Factor and Component Models

Muchang Bahng

Spring 2025

Contents

1 Principal Component Analysis

PCA finds low dimensional approximations to the data by projecting the data onto linear subspaces. To begin with some motivation, let a linear map $A : \mathbb{R}^D \to \mathbb{R}^D$ be full rank, which maps some set of n data points to the space of features. Then it is injective, and therefore for all data $x \in \mathbb{R}^D$ there exists a feature vector $z \in \mathbb{R}^D$ such that $z = Ax$. Generally, real-world data does not span the full space of D dimensions.[1] In fact, if we further assume that the data lies in a linear subspace, we want to compress it into a lower-dimensional vector such that the covariates in this lower dimensional space are also orthogonal, i.e. uncorrelated. We tackle both problems in 2 steps.

1. To compress this representation, we can take a data point $x \in \mathbb{R}^D$ and *approximate* it as a point $\hat{x} \in L_k$ for some k-dimensional subspace $L_k \subset \mathbb{R}^d$ (say that this is done with some function $P : \mathbb{R}^D \to L_k \subset \mathbb{R}^D$).
2. After this projection, we then want to extract the k features such that they are *orthogonal* (i.e. no correlation). This is done with a simple change of basis, which we denote $T : L_k \to \mathbb{R}^k$, giving us $\hat{z} = T\hat{x} = T(P(x))$. We can invert this map $T^{-1} : \mathbb{R}^k \to L_k$ to go from the orthogonalized compressed version $\hat{z}$ to the approximate full version $\hat{x}$.

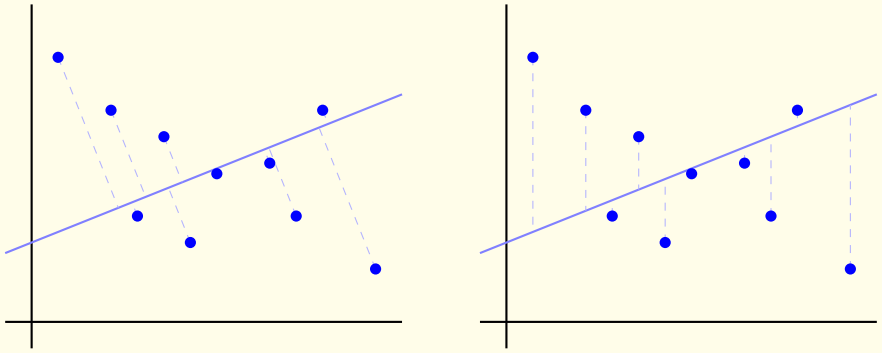
We are done! But which subspace do we choose? Let's formalize what the optimal subspace should be.

Definition 1.1 (Principal Subspace)

Let $x \sim X \in \mathbb{R}^d$, with its normalized $\tilde{x} = x - \mathbb{E}[x]$, and $\mathcal{L}_k$ denote all k-dimensional linear subspaces of $\mathbb{R}^n$. The **kth principal subspace** is defined as

$$\ell_k = \underset{\ell \in \mathcal{L}_k}{\operatorname{argmin}} \mathbb{E}_{\tilde{x}}\big(\min_{y \in \ell} ||\tilde{x} - y||^2 \big) \tag{1}$$

To parse this, let's fix a subspace ℓ. Then, the normalized data $\tilde{x}$ is a random vector and the minimum distance of $\tilde{x}$ onto the subspace ℓ is the inner min term. Taking the expectation of that gives us the expected distance of the data onto the subspace. The principal subspace is the subspace that minimizes this expected distance. The dimension reduced version of x is then $P_k(x) = \mu + \text{proj}_{\ell_k} x$.



(a) PCA minimizes the orthogonal distance to the subspace.

(b) Linear regression minimizes the residual distance to the subspace.

Figure 1: Note that this is in fact different from linear regression as it minimizes the expected *orthogonal distance* to the subspace, rather than the residual distance to the subspace as in linear regression.

We can see that by definition the properties of the principal subspace allows us to construct the best approximation of the points in a lower-dimensional subspace. This seems like a hard optimization problem, but it turns out that the theorem gives a simple solution. Note that we need to do 3 things:

1. Find such a subspace $\mathcal{L}_k \subset \mathbb{R}^D$.

[1] The *manifold hypothesis* that real-world data in high-dimensions actually lies on a lower-dimensional manifold.

2. Find the projection $P_k : \mathbb{R}^D \to \mathcal{L}_k \subset \mathbb{R}^D$. Note that by definition of the principal subspace P_k should be an *orthogonal* projection.
3. Find the bijection $T_k : \mathcal{L}_k \to \mathbb{R}^k$.

It turns out that we can solve all three problems with the singular value decomposition.

Theorem 1.1 (Construction of the kth Principle Subspace)

Given covariates $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}^d$, let $X \in \mathbb{R}^{n \times d}$ be our random matrix representing our data, and its normalized form to $\tilde{X} = X - \mu$. We take the SVD of it.

$$\tilde{X} = U \Sigma V^T \tag{2}$$

where $U \in U(n) \subset \mathbb{R}^{n \times n}, V \in U(d) \subset \mathbb{R}^{d \times d}$ are orthogonal and $\Sigma \in \mathbb{R}^{n \times d}$ is diagonal that represents the singular values of X in decreasing order[a]. The columns of V (rows of V^T) denoted $v_1, \dots, v_d$ are called the **principal axes** and the columns of $U\Sigma$ are called the **principal components**.

1. Then $\ell_k = \text{span}\{v_1, \dots, v_k\} \subset \mathbb{R}^d$, i.e. is the subspace spanned by the columns of V. By shifting it by μ, $\mu + \ell_k$ is the best affine subspace approximation of the x_i's.
2. The projection P is defined

$$\hat{x} = P_k(x) = \mu + \sum_{j=1}^{k} \langle x - \mu, v_j \rangle \, v_j = \sum_{j=1}^{k} \text{proj}_{v_j}(x - \mu) = \mu + \text{proj}_{\ell_k}(x - \mu) \tag{3}$$

   where we can rewrite it as the projection operator since the v_j's are orthonormal.
3. The change of basis T is defined with the mapping $\hat{x} \in \mathcal{L}_k \mapsto \sigma_j v_j \in \mathcal{L}_k$. Note that the v_j's form an orthogonal basis of $\mathcal{L}_k$.

Now let $V_k \in \mathbb{R}^{d \times k}$ represent the first k columns of V (aka first k principal axes), $U_k \in \mathbb{R}^{n \times k}$ represent the first k columns of U, and $\Sigma_k \in \mathbb{R}^{k \times k}$ represent the upper-left $k \times k$ matrix of Σ.[b] The product $U_k \Sigma_k$ represents the matrix containing the first k principal components. The matrix $\tilde{X}_k = U_k \Sigma_k V^T$, which is the low-rank approximation of $\tilde{X}$, is called the **denoised matrix** of $\tilde{X}$.

[a] We can make it decreasing by permuting the rows/columns of the unitary matrices U, V.
[b] Note that V^T, which was originally surjective, is now just injective.

Proof.

For notational convenience let $X = \tilde{X}$. We see that

$$X^T X = V \Sigma^T \Sigma V^T \tag{4}$$

Note that $X \neq V \Sigma^T$ in general. Now let $v_1, \dots, v_d$ be the columns of V. Then

$$X^T X [v_1, \dots, v_d] = X^T X V = V \Sigma^T \Sigma = [\sigma_1^2 v_1, \dots, \sigma_d^2 v_d] \tag{5}$$

Therefore, we can see that the way $X^T X$ acts on V That the v_i's are the eigenvectors of $X^T X$, with σ_i^2 the associated eigenvalues.

Let's take a few moments to appreciate what U and V really represent. In some sense, $U_k \in \mathbb{R}^{n \times k}$ can be considered the dimension-reduced form of $\tilde{X} \in \mathbb{R}^{n \times d}$. To see why consider the following. Let's label the *rows* of U_k as $u^{(1)}, \dots, u^{(n)} \in \mathbb{R}^k$. By transposing the equation of the denoised matrix, we get $\tilde{X}_k^T = V_k \Sigma_k U_k^T$, and so

$$x^{(i)} - \mu = V_k \Sigma_k u^{(i)} \tag{6}$$

for $i = 1, \dots, n$. As an immediate consequence, since T^{-1} maps e_j to $\sigma_j v_j$, we can interpret $U_k \Sigma_k V_k^T$ with the decomposition

$$\overbrace{\underbrace{u^{(i)} \in \mathbb{R}^k \xrightarrow{\Sigma_k} \Sigma_k u^{(i)} \in \mathbb{R}^k \xrightarrow{V_k} \hat{x} \in \mathcal{L}_k}_{T^{-1}} \xleftarrow{P} x \in \mathbb{R}^d}^{P}$$

This is very revealing. To embed the low-rank $u^{(i)}$ representation, it must go through some scaling Σ_k followed by the injective map V_k. Now let's interpret V_k and consider its *columns*, labeled $v_1, \dots, v_k \in \mathbb{R}^d$. These represent the basis vectors that span the subspace $\mathcal{L}_k$, i.e. the upscaled features in the higher-dimensional space. Therefore, V_k represents the injection $e_i \in \mathbb{R}^k \mapsto v_i \in \mathcal{L}_k \subset \mathbb{R}^d$. This means that if we would like to pick a point with some combination of these features, we are really picking a point

$$z = \sum_i z_i v_i \in \mathcal{L}_k \tag{7}$$

Algorithm 1.1 (Fitting)

Given a dataset $X \in \mathbb{R}^{n \times d}$, let us denote the rows as x_i, and say that we are looking for a subspace of dimension k.

1. Compute the mean
$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i \in \mathbb{R}^d \tag{8}$$
2. Standardize the data $\tilde{X} = X - \mu$, i.e. $\tilde{x}_i = x_i - \mu$.
3. Compute the SVD $\tilde{X} = U \Sigma V^T$.
4. Compute the submatrices $V_k \in \mathbb{R}^{k \times k}$ and $\Sigma_k \in \mathbb{R}^{D \times k}$.
5. Define the projection operator $P_k(x) = \mu + \sum_{j=1}^{k} \langle x - \mu, v_j \rangle \, v_j$, the change of basis operator T, and the embedding operator $T^{-1}(z) = \mu + V_k \Sigma_k z$.

A demonstration is done here.

Example 1.1 (Eigenfaces)

In 1991, Turk and Pentland presented an eigenface method of face recognition by taking the low-rank approximation of a dataset of face images.

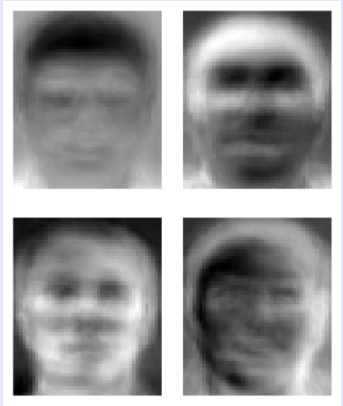


Figure 2: Some eigenfaces from AT&T Labs.

Now a question arises: how do we know that this sample decomposition is a good approximation to the true decomposition? It comes from the fact that the sample covariance $\hat{\Sigma}$ is a good approximation of the true covariance Σ, which we will later prove using concentration of measure.

Theorem 1.2 (Risk)

The risk satisfies

$$R(k) = \mathbb{E}[||x - P_k(x)||^2] = \sum_{j=k+1}^{D} \lambda_j \tag{9}$$

It is essential that you plot the spectrum in decreasing order. This allows you to analyze how well PCA is working. People often use the "elbow" technique to determine where to choose K, and we value

$$\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{d} \lambda_j} \tag{10}$$

accounts for the **variance explained**, which should be high with K low. If you have to go out to dimension $K = 50$ to explain 90% of the variance, then PCA is not working. It may not work because of many reasons, such as there being nonlinear structure within the data.

It turns out that the elements of $\hat{\Sigma}$ are close entry-wise to those of Σ. But if this is true, then does it mean that the eigenvalues of the sample covariance matrix are close to the true eigenvalues of the covariance matrix? It turns out that the answer is no, and we need a proper metric to satisfy this assumption. The metric, as we can guess from linear algebra, is the operator norm, and we will show some results from matrix perturbation theory.

Lemma 1.1 ()

It turns out that

$$||\hat{\Sigma} - \Sigma|| = O_p\left(\frac{1}{\sqrt{n}}\right) \tag{11}$$

where $||\cdot||$ is the operator norm.

Theorem 1.3 (Weyl's Theorem)

If $\hat{\Sigma}$ and Σ are close in the operator norm, then their eigenvalues are close.

$$||\hat{\Sigma} - \Sigma|| = O_p\left(\frac{1}{\sqrt{n}}\right) \implies |\hat{\lambda}_j - \lambda_j| = O_p\left(\frac{1}{\sqrt{n}}\right) \tag{12}$$

This only talks about their eigenvalues, but this does not necessarily imply that the eigenvalues are close. We need an extra condition.

Theorem 1.4 (David-Kahan Theorem)

If $\hat{\Sigma}$ and Σ are close in the operator norm, and if the eigenvectors of Σ are well-conditioned, then the eigenvectors of $\hat{\Sigma}$ are close to the eigenvectors of Σ. More specifically,

$$||\hat{v}_j - v_j|| \leq \frac{2^{3/2}||\hat{\Sigma} - \Sigma||}{\lambda_j - \lambda_{j+1}} \tag{13}$$

1.1 Kernel PCA

Definition 1.2 (Kernel PCA)

Let N_i be the neighborhood around X_i. Then, we want to find a mapping $W : \mathbb{R}^n \to \mathbb{R}^k$ that minimizes

$$\min_W \sum_{i=1}^n \left|\left| X_i - \sum_{j \in N_i} W_{ij} X_j \right|\right|^2 \text{ where } \sum_j W_{ij} = 1 \tag{14}$$

We can constrain the weights in W so that anything that is not in the neighborhoods are 0.

2 Linear Latent Variable Models

Note that in PCA, we have taken some data x in high-dimension D and reduced it to a lower-dimensional orthogonal representation in $\mathbb{R}^k$. In other words, the corresponding z represents x in another space, which we call a **latent space**. A model that represents data from the original space $\mathcal{X}$ to a latent space $\mathcal{Z}$ is called a *latent variable model*. We will extend on this.

Say that we have some covariates $x^{(i)} \sim X$ and we want to find its true distribution p^*. In density estimation so far, what we have done is define a family of distributions $\{p_\theta\}$ and optimize the loss by maximizing the MLE or something else.

$$\min_\theta L(p_\theta, p^*) = \max_\theta \prod_i p_\theta(x^{(i)}) \tag{15}$$

In order to do this we work with explicitly parameterized distribution families (e.g. Gaussian, Gamma, multinomial, etc.), but this is too simple to model complex things in real like (e.g. the distribution of faces). Therefore, we consider *implicitly parameterized* probability distributions by "adding" a latent distribution Z, creating the joint distribution (X, Z). This may look more complicated, but it captures a much richer family of distributions.

Definition 2.1 (Generative Latent Variable Model)

A **latent variable model** is a model of a distribution $p^*(x)$ over a space $\mathcal{X}$ using implicitly parameterized probability distributions p_θ constructed as such:

1. We define a simple random variable Z over $\mathcal{Z}$ with its distribution $p(z)$, called the **prior**.[a]
2. We define a family of functions $\{f_\theta\}$ defined over z and parameterized by θ.
3. We define a way to convert any $f_\theta(z)$ into a distribution $p(x \mid z)$, called the **likelihood** or **generative component**. There are generally two ways to do this:
   (a) Let the random variable $X \mid Z = z$ be an explicitly parameterized distribution, and have $f_\theta(z)$ be the parameters of $X \mid Z = z$. Therefore, we take the output of $f_\theta(z)$ and plug in these values as the parameters of $X \mid Z = z$.[b]
   (b) Have f_θ be a transformation of random variables, i.e. $X = f(Z)$. This may result in a conditional pdf that is not explicitly parameterizable.

This defines the family of joint distributions p_θ over $(\mathcal{Z}, \mathcal{X})$. It is easy to sample $(x, z) \sim p_\theta$: sample $z \sim p$, then compute $f_\theta(z)$, use this to define $p_\theta(x \mid z)$, and finally sample from the likelihood. Therefore, the joint is also of a simple nature.

While we assume simple, explicitly parameterized forms for the prior and the likelihood, we do not assume anything about

1. the **marginal** $p_\theta(x)$. Usually this is an extremely complicated distribution, which is equivalent to
$$p_X(x) = \int_{z \in \mathbb{R}^k} p(x \mid z)\, p_Z(z)\, dz = \mathbb{E}_Z[p(X \mid Z)] \tag{16}$$
from marginalizing but is computationally impossible to integrate.
2. the **posterior** $p_\theta(z \mid x)$ that describes the hidden features given some data point. This is also

known as the **inference component**. By Bayes rule, we have

$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z)\, p(z)}{p_\theta(z \mid x)} \iff p_\theta(z \mid x) \propto p_\theta(x \mid z)\, p(z) \tag{17}$$

which we might be able to sample from using MCMC.

[a] Almost always a uniform or normal distribution suffices. If not, we can constrain it to be factorable (i.e. is the product of its marginal distributions: $p(z) = \prod_i p(z_i)$) so that it is easy to sample from. Occasionally, the stronger assumption of the z_i's being iid is made.

[b] For example, let $f_\theta(z) = (f_1(z), f_2(z))$. Then we define the corresponding distribution $X \mid Z = z \sim \mathcal{N}(f_1(z), e^{f_2(z)})$.

Like we do with everything else in math, we take a look at the simplest example: when the class $\{f_\theta\}$ are linear functions that represent *transformations*[2] $X = f(Z)$ of the random variable Z. This is known as **linear latent variable modeling**.

$$X = \mu + WZ + \epsilon \tag{18}$$

where the noise ϵ is typically Gaussian and diagonal (but not necessarily the same component-wise variances). Finally, we can use techniques like MLE to estimate W, μ, and the parameters of ϵ. The entire reason we want to do this is that we are hoping that we can construct a complex distribution X from a simple distribution Z with $d >> k$, connected by some well-studied function $X = f(Z)$. In the linear case, $W \in \mathbb{R}^{d \times k}$, and the latent variables z give a more compact, parsimonious explanation of dependencies between the components of the observations x.

Definition 2.2 (Factor Analysis)

Factor analysis is a specific case of a linear latent variable model where

$$X = \mu + WZ + \epsilon, \text{ where } z \in \mathcal{N}(0, I), \epsilon \sim \mathcal{N}\big(0, \text{diag}(\sigma_1^2, \dots, \sigma_k^2)\big) \tag{19}$$

It should be clear to us that X should be Gaussian[a] and that $\mathbb{E}[X] = \mu$, with

$$\begin{aligned}
\text{Var}[X] &= \mathbb{E}[(X-\mu)(X-\mu)^T] && (20)\\
&= \mathbb{E}[(WZ+\epsilon)(Z^TW^T + \epsilon^T)] && (21)\\
&= \mathbb{E}[WzZ^TW^T] + \mathbb{E}[\epsilon\epsilon^T] && (22)\\
&= W\mathbb{E}[zz^T]W^T + \mathbb{E}[\epsilon\epsilon^T] && (23)\\
&= WW^T + \text{diag}(\sigma_1^2, \dots, \sigma_d^2) && (24)
\end{aligned}$$

The W, μ, and σ_i's can be estimated using MLE methods.

[a] Since linear transformations of Gaussians are Gaussian

2.1 Probabilistic PCA

We want to take PCA and extend it to be a *generative model*, which allows you to sample data. In regular PCA, we saw that for some $z \in \mathbb{R}^k$ in the latent space, $\hat{x} = \mu + V_k\Sigma_k z$. Therefore, if we just change z from a point to a probability distribution (e.g. Gaussian), we can take a random variable $z \sim \mathcal{N}(0, I)$ from $\mathbb{R}^k$, and then transform it to get a random variable $x = \mu + U_k\Sigma_k z$, which will give a density.

$$x \sim \mathcal{N}\big(\mu, (V_k\Sigma_k)(V_k\Sigma_k)^T\big) = \mathcal{N}\big(\mu, V_k\Sigma_k U_k^T U_k \Sigma_k V_k^T\big) = \mathcal{N}\big(\mu, X_k^T X_k\big) \tag{25}$$

Note that in here, x is a random variable that we are trying to fit to the data X_k. However, $X_k \in \mathbb{R}^{n \times d}$ with $d << n$, and so $X_k^T X_k \in \mathbb{R}^{d \times d}$ is not full rank, and so the distribution is restricted to strictly the

[2] Not have its output parameterize $X \mid Z = z$.

k-dimensional subspace $L_k \subset \mathbb{R}^D$. We want to add a bit of noise beyond the subspace, so we add an extra small Gaussian ϵ around it. In general factor analysis above, we set ϵ to have an arbitrary diagonal Gaussian, but for PPCA we just use an isotropic one $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, giving us

$$x = \mu + U_k \Sigma_k z + \epsilon \implies X \sim \mathcal{N}(\mu, X_k^T X_k + \sigma^2 I) \tag{26}$$

Now rather than treating X_k^T as a data matrix that we use to calculate the principal subspace, we treat it as a parameter matrix $W \in \mathbb{R}^{d \times n}$ that we want to fit [TB99]. Note that PPCA is really a specific instance of factor analysis, and we assume that the latent variable z follows a standard Gaussian $\mathcal{N}(0, 1)$.

Definition 2.3 (Probabilistic PCA)

The **probabilistic PCA** model is a latent factor model with $Z \sim \mathcal{N}(0, I)$ and

$$X = f_\theta(Z) = \mu + (WW^T + \sigma^2 I)^{1/2} Z \tag{27}$$

and $\theta = \{\mu, W, \sigma\}$, which gives

$$X \sim \mathcal{N}(\mu, WW^T + \sigma^2 I) \tag{28}$$

Optimizing this model is actually quite easy.

Theorem 2.1 (MLE of PPCA Model)

Given $x^{(i)} \sim X$ iid, the MLEs for W, μ, σ are

$$\mu_{MLE} = \frac{1}{N} \sum_{i=1}^N x^{(i)} \implies \hat{\mu}_{MLE} = \frac{1}{N} \sum_{i=1}^N x^{(i)} \tag{29}$$

$$\hat{\sigma}^2_{MLE} = \frac{1}{d-k} \sum_{j=k+1}^d \lambda_j \tag{30}$$

$$W_{MLE} = U_q (\Lambda_d - \hat{\sigma}^2_{MLE} I_d)^{1/2} R \tag{31}$$

Proof.

Given $x^{(i)} \sim X$ iid, the MLEs for W, μ, σ have a closed form, and model parameter estimation can be performed iteratively and efficiently. We have

$$\mu_{MLE} = \frac{1}{N} \sum_{i=1}^N x^{(i)} \implies \hat{\mu}_{MLE} = \frac{1}{N} \sum_{i=1}^N x^{(i)} \tag{32}$$

and setting the biased MLE estimator of the variance,

$$\widehat{\text{Var}}_{MLE}(\mu_{MLE}) = S = \frac{1}{N} \sum_{i=1}^N (x^{(i)} - \mu_{MLE})(x^{(i)} - \mu_{MLE})^T \tag{33}$$

we can derive the MLE of W.[a] We can find the MLE estimate of σ first by taking a look at $C = \text{Var}[X] = WW^T + \sigma^2 I$. It is the sum of positive semidefinite patrices that are also symmetric, so by the spectral theorem it is diagonalizable and has full rank d. But WW^T is rank k, so $d-k$ of the eigenvalues of WW^T is 0, indicating that the same $d-k$ smallest eigenvalues of C is σ^2. Therefore, we can take the smallest $d-k$ eigenvalues of our MLE estimator of C, which is S, and average them to get our MLE for σ.

$$\hat{\sigma}^2_{MLE} = \frac{1}{d-k} \sum_{j=k+1}^d \lambda_j \tag{34}$$

We can approximate $WW^T = C - \sigma^2 I \approx S - \hat{\sigma}^2_{MLE} I$, and by further taking the eigendecomposition $C = U\Sigma U^T \implies WW^T = U(\Sigma - \sigma^2 I)U^T$ and cutting off the last $d-k$ smallest eigenvalues and their corresponding eigenvectors, we can get

$$W_{MLE} = U_q(\Lambda_d - \hat{\sigma}^2_{MLE} I_d)^{1/2} R \tag{35}$$

where the R just accounts for any unitary matrix.

[a]Note that W_{MLE} is not unique. Say that W^* is an MLE, then, for any unitary $U \in \mathbb{R}^{k \times k}$, we have $W^* W^{*T} = (W^*U)(W^*U)^T$.

Now as $\sigma \to 0$, the density model defined by PPCA becomes very sharp around these d dimensions spanned by the columns of W. At 0, our MLE of W is simplified and we have

$$X = W_{MLE} z + \mu_{MLE} + \epsilon = U_q \Lambda_q^{1/2} z + \mu_{MLE} \tag{36}$$

which essentially reduces to regular PCA. That is, the conditional expected value of z given X becomes an orthogonal projection of $X - \mu$ onto the subspace spanned by the columns of W. Intuitively, we can see that we are estimating the Gaussian, which corresponds to the mean squared distance from each $x^{(i)}$ to ℓ_k.

2.2 Linear Independent Component Analysis

ICA is a method to separate a multivariate signal into additive, statistically independent components. It does come with a lot of assumptions, and is a specific instance of a linear factor model where $\mu = 0$ and $\epsilon = 0$.

Definition 2.4 (Linear ICA)

In **linear ICA**, we have the simple model.

$$x = Wz \tag{37}$$

In here, $X \in \mathbb{R}^d$ is a mixture vector and $W \in \mathbb{R}^{d \times k}$ is a **mixing matrix**. Both W and z are unknown, and we need to recover them given x. We have 2 strong assumptions.

1. Each component of z is independent (not just uncorrelated). This is an easy enough assumption to intuit.
2. Independent components of z must *not* be Gaussian.[a]

[a]This is needed for us to be able to "unmix" the signals. To see why, just suppose z was Gaussian, and so the vector Rz is also Gaussian for any invertible R. Therefore, we could find an infinite number of solutions of form $x = WR^{-1}Rz$ and have no way to separate them.

Algorithm 2.1 (Fitting)

Now let's see how linear ICA actually estimates W and z. Once W is estimated, the latent components of a given test mixture vector, x^* is computed by $z^* = W^{-1}x^*$. So now all there's left to do is to estimate W, which we want to estimate so that $W^{-1}x$ is far from Gaussian. The reason for this is that given a bunch of independent non-Gaussian h_i's, if we mix them with a matrix that is not $\pm I$, then by CLT, a linear combination of random variables will tend to be Gaussian, and so for an arbitrary W we would expect x to be Gaussian. Therefore, what we want to do is guess some matrix A, and compute

$$Ax = AWh \tag{38}$$

and if we get things right, $A \approx W^{-1}$, and the result of Ax would look pretty non-Gaussian. If it it not the case, then AW will still be some mixing matrix, and so Ax would look Gaussian. So now the question reduces to how do we choose this A? There are multiple ways to measure non-Gaussianity:

1. The absolute or squared kurtosis, which is 0 for Gaussians. This is a differentiable function w.r.t. W, so we can try maximizing it. This is done for the sample kurtosis, of course.
2. Another measure is by maximizing the neg-entropy.

There are further ambiguities with ICA regarding uniqueness of a best representation. For one, we can only estimate the latent components up to a scaling factor since we will still get

$$x = (\alpha W)(\frac{1}{\alpha} z) \text{ for some } \alpha > 0 \tag{39}$$

We can fix this by forcing $\mathbb{E}[z_i^2] = 1$. However, there is still an ambiguity for the sign of hidden components, but this is insignificant in most applications. Second, we can estimating the components up to permutation. We have

$$x = WP^{-1}Pz \tag{40}$$

for some permutation matrix P.

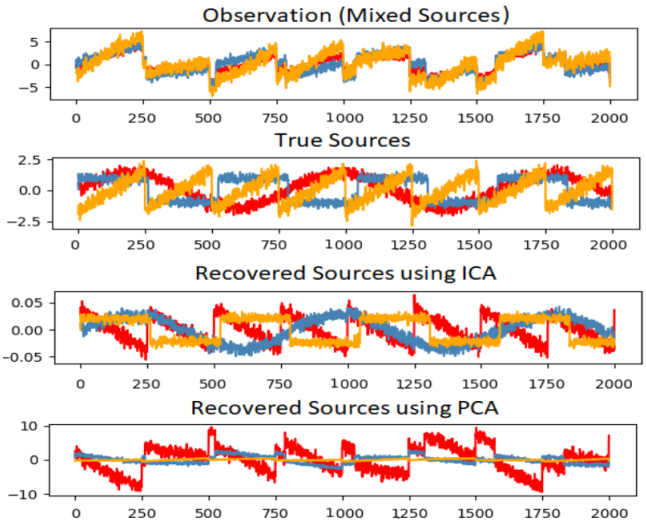


Figure 3: We can perform this on three mixed signals with additive noise, and ICA does very well, though again some recovered signals are scaled or permuted weirdly.

2.3 Slow Feature Analysis

Slow feature analysis also another special case of a linear factor model that uses information from time signals to learn invariant features. It is motivated by a general principle called the **slowness principle**. The idea is that the important characteristics of scenes change very slowly compared to the individual measurements that make up a description of a scene. For example, in computer vision, individual pixels can change very rapidly. If a zebra moves from left to right across the image, an individual pixel wil rapidly change from black to white. By comparison, the feature indicating whether a zebra is in the image will not change at all, and the feature describing the zebra's position will change slowly. Therefore, we want to regularize our model to learn features that change slowly over time.

We can apply the slowness principle to any differentiable model trained with gradient descent. That is, we can add the following term to the loss function:

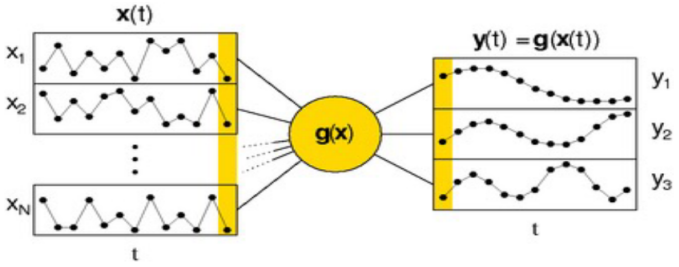
$$\lambda \sum_i d\big(f(x^{(t+1)}), f(x^{(t)})\big) \tag{41}$$

where λ is a hyperparameter determining the strength of the slowness regularization term, t is the time index, f is the feature extractor to be regularized, and d is the distance between $f(x^{(t)})$ and $f(x^{(t+1)})$. A common choice for d is the mean squared difference.

Essentially, given a set of time-varying input signals $x^{(t)}$, SFA learns a nonlinear function f that transforms x into slowly-varying output signals y. Obviously, we can't just take some trivial function like $f = 0$, so we have the following constraints

$$\mathbb{E}_t[f(x^{(t)})_i] = 0 \tag{42}$$

$$\mathbb{E}_t[f(x^{(t)})_i^2] = 1 \tag{43}$$



We can restrict the nonlinear f to some subspace of functions, and this becomes a standard optimization problem where we solve

$$\min_\theta \mathbb{E}_t\left[\left(f(x^{(t+1)})_i - f(x^{(t)})_i\right)^2\right] \tag{44}$$

2.4 Latent Dirichlet Allocation

2.5 Sparse Dictionary Learning

Latent variables can help us represent data in lower dimensions, but another advantage is that we can get *sparse* representations as well. What we want to do in sparse coding is that for each input $x^{(i)}$, we want to find a latent representation $z^{(i)}$ such that it is sparse (i.e. has many 0s) and also we can reconstruct the original input $x^{(i)}$ well. We have basically two things to optimize: the latent representations z and the decoding mechanism, which we can do with a *dictionary matrix* D. Note that we are optimizing for *both* the latent encodings and the decoding mechanism, and so this isn't a generative model.

Definition 2.5 (Sparse Dictionary Encoding Model)

The **sparse dictionary encoding model** is a representation model defined

$$X = g_D(Z) = DZ \tag{45}$$

where $D \in \mathbb{R}^{d \times k}$ is a **dictionary matrix** that decodes the latent $Z \in \mathbb{R}^k$ to $X \in \mathbb{R}^d$. Note that both the $z^{(i)}$'s and D are optimized, so we want to perform the *joint* optimization[a]

$$\min_D \frac{1}{N} \sum_{i=1}^N \min_{z^{(i)}} \underbrace{\frac{1}{2}||x^{(i)} - Dz^{(i)}||_2^2}_{\text{reconstruction error}} + \underbrace{\lambda||z^{(i)}||_1}_{\text{sparsity penalty}} \tag{46}$$

[a] To break this term down, let's just assume that we have a fixed dictionary D. Then, we just need to minimize with respect to each $h^{(t)}$. Now we can add the dictionary parameter back again.

Note that the reconstruction, or decoding, of $x = Dz$ is linear and explicit, but if we want to encode $x \mapsto z$, we need to substitute the x into the term above and minimize it w.r.t. D and z to solve it. Therefore, this encoder is an implicit and *nonlinear* function of x.

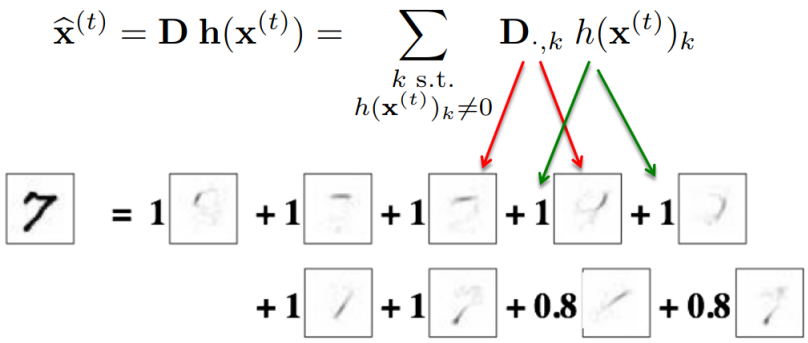


Figure 4: We can reconstruct an image of a seven as a linear combination of a set of images. Note that each of the images of strokes are columns of W and the coefficients make up the sparse vector h.

Let's think about how we can optimize the objective function w.r.t. h, keeping D constant. We can do stochastic gradient descent, which gives us the steps

$$\nabla_{h^{(t)}} \mathcal{L}(x^{(t)}) = D^T(Dh^{(t)} - x^{(t)}) + \lambda \operatorname{sign}(h^{(t)}) \tag{47}$$

but this wouldn't achieve sparsity since it overshoots the 0 all the time. Therefore, we can clip it, or we can use proximal gradient descent/ISTA to take a step, and shrink the parameters according to the L1 norm.

$$h^{(t)} = h^{(t)} - \alpha D^T(Dh^{(t)} - x^{(t)}) \tag{48}$$

$$h^{(t)} = \operatorname{shrink}(h^{(t)}, \alpha\lambda) \tag{49}$$

where $\operatorname{shrink}(a, b) = [\ldots, \operatorname{sign}(a_i)\ \max(|a_i| - b_i, 0), \ldots]$. This is guaranteed to converge if $1/\alpha$ is bigger than the largest eigenvalue of $D^T D$.

Bibliography

[TB99] Michael E. Tipping and Christopher M. Bishop. Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society Series B*, 61(3):611–622, 1999.