supervised learning**: unsupervised learning of an embedding where the representation of one data point is easily predicted from the representation of a neighboring data point

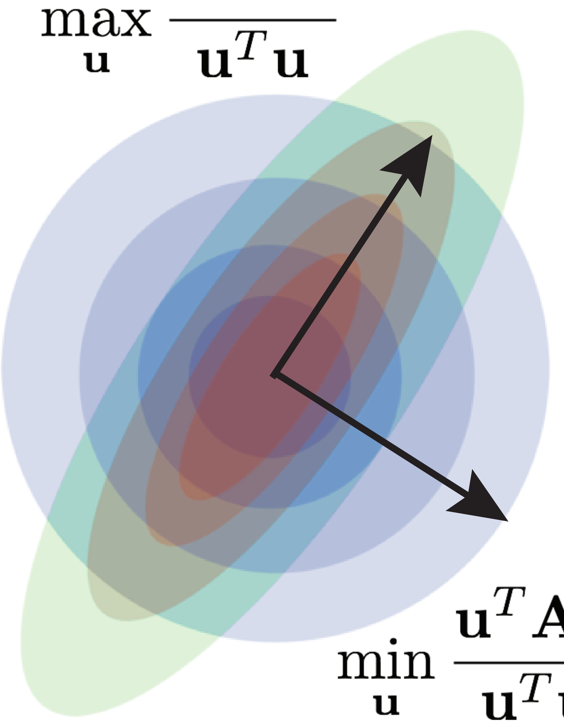
Many other classic ML paradigms have been adapted to deep learning:

- Density estimation ⟶ Deep generative models
- Variational Bayes ⟶ Variational Autoencoders
- Reinforcement learning ⟶ Deep reinforcement learning

Can we adapt spectral learning to the deep learning paradigm?
Can we do test-time inference in O(1) time wrt the size of the training data?

An Optimization View of Spectral Methods

$$\max_{\mathbf{u}} \frac{\mathbf{u}^T\mathbf{A}\mathbf{u}}{\mathbf{u}^T\mathbf{u}}$$



$$\min_{\mathbf{u}} \frac{\mathbf{u}^T\mathbf{A}\mathbf{u}}{\mathbf{u}^T\mathbf{u}}$$

The smallest eigenvalue of a symmetric matrix A is given by the minimum of the **Rayleigh quotient**:

$$\min_{\mathbf{u}} \frac{\mathbf{u}^T\mathbf{A}\mathbf{u}}{\mathbf{u}^T\mathbf{u}}$$

The Rayleigh quotient can be generalized from one eigenvector to many eigenvectors U = (u_1, u_2,...,u_k):

$$\min_{\mathbf{U}} \mathrm{Tr}\left(\mathbf{U}^T\mathbf{A}\mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\right)$$

To generalize from eigenvectors to eigenfunctions, replace matrix A with kernel *k* and sums over indices with expectations wrt a distribution p(x)

Linear function:

$$(\mathbf{A}\mathbf{u})_i = \sum_j A_{ij}u_j$$

Linear operator:

$$\mathcal{K}[u](\mathbf{x}) = \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}')}[k(\mathbf{x},\mathbf{x}')u(\mathbf{x}')]$$

Then the many-vector Rayleigh quotient generalizes to:

$$\min_{\mathbf{u}} \mathrm{Tr}\left(\mathbb{E}_{\mathbf{x},\mathbf{x}'}\left[k(\mathbf{x},\mathbf{x}')\mathbf{u}(\mathbf{x})\mathbf{u}(\mathbf{x}')^T\right]\mathbb{E}_{\mathbf{x}}\left[\mathbf{u}(\mathbf{x})\mathbf{u}(\mathbf{x})^T\right]^{-1}\right)$$

So why can't we just plug in a deep neural network for **u** and fit it by SGD?
Two problems:

1) The Rayleigh quotient for eigenfunctions is a function of the **inverse of an expectation**. Naively computing stochastic gradients from minibatches gives biased results.
2) The Rayleigh quotient is invariant to linear transformations of the output. We **can't separate the lower eigenfunctions from the higher ones**!

Spectral Inference Networks offer a solution to both of these problems.

Getting Stochastic Optimization to Converge

Simplify notation:

$$\mathbf{\Pi} = \mathbb{E}_{\mathbf{x},\mathbf{x}'}\left[k(\mathbf{x},\mathbf{x}')\mathbf{u}(\mathbf{x})\mathbf{u}(\mathbf{x}')^T\right] \quad \text{Quadratic constrained objective}$$

$$\mathbf{\Sigma} = \mathbb{E}_{\mathbf{x}}\left[\mathbf{u}(\mathbf{x})\mathbf{u}(\mathbf{x})^T\right] \quad \text{Covariance of the features}$$

Unconstrained objective

$$\min_{\mathbf{u}} \mathrm{Tr}\left(\mathbf{\Sigma}^{-1}\mathbf{\Pi}\right)$$

Gradient of objective

$$\mathrm{Tr}\left(\mathbf{\Sigma}^{-1}\nabla_\theta\mathbf{\Pi}\right) - \mathrm{Tr}\left(\mathbf{\Sigma}^{-1}\mathbf{\Pi}\mathbf{\Sigma}^{-1}\nabla_\theta\mathbf{\Sigma}\right)$$

Danger! Danger!

Use the **moving average** of $\mathbf{\Sigma}$ and $\nabla_\theta\mathbf{\Sigma}$ to reduce the bias of the gradients.
If we are careful about the learning rate schedules, this is an instance of **two timescale optimization**, and is guaranteed to converge to a local minimum (Borkar 1997).

Downside - need to track the full Jacobian of $\mathbf{\Sigma}$

Imposing an Ordering on the Embedding

The objective function is invariant to linear transformation of the output of the network - we can't separate the lower eigenfunctions from the higher ones!
For interpretable or disentangled features, we want a unique ordering of latent dimensions.

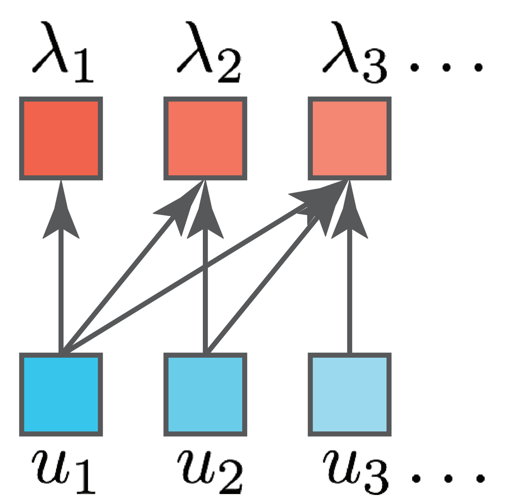
We can force a unique ordering on the different features by modifying the gradients.
Start by rewriting the objective in symmetric form:

$$\mathrm{Tr}\left(\mathbf{\Sigma}^{-1}\mathbf{\Pi}\right) = \mathrm{Tr}\left(\mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{\Pi}\right) = \mathrm{Tr}\left(\mathbf{L}^{-1}\mathbf{\Pi}\mathbf{L}^{-T}\right) = \mathrm{Tr}\left(\mathbf{\Lambda}\right) = \sum_i \lambda_i$$

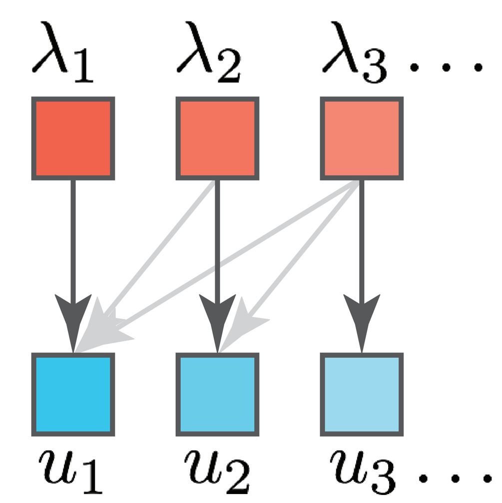
At the solution, the diagonal will exactly be the eigenvalues of the operator.

To impose an ordering on the eigenfunctions, we zero out the gradient going back from the higher eigenvalues to the lower eigenfunctions:

Forward Pass:



Backward Pass:



The entire masked gradient of all eigenvalues wrt parameters can be written in closed form:

$$\tilde{\nabla}_\theta\mathrm{Tr}(\mathbf{\Lambda}) = \mathbb{E}\left[k(\mathbf{x},\mathbf{x}')\mathbf{u}(\mathbf{x})^T\mathbf{L}^{-T}\mathrm{diag}(\mathbf{L})^{-1}\frac{\partial\mathbf{u}}{\partial\theta}\right] - \mathbb{E}\left[\mathbf{u}(\mathbf{x})^T\mathbf{L}^{-T}\mathrm{triu}\left(\mathbf{\Lambda}\mathrm{diag}(\mathbf{L})^{-1}\right)\frac{\partial\mathbf{u}}{\partial\theta}\right]$$

Slow Feature Analysis

Slow Feature Analysis (Wiskott and Sejnowski 2002) is a **special case** of Spectral Inference Networks with the kernel:

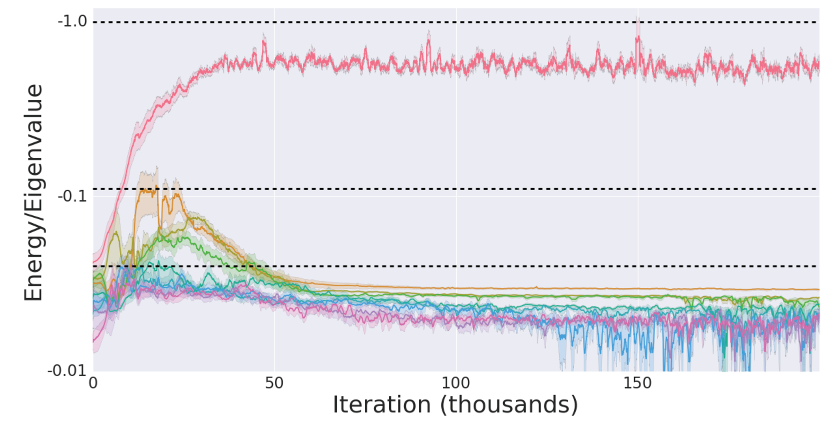
$$\mathbb{E}_{\mathbf{x},\mathbf{x}'}\left[k(\mathbf{x},\mathbf{x}')u(\mathbf{x})u(\mathbf{x}')\right] = \mathbb{E}_{\mathbf{x}_t}\left[||u(\mathbf{x}_t) - u(\mathbf{x}_{t+1})||^2\right]$$

Spectral Inference Networks provide a **fully end-to-end** way to fit Slow Feature Analysis with **generic function approximators**

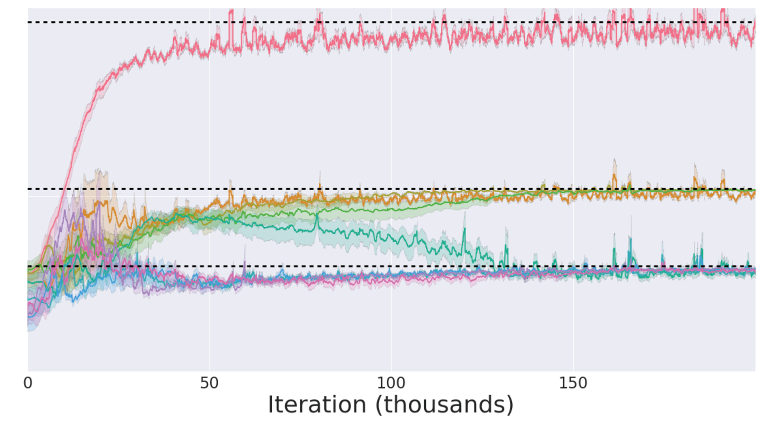
Experimental Results

Sanity check:
Schoedinger equation for two-dimensional hydrogen atom: $\mathcal{K}[u](\mathbf{x}) = -\nabla^2 u(\mathbf{x}) - \frac{u}{|\mathbf{x}|}$

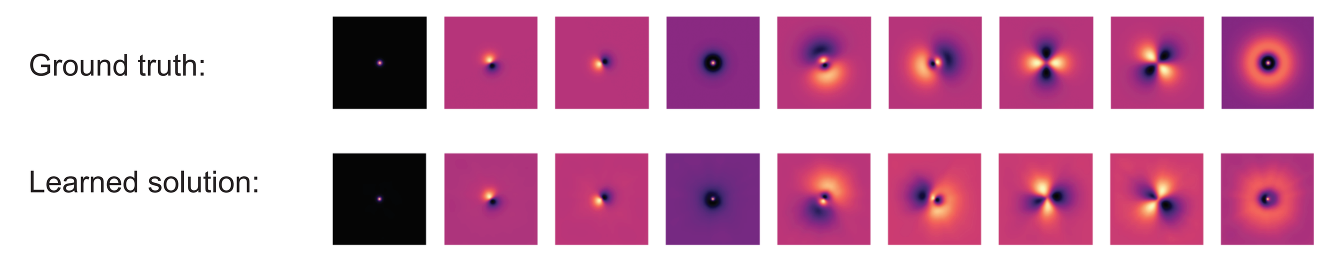
- Known **closed-form** solution: $\lambda = -1, -\frac{1}{4}, -\frac{1}{9}, \dots$
- Validate both convergence with small batches and correct ordering of eigenfunctions



Eigenvalues without moving average

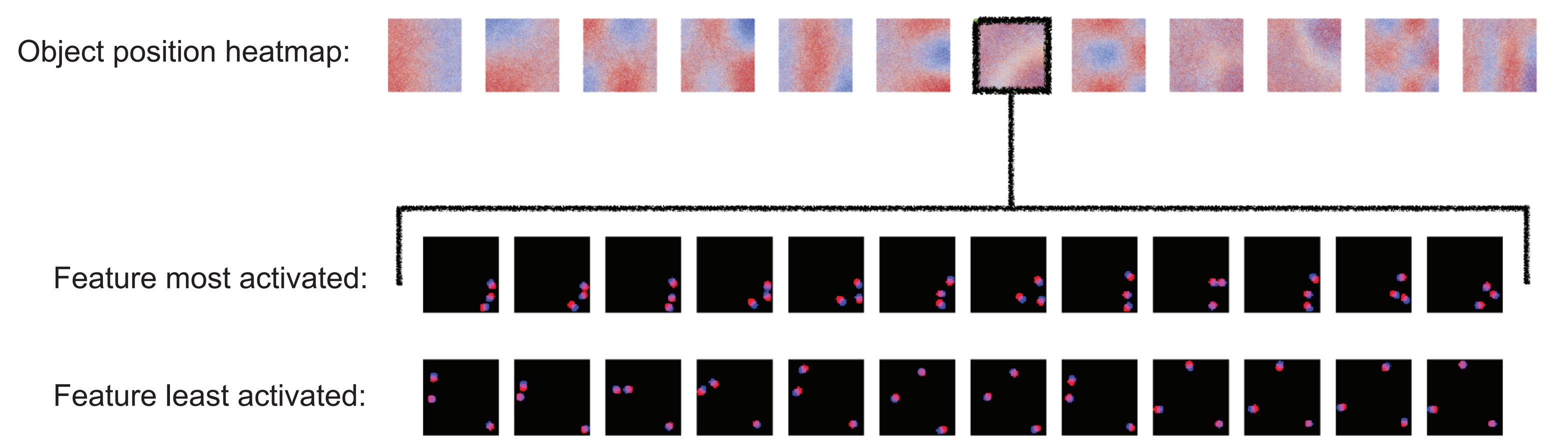


Eigenvalues with moving average



Video data:

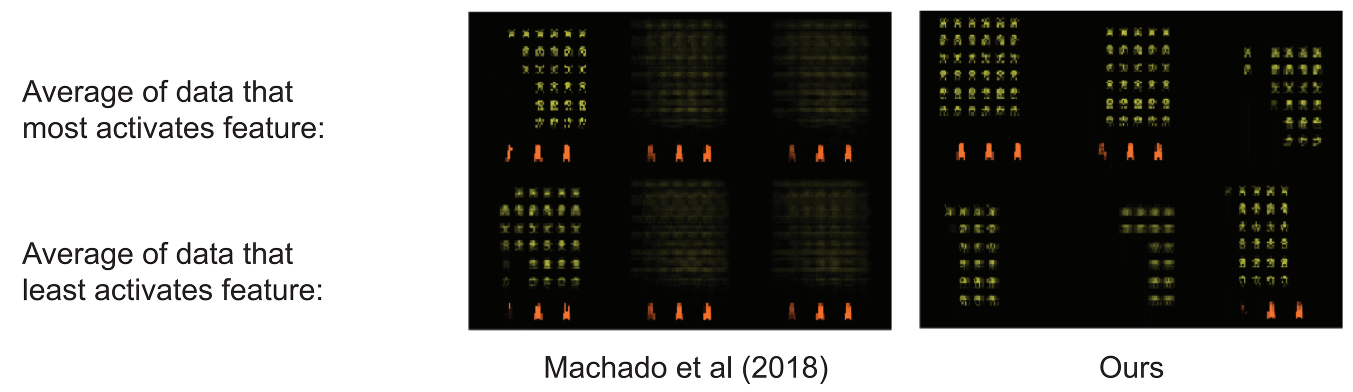
- Toy data: video of 3 bouncing balls
- Slow feature analysis kernel
- Ordinary convolutional network, nothing fancy about the architecture



Learned to recognize when 3 balls are all in one corner or two are in the opposite corner as the third - both an **interpretable** and **nonlinear** feature

Reinforcement Learning Applications: Eigenoption discovery

- Compute eigenfunction of transition operator, use as reward for options
- Training data: Atari games with random actions
- Same hyperparameters as for video data above
- We learn a representation much more discriminative for features of the environment



References

[1] J. Tenenbaum, V. de Silva and J. C. Langford, Science 2000
[2] S. T. Roweis and L. K. Saul, Science 2000
[3] M. Belkin and P. Niyogi, Neural Computation 2003
[4] A. Y. Ng, M. I. Jordan and Y. Weiss, NeurIPS 2002
[5] Y. Bengio, J. Paiement, P. Vincent, O. Dellaleau, N. L. Roux and M. Ouimet, NeurIPS 2004
[6] A. Anandkumar, D. P. Foster, D. Hsu, S. M. Kakade, Y. Liu, NeurIPS 2012
[7] D. Hsu, S. M. Kakade and T. Zhang, Journal of Computer and System Sciences 2012
[8] D. Hsu and S. M. Kakade, ITCS 2013
[9] V. S. Borkar, Systems and Control Letters 1997
[10] L. Wiskott and T. Sejnowski, Neural Computation 2002
[11] M. C. Machado, C. Rosenbaum, X. Guo, M. Liu, G. Tesauro and M. Campbell, ICLR 2018