Convex Invariance Learning

Tony Jebara
Department of Computer Science
Columbia University
New York, NY 10027
jebara@cs.columbia.edu

Abstract

Invariance and representation learning are important precursors to modeling and classification tools particularly for non-Euclidean spaces such as images, strings and non-vectorial data. This article proposes a method for learning invariances in data while jointly estimating a model. The technique results in a convex programming problem with a consistent and unique solution. Representation variables are considered as affine transformations confined by multiple equality and inequality constraints. These interact individually with each datum yet maintain the overall solvability of the model estimation process while uniquely solving for the representational variables themselves. The method is applicable to various types of modeling, including maximum likelihood estimation, principal components analysis, and discriminative methods. Starting from affine invariance, several types of invariances are proposed and implemented as convex programs including clustering, permutation, selection, rotation, and translation. Experiments on non-vectorial data such as images and *collections of tuples* provide promising results.

1 Introduction

Recent progress in discriminative learning, support vector machines and regularization theory have encouraged the view that model estimation can be performed by minimizing a penalty function subject to e.g. linear classification constraints on the model [10] [4]. An important aspect of these formalisms is that their learning algorithms generate solvable convex programs with appealing computational properties. We propose a similar convex programming scheme for learning invariances or representations of the data. Instead of considering classification constraints on models, we consider constraints on a space of transformations. While traditional machine learning methods focus on estimating models (generative or discriminative) from vectorial data, non-vectorial data such as strings, images, audio and video require invariance and representation learning before interacting with a model. For example, permutational invariance of pixels in images is crucial to solve the so-called correspondence problem [2]. Therefore, the flexible specification of invariants and their reliable estimation is an imperative task of machine learning in these domains. Prior efforts in uncovering these representations involved iterative methods which were often difficult to embed jointly within a model estimation algorithm and also suffered from local minima. For instance, in [1] learning transformations for generative models requires an iterative EM or variational implementation and also discrete enumeration of all possible transforms. Similarly, iterative techniques such as *congealing* [7] can uncover image rotations yet these again have local minima and do not scale to, e.g. discriminative model estimation frameworks. Correspondence algorithms for image registration and alignment also have local minima problems and require relaxation or annealing [2].

In this paper we propose a novel approach to learning invariance and representation parameters via the Convex Invariance Learning (CoIL) framework. The text is organized as follows: we begin with motivation and a broad description of convex invariance learning. We then specifies various types of invariants as hulls of linear constraints. Subsequently, several traditional model estimation criteria are augmented to jointly perform invariance learning. We then discuss the collection of tuples representation for images, audio and other signals and the particular types invariance it requires. Implementation details are given for learning a model while estimating the required invariances. Finally preliminary experimental results are shown along with discussions.

2 Invariance

In this paper, we adopt the view that representation learning can be cast as learning with invariances and is performed by estimating the transformation operations on data points prior to their interaction with a model. A representation often implies (or is specified by) invariance properties in an object. For example, if we wish to represent a bag or collection of objects (i.e. a bag of vectors or pixels), then the ordering of the objects in the bag should be invariant. Permuting the order of the objects should not change the representation (i.e. permutational invariance). Another example is in vision where an image patch might need to be invariant to rotations and scaling of the image (i.e. affine invariance). Thus, in many cases, representations have built-in invariance properties. We will handle invariance in terms of allowable transformation operations upon an object.

We motivate invariance and representation learning using a simple example of a manifold estimation problem, such as principal components analysis (PCA). In Figure 1(a) we see a data set in $\Re^3$ which needs to be modeled by a lower dimensional manifold. Clearly, no appropriate manifold is apparent and PCA will not provide a useful result. However, if we generalize PCA and add invariants to the data, this is no longer the case (see Figure 1(b)). For instance, we may note an invariance property in our data and provide each datum with a path or manifold [1] that it may move along prior to forming the PCA subspace model. If points can be moved along their paths invariantly, they clearly form a two-dimensional subspace. Therefore, we can consider invariance learning as the estimation of transformations on the data (i.e. the paths around each datum) while simultaneously forming a model (i.e. the PCA subspace or another model estimation criterion).

2.1 Soft Invariance

An alternative to the above definition is *soft-invariance* where changes in a datum from its original setting incur a variable cost. This still permits us to consider a path or manifold of invariance around a point yet also balances the trade-off between the model's estimation and the excessive use of the invariance. For instance in Figure 1(c), the invariant paths are shown in fainter shades of gray at locations where the invariance incurs more cost and is in an unlikely configuration. Soft invariance can also relieve possible ambiguities in the model. We now formalize invariance

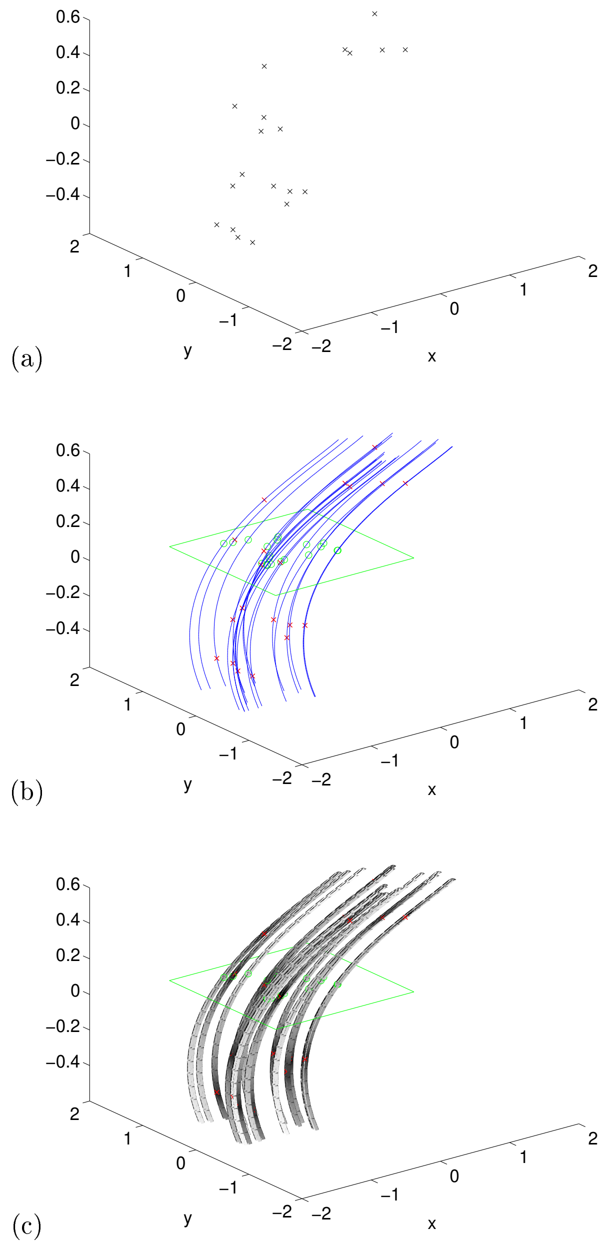


Figure 1: Invariant Manifold Learning

[1] A path is a 1-dimensional manifold. In general, we will consider many degrees of freedom in the invariance and describe possible configurations by a manifold.

and soft invariance within a computational framework.

2.2 A Convex Framework

The convex invariance learning framework begins with an input dataset of T vectors, $X_1, \ldots, X_T$ and *possibly* some corresponding labels $y_1, \ldots, y_T$. We endow each input data point with an affine transformation matrix A_t that interacts linearly with it as follows: $\sum_j A_t^{ij} X_t^j$. In general, we shall be solving for these matrices by minimizing a convex cost function $C(A_1, \ldots, A_T)$. In addition, these many transformation matrices $A = A_1, \ldots, A_T$ satisfy a set of linear equality/inequality constraints, giving rise to the following problem[2] definition:

$$\min_A C(A) \; subject\; to \sum_{ij} A_t^{ij} Q_{td}^{ij} + b_{td} \;\geq\; 0 \; \forall t, d \qquad (1)$$

Through the many imposed *linear* constraints on each of the A_t matrices[3], we obtain a convex hull on the A_t matrices. This convex hull on the A_t matrices in turn defines a region, manifold, or path of invariance around the datum as we span multiple valid settings of the A_t.[4] The above formulation is solvable via the variety of convex programming techniques, including duality methods[5] and, most importantly, has a unique solution. The general optimization picture that emerges is shown in Figure 2. In Section 3 we will discuss various strategies for designing these constraints on the affine matrices. Subsequently, in Section 4 we develop ways to compute $C(A)$ which will emerge from classical model learning algorithms acting upon the training data (X, y). To achieve the aforementioned soft invariance, we can apply a penalty function to each matrix which favors certain configurations over others within the valid convex space. If this penalty function is convex, then we can solve for the best setting of the affine matrices as a convex program by exploring the hull of constraints while minimizing a cost function plus penalties: $C(A) \leftarrow C(A) + \sum_t P_t(A_t)$. More generally, we can consider a single convex penalty function that involves all the matrices jointly.

At this point, we have circumvented the crucial model Θ estimation problem, which we will insert jointly in the above framework in subsequent sections. We next elaborate some of the types of useful invariance we can obtain from the affine matrices.

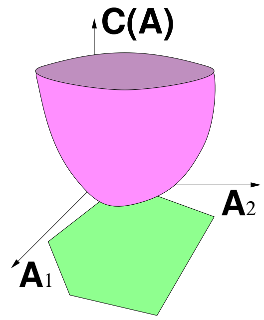


Figure 2: A Convex Program over the Transformation Matrices

3 Linear Invariance Constraints

While full affine invariance is often too general and leads to meaningless solutions (i.e. setting all $A_t = 0$ maximizes likelihood), by imposing various aggressive equality and inequality constraints as well as by preprocessing the data, we can obtain many types of meaningful invariances. For instance, it is trivial to use linear constraints to force sub components of the A_t affine transformations to be identical to each other or to limit their range and get a more controlled invariance space. Below, we list a number of linear restrictions that are useful for our particular applications.

- **Permutation and Correspondence** By limiting affine matrices (element wise or block wise) to be doubly-stochastic, i.e. $\sum_i A_t^{ij} = 1$, $\sum_j A_t^{ij} = 1$ and $A_t^{ij} \geq 0$ we approximate permutation matrices and permit the input space X_t to be broken down into a collection of tuples that can be resorted. Thus, an image vector acts as a collection of (x,y,intensity) tuples. Singly stochastic matrices are also possible. The permutation matrices are thus *soft* (i.e. not restricted to have $[0,1]$ entries). Examples of this type of invariance are shown in Section 5.

- **Permutation Sub-Groups** Consider a constant permutation matrix P applied multiple times to permute each X_t. Individual transformations for each datum differ: each is the constant matrix raised to a different power: $A_t = P^{f(t)}$. For P of size $D \times D$, only D powers are nonredundant. These D distinct configurations can be spanned by a $D-1$ dimensional convex hull (now over doubly-stochastic matrices) connecting the $P^1, P^2, \ldots, P^D$ entries linearly.

[2] More generally, nonlinear constraints may be used as long as they are convex.

[3] More generally, we can consider linear constraints that involve all the matrices jointly.

[4] In group theory, if the A_t matrices form a group, then the corresponding space of configurations of the X_t vectors they act upon is called an *orbit*. In the proposed framework, the A_t matrices may sometimes form a group but not necessarily so the term is not always appropriate.

[5] Dual space solutions may or may not be more efficient.

- **Clustering** To cluster data into M different centroids, concatenate the D-dimensional mean of the data $\bar{X}$ to each input $\tilde{X}_t \leftarrow [X_t; \bar{X}]$. Each A_t is constrained to be $M \times 2$ identity matrices, with each I scaled by a scalar α_{mn}. The α are positive and satisfy $\sum_m \alpha_{m1} = 1$, $\sum_m \alpha_{m2} = M - 1$ and $\sum_n \alpha_{mn} = 1$. Multiplying $\tilde{X}_t$ with A_t creates an MD-dimensional output vector placing the original input X_t in one of the M slots while copying the mean into the remaining empty ones.

- **Latent Variables** As in the clustering scenario, mixture models and latent variable models can be recast as transformations on the data. Latent variables typically index different components of a model, i.e. a mixture of Gaussians has a latent class variable or a hidden Markov model has a hidden state which selects the emission. We can mimic such latent variables by considering instead constrained transformations on the data itself, hence maintaining the convexity of the problem (latent variable maximum likelihood problems suffer from local minima). For instance, we may consider mimicking a hidden Markov model by allowing the input X to be a series of concatenated tuples which need to be assigned to one of M different Gaussian emission models. Unlike the clustering scenario above, additional constraints link temporally adjacent tuples together.

- **Selection** To attenuate or zero-out certain components of the D-dimensional input vector before it interacts with the model, use a diagonal A_t whose diagonal terms are positive and sum to a value less than D.

- **Translation** To translate data by a vector $X_t + Y_t$, concatenate unity to each X_t and limit the matrices to be partitioned into $A_t = [I \;\; Y_t]$.

- **Rotation and Scaling** Rotations R are a subset of affine matrices, yet require quadratic constraints to delimit (i.e. $RR^T = I$). However, a single rotation about a fixed axis can be approximated by linear constraints. This is a 2×2 matrix $[\cos\phi \;\, -\sin\phi \, ; \, \sin\phi \;\, \cos\phi]$ within an identity matrix which determines the axis of rotation. ϕ-dependent terms are constrained by equating the 2 diagonal terms and forcing off-diagonal terms to sum to zero and remain in $[-1, 1]$. This generates a rotation matrix with scaling.

In this article, we primarily explore the permutation case, deferring other transformations to later papers.

4 Convex Cost Functions

We next consider the model estimation process combined with the convex invariance framework to jointly learn the model and the representation variables.

4.1 Maximum Likelihood Gaussian Mean

For jointly performing model estimation while learning invariances, we first consider the simplest case of estimating a Gaussian mean (fixed covariance) with maximum likelihood. Maximizing log-likelihood $l(A, \mu) = \sum_t \log \mathcal{N}(A_t X_t; \mu, I)$ over the model gives the usual estimate for $\hat{\mu} = 1/T \sum_t A_t X_t$. Reinserting the mean estimate into the log-likelihood, we obtain:

$$l(A, \hat{\mu}) \;\; = \;\; -\frac{TD}{2}\log(2\pi) - \frac{1}{2}\sum_t \|A_t X_t - \hat{\mu}\|^2$$

We convert (via negation) the above likelihood into an equivalent cost function over the A_t matrices which is rewritten as the trace of the covariance of the data:

$$C(A) \;\; = \;\; trace(Cov(A\,X))$$

The trace of the covariance is a convex quadratic function of the affine matrices. Combined with linear constraints, this Gaussian mean estimator is directly solvable via a polynomial time quadratic program. In practice, however, we implement faster axis-parallel methods as in Section 6. The above criterion selects affine matrices that cluster data spherically, centering it towards a common mean.

4.2 Maximum Likelihood Gaussian Covariance

If we generalize to Gaussians of variable covariance as in $\mathcal{N}(AX; \mu, \Sigma)$ we can also impute the ML covariance estimate $\hat{\Sigma} = 1/T \sum_t (A_t X_t - \hat{\mu})(A_x X_t - \hat{\mu})^T$ to obtain:

$$\begin{aligned} l(A, \hat{\mu}, \hat{\Sigma}) \;\; = \;\; & -\frac{TD}{2}\log(2\pi) - \frac{T}{2}\log|\hat{\Sigma}| \\ & -\frac{1}{2}\sum_t (A_t X_t - \hat{\mu})^T \hat{\Sigma}^{-1}(A_t X_t - \hat{\mu}) \end{aligned}$$

After simplifications, the maximum likelihood setting of A is given by the equivalent cost function:

$$\tilde{C}(A) \;\; = \;\; |Cov(A\,X)|$$

We choose to equivalently minimize the logarithm of the above cost function $C(A) = \log|Cov(A\,X)|$ which shares the same optima. If we regularize the cost function by adding a small identity matrix to the covariance and adding a small $tr(Cov(A\,X))$ term (as in the

Gaussian mean case), we can also guarantee that this cost function is convex. More specifically, we have:

$$C(A) = \log|Cov(A\,X) + \epsilon_1 I| + \epsilon_2 tr(Cov(A\,X))$$

Both ϵ_1 and ϵ_2 are kept small (≈ 0.4).

We can prove convexity by only considering eigenvalues of the covariance (determinants and traces can be computed solely via eigenvalues). The log-determinant and trace terms become a summation over each eigenvalue or dimension separately. Since eigenvalues are convex quadratic functions of the data, each dimension we sum over has the following general form $f(x) = \log(x^2 + c) + bx^2$. The $f(x)$ functionsare each individually convex functions when $bc \geq 1/8$. Since the (regularized) log determinant of a covariance matrix is convex in the data, it is also convex in linear or affine matrices on the data such as the A_t. Therefore $C(A)$ is again convex and Equation 1 results in a convex program. However, it is not a quadratic program. We can instead minimize $C(A)$ by iteratively upper bounding using a quadratic function in A. This permits us to sequentially solve multiple quadratic programs interleaved with variational bounding steps until we converge to the global solution. First consider $\log|S|$ where we have defined $S = Cov(A\,X) + \epsilon_1 I$. The logarithm of the determinant is concave over covariance matrices [3]. Since $\log|S|$ is concave, we can upper bound it with a tangential linear function in S that is equal and has the same gradient $R = S_0^{-1}$ at the current setting of $S = S_0$ which is computed from our current setting of our matrices, $A = A_0$. The upper bound is then:

$$\log|S| \leq trace(RS) + \log|S_0| - tr(RS_0)$$

Adding our additional regularizer term with ϵ_2 to the above, we obtain the following upper bound on $C(A)$:

$$\begin{aligned} C(A) \leq\ & trace(R(Cov(A\,X) + \epsilon_1 I)) + \log|S_0| \\ & - tr(RS_0) + \epsilon_2 tr(Cov(A\,X)) \end{aligned}$$

Simplifying the bound by removing terms that are constant over A, we the following surrogate cost to minimize:

$$\begin{aligned} \tilde{C}(A) &= tr(MCov(A\,X)) \\ where\ M &= (Cov(A\,X) + \epsilon_1 I)^{-1} + \epsilon_2 I \end{aligned}$$

We thus update M for the current setting of the A_t matrices (by computing the covariance of the data after each A_t is applied to each X_t), then lock it for a few iterations while we minimize the trace to update the A parameters. Updates of M are interleaved with updates of the A matrices until convergence. The above criterion attempts to cluster data ellipsoidally such that it forms a low-dimensional sub-manifold. It is well known that the determinant of a covariance matrix behaves like a volumetric estimator and approximates the volume of the data. Minimizing volume by varying the constrained affine matrices is a valuable preprocessing step for PCA since it concentrates signal energy into a smaller number of eigenvalues, improving the effectiveness and reconstruction accuracy in the PCA subspace. Therefore, this criterion attempts to flatten the data via transformations such that it *forms as flat and low-dimensional a subspace as possible.*

4.3 Fisher Discriminant

Fisher's discriminant finds a w vector that maximizes $\frac{w^T U w}{w^T S w}$ to linearly separate labeled input data in a binary classification task. Here $U = (\mu_+ - \mu_-)(\mu_+ - \mu_-)^T$ computes the variance between the class means. Meanwhile $S = \Sigma_+ + \Sigma_-$ is the sum of the covariances of each class. Evidently increasing the distance between the means U while decreasing the within-class scatter S improves the separation. Therefore, to estimate good affine matrix parameters prior to computing the Fisher discriminant model, we minimize:

$$\tilde{C}(A) = |\Sigma_+ + \Sigma_- - \lambda(\mu_+ - \mu_-)(\mu_+ - \mu_-)^T|$$

Optimizing the logarithm of the above with a regularized of the covariance (as was done earlier) can again produce a convex cost function as long as the λ scalar is chosen to maintain a positive determinant. Once again, the above $C(A) = \log\tilde{C}(A)$ is convex as the previous in Gaussian covariance since the affine parameters appear quadratically within the determinant. The above criterion discriminatively estimates affine matrices that cluster data of a given class while repelling the means of the two classes. In later work, we will elaborate large margin criteria for convex invariance learning via support vector machine [10] and maximum entropy discrimination frameworks [4].

5 Collection of Tuples

We now discuss the case where the A_t are constrained to be permutation matrices. This form of invariance is crucial for so-called *collections of tuples.* A gray-scale image, for instance, can be represented as a collection of XYI tuples (x-coordinate, y-coordinate and intensity value). While traditional appearance-based representations of images generate a single vector of concatenated intensity values on the image $(I_1 \ldots I_N)$, such a representation ignores important properties in the data structure such as the inherent spatial proximity of certain pixels, and so forth. Furthermore, vector-based representations of images can produce highly nonlinear behavior under, for instance, simple translation of the imagery. It is possible to recover translation by considering nonlinear operators on the image vectors [1] or by having a finely sampled space of data and using local metric-based representations [9]. However, translation of XYI tuples is more immediate and merely involves adding a scalar value to all the X entries or all the Y entries and gives rise to linear behavior in this representation. [6]

While the collection-of-tuples representation is convenient in principle, it cannot directly be mapped into into vector form since tuples are in no particular order. Furthermore, it would be inappropriate to enforce an arbitrary ordering on what is effectively a *bag* of tuples. Instead, a flexible permutation matrix can be applied to an arbitrary ordering of the tuples to permit us to estimate the ordering before interacting with, e.g. a Gaussian model. This block-wise permutation matrix A merely resorts the tuples in each image prior to forming a vector, i.e. $Y = AX$. Using a dataset of several images where each has its own A matrix, this framework can effectively resort all the pixels of the image such that a common registration or correspondence[2] between image pixels is established permitting a much better final model (i.e. a Gaussian fit or a Fisher discrimination). For instance, images of faces should get aligned and registered such that pixels corresponding to the nose map to the same position in the ordering in the Y vector, while pixels corresponding the left eye map to a another consistent position in the ordering.

Similarly, other forms of data can also be viewed as a collection of tuples. Audio spectrograms have an amplitude and frequency value for each band. Therefore, one can consider them to be a collection of AF tuples. Sequence data as well can be treated as a collection of tuples. For example a uni-dimensional time series $x(t)$ can be written down as a collection of XT tuples (X value and time) permitting translation in time.

6 Implementation

The cost functions so far considered basically minimize a constrained trace of a covariance:

$$trace(MS) = \frac{1}{T}\sum_{mpnqi} A_i^{mn} A_i^{pq} X_i^q M^{pm} X_i^n - \frac{1}{T^2}\sum_{mpnqij} A_i^{mn} A_j^{pq} X_j^q M^{pm} X_i^n$$

While all the transformations listed earlier are compatible with the above cost function, we focus here on the permutation matrix case for handling collections of tuples. In implementing the above cost functions, it is evident that certain degenerate solutions may arise since we approximate permutation with doubly-stochastic matrices. For instance, A_t's entries might all be a low-valued constant, $c = 1/M$ which averages out all the entries in the vector.

One possible approach to avoiding degenerate solutions is to force the A_t matrices to only have binary elements yet this violates the convex program framework. Interestingly, minimizing the cost over binary A_t matrices is a min-cut problem. This optimization approach will be deferred for the moment. Instead, we will encourage the convex program to estimate *hard* doubly-stochastic matrices by adding a quadratic penalty function to the above cost which is $-\lambda \sum_{imn}(A_i^{mn} - c)^2$. The λ is chosen adaptively to maintain the convexity of the overall cost.

Other simplifications are possible. For instance, in the determinant minimization, we may lock the Gaussian mean estimate to be one of the original data points, i.e. $\mu = X_i$ for the i'th image and also lock its corresponding permutation matrix to identity, i.e. $A_i = I$. Since we are solving for a correspondence we can lock the particular ordering of a single datum (or image) without losing any effective flexibility in the estimation. This helps avoid the aforementioned degenerate solution where all permutation matrices become softened to a constant and average out the tuples. We next elaborate an update rule for monotonically minimizing the cost function in an axis-parallel manner.

6.1 Axis-Parallel Optimization

To minimize the cost function with constraints, many methods are possible, including quadratic programming. We follow the SMO approach [8] and vary a single A_t matrix for a single datum (the t'th one) at a time. We only update 4 of its entries $(A_t^{mn}, A_t^{mq}, A_t^{pn}, A_t^{pq})$ while all others are locked. Even though we ultimately estimate each scalar in each A_t matrix, only 4 scalars are updated at a time. Double-stochasticity gives the equality constraints: $A_t^{mn} + A_t^{mq} = a$, $A_t^{pn} + A_t^{pq} = b$, $A_t^{mn} + A_t^{pn} = c$ and $A_t^{mq} + A_t^{pq} = d$. So only one degree of freedom is left to compute *per iteration* and is updated as in Figure 3. The operations involve computing all possible inner products between the X-tuples X_n and X_q weighted by all relevant sub-matrices M_{mm}, M_{mp}, M_{pp} and M_{pm}. For clarity, here the matrices and vectors are indexed with subscripts instead of superscripts and all entries where the datum index does not appear refer to the datum at the t'th index.

We also limit A_t^{mn} to satisfy the inequalities $A_t^{mn} \in [\max(0, a-d, c-1), \min(a, c, 1+a-d)]$. After updating

[6] Appearance-based models treat images as a vector of intensities alone. These tediously represent translation and morphing by addition and deletion of intensity values.

$A_t^{mn} \leftarrow \frac{NUM}{2DEN}$
$NUM = cX_n^T M_{mp} X_n + cX_n^T M_{pp} X_q + 2aX_q^T M_{pm} X_q + cX_n^T M_{pm} X_n + aX_n^T M_{mm} X_q - 2aX_q^T M_{mm} X_q - 2cX_n^T M_{pp} X_n - aX_n^T M_{mp} X_q - cX_n^T M_{pm} X_q - aX_n^T M_{pm} X_q - X_q^T M_{mp} X_q d - X_n^T M_{pp} X_q d + X_n^T M_{pp} X_q a + X_n^T M_{mp} X_q d - X_q^T M_{pm} X_q d + 2X_q^T M_{pp} X_q d - 2X_q^T M_{pp} X_q a + 2aX_q^T M_{mp} X_q + H1 - H3 + H4 - H2 - cX_q^T M_{mp} X_n - aX_q^T M_{mp} X_n - aX_q^T M_{pm} X_n + aX_q^T M_{mm} X_n + cX_q^T M_{pp} X_n + X_q^T M_{pm} X_n d - X_q^T M_{pp} X_n d + X_q^T M_{pp} X_n a + 4a\lambda + 2c\lambda - 2d\lambda$
$DEN = X_q^T M_{mp} X_q - X_n^T M_{mm} X_n - X_q^T M_{mm} X_q - X_n^T M_{pp} X_n + X_n^T M_{pm} X_n - X_q^T M_{pp} X_q + X_n^T M_{mm} X_q - X_n^T M_{pm} X_q + X_n^T M_{mp} X_n - X_n^T M_{mp} X_q + X_n^T M_{pp} X_q + X_q^T M_{pm} X_q - X_q^T M_{pm} X_n - X_q^T M_{mp} X_n + X_q^T M_{pp} X_n + X_q^T M_{mm} X_n + 4\lambda$
$H1 = (X_n^T M_{um}^T + X_n^T M_{mu})(\sum_{u,v \neq \{mn,mq,pn,pq\}} A_{uv} X_v - \frac{1}{T-1} \sum_{u,v,\tau \neq t} A_{\tau,uv} X_{\tau,v})$
$H2 = (X_n^T M_{up}^T + X_n^T M_{pu})(\sum_{u,v \neq \{mn,mq,pn,pq\}} A_{uv} X_v - \frac{1}{T-1} \sum_{u,v,\tau \neq t} A_{\tau,uv} X_{\tau,v})$
$H3 = (X_q^T M_{um}^T + X_q^T M_{mu})(\sum_{u,v \neq \{mn,mq,pn,pq\}} A_{uv} X_v - \frac{1}{T-1} \sum_{u,v,\tau \neq t} A_{\tau,uv} X_{\tau,v})$
$H4 = (X_q^T M_{up}^T + X_q^T M_{pu})(\sum_{u,v \neq \{mn,mq,pn,pq\}} A_{uv} X_v - \frac{1}{T-1} \sum_{u,v,\tau \neq t} A_{\tau,uv} X_{\tau,v})$

Figure 3: Update Rule for Permutation Invariance

A_t^{mn}, we update the other 3 entries via their linear dependence on A_t^{mn}. Iterating the update rule randomly over different entries and matrices while intermittently recomputing bounds (via the inverse M) converges monotonically to the global minimum of $C(A)$.

Simplifications can be used to speed up convergence. One example is online estimation of the A_t matrices, i.e. optimizing them one at a time incrementally and locking them. Another example is initializing the optimization of the A_t by forcing the initial M matrix to be identity and then replacing it with the proper inverse covariance for subsequent initializations.

If a new (test) datum X_{T+1} is observed after training, it can be added to the cost function and we reperform a re-estimation of all $A_1 \ldots A_{T+1}$ matrices. However, for efficiency, it is also reasonable to lock all previous matrices A_t for $t = 1..T$ which converged during training and only estimate the final corresponding A_{T+1} matrix. This efficiently approximates the desired solution for a new datum in an online manner.

Alternatively, once the affine matrices A minimize the determinant, it is natural to apply PCA on the covariance matrix since the data has been flattened into a subspace. Recall that minimizing the determinant reduces the volume of the data and will typically result in a lower dimensional manifold representation. If k eigenvectors are maintained and sufficiently represent the covariance, we may perform a simpler optimization to obtain A_{T+1} for the new datum. This is done by *aligning* the new datum to the eigenspace by choosing the affine matrix A_{T+1} that minimizes the squared error of the reconstruction of the datum. If the eigenspace is composed of eigenvectors $V_1, \ldots, V_k$, this requires minimizing the following quadratic cost subject to double-stochasticity constraints:

$$A_{T+1} = \arg\min_A \| \sum_k (V_k^T A X_{T+1}) V_k - A X_{T+1} \|^2$$

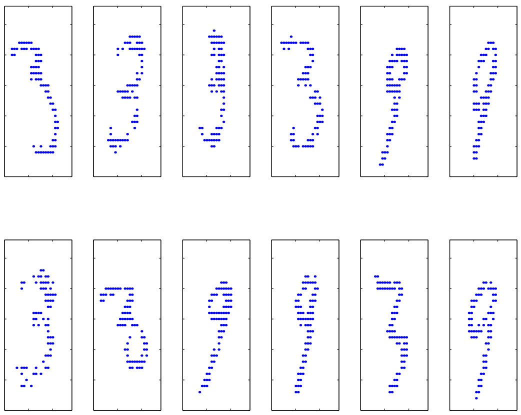
This is again more efficient with a slightly worse approximation to the true A_{T+1} that would have resulted from retraining with the new datum.

7 Experiments

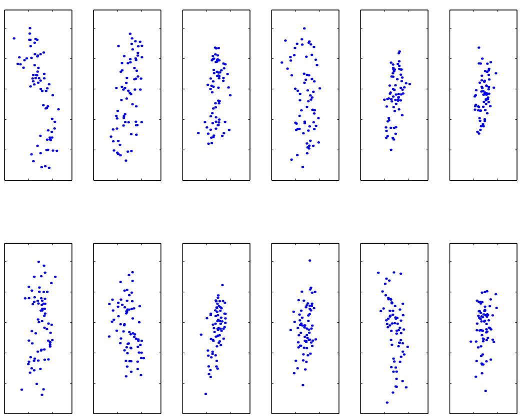
To test the framework we used 3 different data sets: point images (XY) of digits, intensity images (XYI) of faces and spectrograms (AF) on audio.

7.1 Point Image Representation

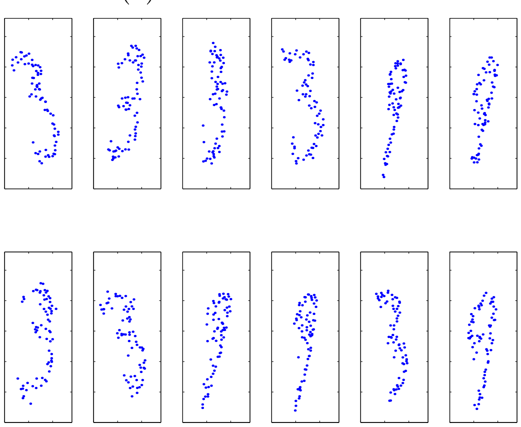
In the first dataset, we obtained 28x28 gray scale images of the digits 3 and 9 which were then represented as a collection of 70 XY pixels by sampling the region of high intensity. This effectively generates clouds of 2D points shaped in the form of 3's or 9's. A total of 20 such images was collected with 70 (x,y) pixels each and used to estimate the A_t matrices. Figure 4(a) depicts 12 exemplars of the original image data as point clouds. In Figure 4(b), PCA's reconstruction on the collection of tuples is shown. Finally, in Figure 4(c), PCA (with the same number of eigenvectors) was applied to the collection of tuples after estimation of the permutation matrices. Note that the images are reconstructed more faithfully in the latter case due to the estimation of the permutation or correspondence. Unlike the PCA eigenvectors which are do not resolve correspondence, CoIL eigenvectors seem to translate and morph the digits smoothly.



(a) Original images



(b) PCA reconstruction



(c) CoIL reconstruction

Figure 4: Reconstruction of digit images via PCA with and without permutation estimation.

7.2 Intensity Image Representation

One major difficulty with using large datasets involves storing the A_t matrices which, for a collection of N pixels each requires $O(N^2)$ scalars to define the transformation matrix. An alternative formulation to representing doubly-stochastic matrices can be found in [6] where each matrix is stored as $2N$ scalars and estimated with a statistical physics approximation called the *Invisible Hand* algorithm. For larger datasets, we modified the above framework to utilize this alternative formulation which converges more quickly and requires less storage. Roughly 300 grayscale images of faces were collected and sampled to form a collection of 2000 XYI tuples (sampling was constrained to the face using a simple skin-color distribution that ignores the background). The face images were of a single individual's face as it spans many lighting, 3D pose and expression configurations. Once the permutation transformation matrices were computed, we found an eigenspace of 20 components over the dataset and reconstructed the images from 20 coefficients alone. Figure 5 depicts the accuracy of the reconstructed faces when PCA (20 eigenvectors) alone was used as well as when the PCA-variant which performs permutation estimation. The images show much higher fidelity when permutations are also estimated. The CoIL XYI eigenvectors act smoothly, rotating and morphing the face in 3D as well as changing its illumination [5]. Traditional appearance-based PCA causes *ghosting* effects where translated images do not move smoothly but, instead, appear to fade in and out.

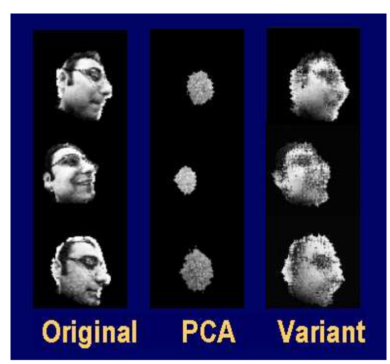
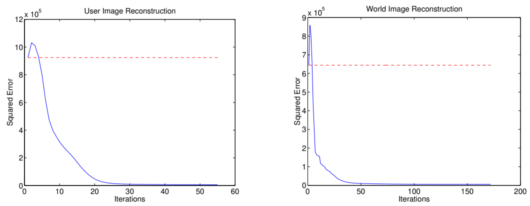


Figure 5: Reconstruction of facial images with PCA and with CoIL estimating permutation matrices.

Figure 6 depicts the squared error of the reconstruction that PCA generates as well as the permutation-based variant (as it converges). Squared error is reduced by approximately 3 orders of magnitude over PCA. In (a), only a single individual's face was used while in (b) the data contains multiple faces of different identities.

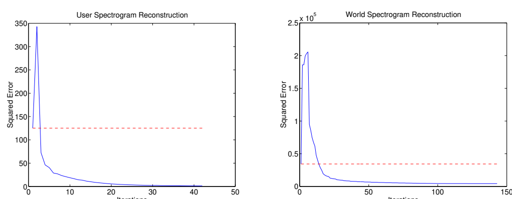


(a) Face Images (b) Multi-Person Images

Figure 6: Reconstruction error for images with 20 eigenvectors. The dashed red line is PCA error while the solid blue line is CoIL error.

7.3 Audio Representation

Similarly, we collected several thousand spectrograms of 200 frequency bands and represented them as a collection of 200 2-tuples of amplitude and frequency. Again, we used the Invisible Hand algorithm due to the large dataset in this experiment. With 20 eigenvectors, the method has significantly better reconstruction error than PCA. Figure 7 depicts squared reconstruction error for audio from a close-talking microphone in (a) as well as an ambient audio microphone.



(a) Close talking mike (b) Wide area mike

Figure 7: Reconstruction error for spectrograms with 20 eigenvectors. The dashed red line is PCA error while the solid blue line is CoIL error.

8 Discussions

We have developed a new framework for specifying affine invariances within various model-learning problems and for estimating the invariances jointly. Computations were constructed such that solutions are unique and emerge from convex programs. The optimizations involve minimizing traces and determinants over covariances matrices on the data which are solvable by quadratic programming and iterative bounds. This framework has many possible application invariances, including clustering, permutation, and so forth. We explored the permutation invariance for dealing with datums that are organized into collections of tuples (i.e. images and pixels) showing that modeling can benefit from the simultaneous estimation of example-specific transformations. Data points are effectively aligned before interacting with a model. They are also aligned as a whole dataset instead of via ad hoc pair-wise criteria [2]. Future work will explore the other types of invariance as well as other possible implementations of the framework (gradient descent, convex programming, etc.) for improved convergence. However, our preliminary experiments are motivating and indicative of the potential of the method.

References

[1] B. Frey and N. Jojic. Estimating mixture models of images and inferring spatial transformations using the EM algorithm. In *CVPR*, 1999.

[2] S. Gold, C.P. Lu, A. Rangarajan, S. Pappu, and E. Mjolsness. New algorithms for 2D and 3D point matching: Pose estimation and correspondence. In *NIPS 7*, 1995.

[3] D. Jakobson and I. Rivin. Extremal metrics on graphs. *Forum Math*, 14(1), 2002.

[4] T. Jebara and T. Jaakkola. Feature selection and dualities in maximum entropy discrimination. In *Uncertainty in Artifical Intelligence 16*, 2000.

[5] M. Jones and T. Poggio. Hierarchical morphable models. In *CVPR*, 1998.

[6] J. Kosowsky and A. Yuille. The invisible hand algorithm: Solving the assignment problem with statistical physics. *Neural Networks*, 7:477–490, 1994.

[7] E. Miller, N. Matsakis, and P. Viola. Learning from one example through shared densities on transforms. In *Computer Vision and Pattern Recognition*, 2000.

[8] J. Platt. Using analytic QP and sparseness to speed training of support vector machines. In *Neural Information Processing Systems 11*, 1999.

[9] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500), 2000.

[10] V. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.