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 Nyström 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
- 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 w.r.t the size of the training data?

An Optimization View of Spectral Methods

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

\[ \min_u \frac{u^T A u}{u^T u} \]

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

\[ \min_u \frac{u^T A U (U^T U)^{-1} u}{u^T u} \]

Linear function: \((A u)_i = \sum_j A_{ij} u_j\)
Linear operator: \(\mathcal{K}[u](x) = \mathbb{E}_{x' \sim p(x')} [k(x, x') u(x')]\)

Then the many-vector Rayleigh quotient generalizes to:

\[ \min_u \frac{\mathbb{E}_{x'} [k(x', x) u(x) u(x')^T] \mathbb{E}_x [u(x) u(x')^T]}{u^T u} \]

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, replace matrix A with kernel \( k \) and sums over indices with expectations wrt a distribution \( p(x) \)

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:

\[ \Pi = \mathbb{E}_{x,x'} [k(x, x') u(x) u(x')^T] \]
\[ \Sigma = \mathbb{E}_x [u(x) u(x')^T] \]

Quadratic constrained objective

\[ \min_u \text{Tr} (\Sigma^{-1} \Pi) = \text{Tr} (\Sigma^{-1} \Sigma^{1/2} \Pi \Sigma^{-1/2}) = \text{Tr} (\Lambda) = \sum_i \lambda_i \]

Covariance of the features

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:

Linear function:

\[ (A u)_i = \sum_j A_{ij} u_j \]

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

\[ \nabla_u \text{Tr}(\Lambda) = E \left[ k(x, x') u(x) L^{-T} \text{diag}(L^{-1}) \frac{\partial u}{\partial x'} \right] - E \left[ u(x) L^{-T} \text{tr}(A \text{diag}(L^{-1}) \frac{\partial u}{\partial x'}) \right] \]

Slow Feature Analysis

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

\[ \mathbb{E}_{x,x'} [k(x, x') u(x) u(x')] = \mathbb{E}_{x,x'} \| u(x) - u(x_{i+1}) \|^2 \]

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

Slow Feature Analysis

Spectral Inference Networks: Unifying Deep and Spectral Learning


Experimental Results

Sanity check:
Schoedinger equation for two-dimensional hydrogen atom:
• Known closed-form solution:
• Validate both convergence with small batches and correct ordering of eigenfunctions

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

Reinforcement Learning Applications: Eigenoption discovery
• Compute eigenfunction of transition operator, use as reward for options
• Same hyperparameters as for video data above
• We learn a representation much more discriminative for features of the environment

References