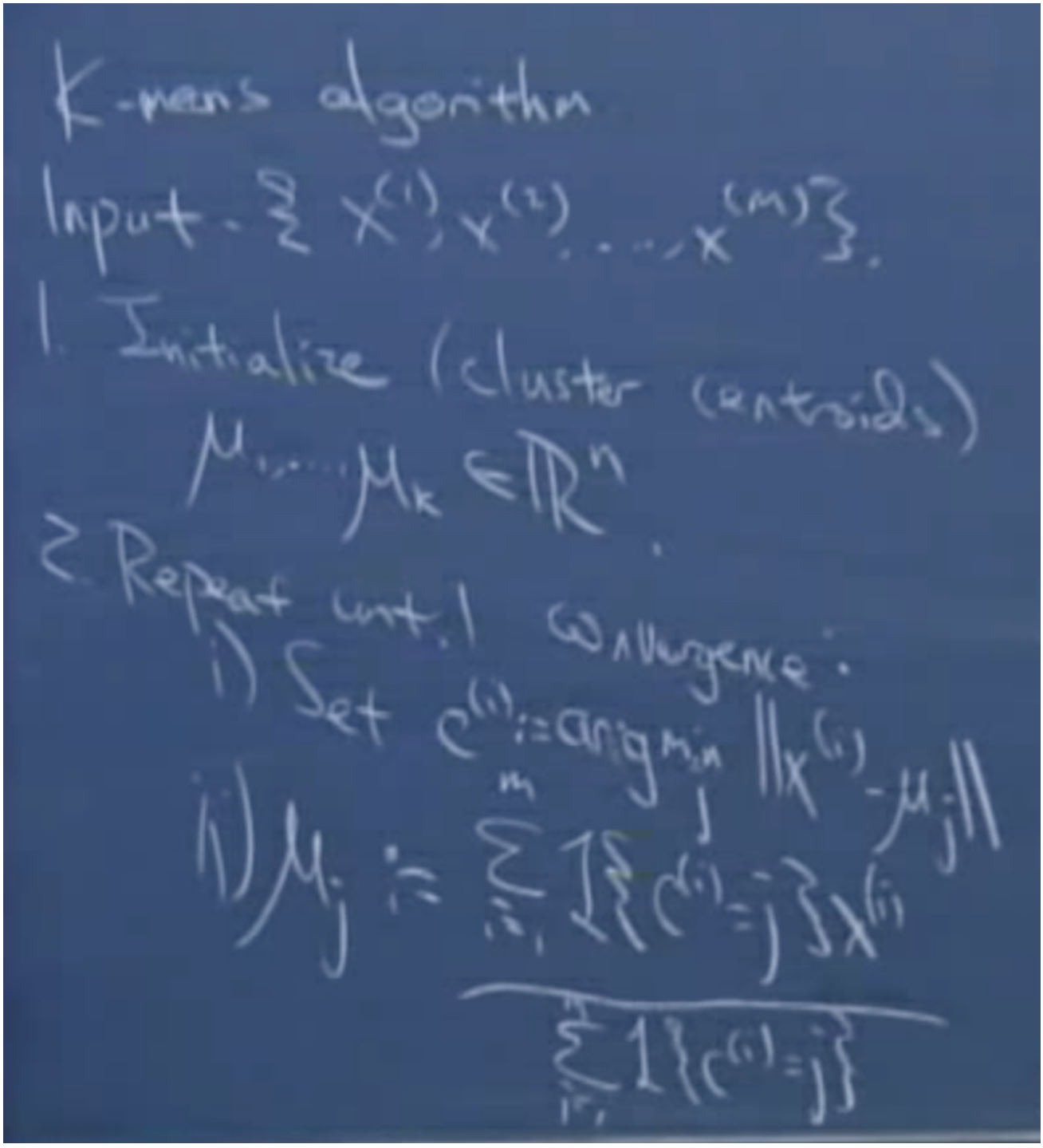
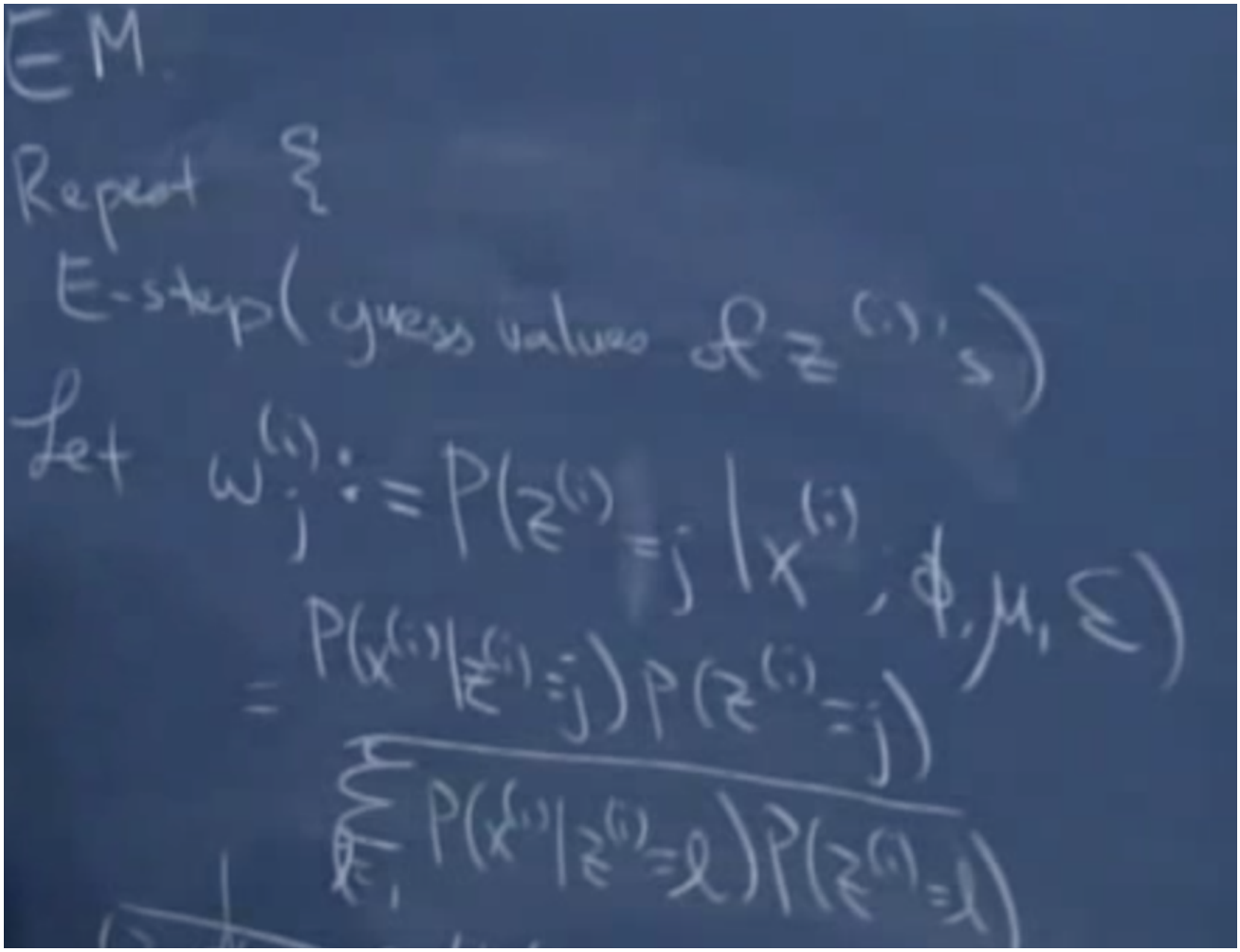
unsupervised learning

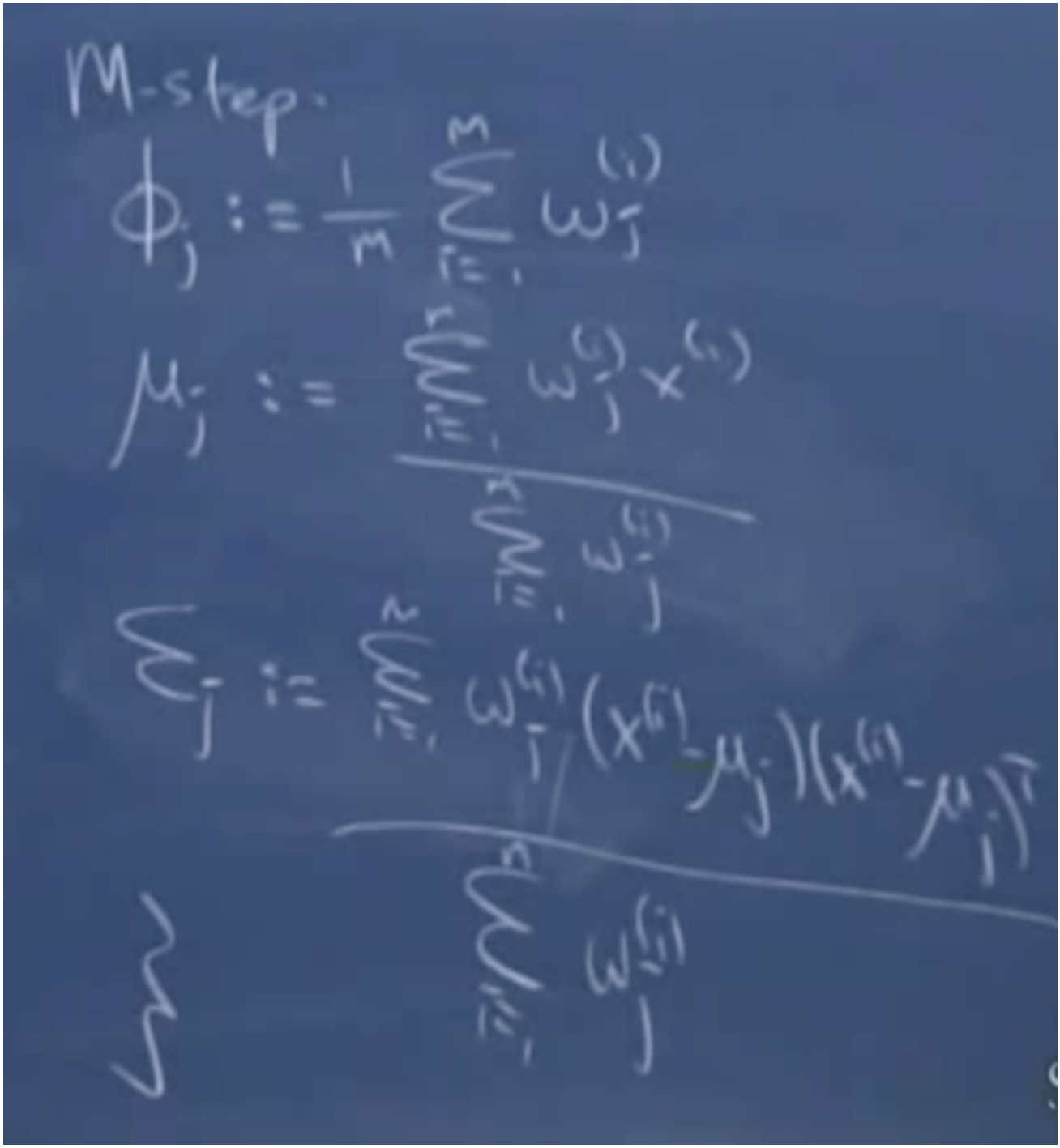
- in supervised learning, we were given a dataset and we were given a label and we were learning a mapping function.
- in unsupervised learning, we are given some data, no labels and the goal is to learn structure in the data.
- An example is news clustering on news.google.com or gmail/inbox bundling
- k-means algorithm for clustering:



- convergence means that after an iteration there is no change (and there can't be any change going forward)
- it can be shown quite easily that k-means is equivalent to performing coordinate descent on $\sum_i^m ||x_i - \mu_j||^2$ where x_i gets assigned to cluster with centroid μ_j.
- k-mean eventually converges, but might hit a local optimum → that function is not convex. so one thing we can do is try multiple random initializations of the centroids.
- how do we choose the number of clusters k?
  - what is often done is we just pick this manually
  - there are automatic algorithms to pick k, but we won't talk about them
- density estimation:
  - goal is to estimate the density of some data.
  - a common application is to detect a highly unlikely new data point (eg. for anomaly detection)
  - often the density we are trying to estimate don't fall into any textbook distribution like gaussian, poisson etc.
  - for some reason, andrew assumes that the observed data is like this:
    - there is latent (hidden, unobserved) variable z which determines which probability distribution to choose from. andrew assumes that each of these two is a gaussian. once we've chosen it, we sample from the chosen gaussian which is $p(x|z)$. so the probability of a observed $x \cap z$ is $p(x \cap z) = p(x|z)p(z)$. Note that this is not just the probability of x, it is $p(x \cap z)$.
    - This is like Gaussian discriminant analysis, except in that case, we know the z's for each thing in the training set. Here we don't (its an unlabeled dataset, hence unsupervised)
    - one other detail → note that z is a multinomial
  - EM algorithm:
    - For now, look at these equations and try to make intuitive sense of them. We will derive a theoretical basis for deriving specific cases of the EM algorithm next.
    - in the E step, we guess the value of z_i (which distribution it came from). we compute $p(z_i = j_i|x_i, \phi, \mu, \Sigma)$ (in general, it is multivariate gaussian) using bayes' rule.



- in the M step, we compute the parameters. here, ϕ, μ, Σ



- Note the similarity of the above with Gaussian discriminant analysis. in GDA, we had the labels (z_i's). Here we are estimating both.
- A broader view of the EM algorithm (not just for a mixture of gaussians):
  - Recall that Jensen's inequality says that for a convex f and any random variable x, $f(E(x)) \leq E(f(x))$. if f is concave, the inequality flips.
  - The EM is really a maximum likelihood estimation: we want to maximize the likelihood of our data so maximize the sum of log likelihoods, $\sum_i log(p(x_i|\theta))$
  - $Q_i(z_i)$ is the probability distribution of a data point i belonging to the different clusters.
  - using jensen's this is:

$$\sum_i \log p(x^{(i)};\theta) = \sum_i \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)};\theta) \quad (1)$$

$$= \sum_i \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})} \quad (2)$$

$$\geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})} \quad (3)$$

EM:

Repeat until convergence {

(E-step) For each i, set

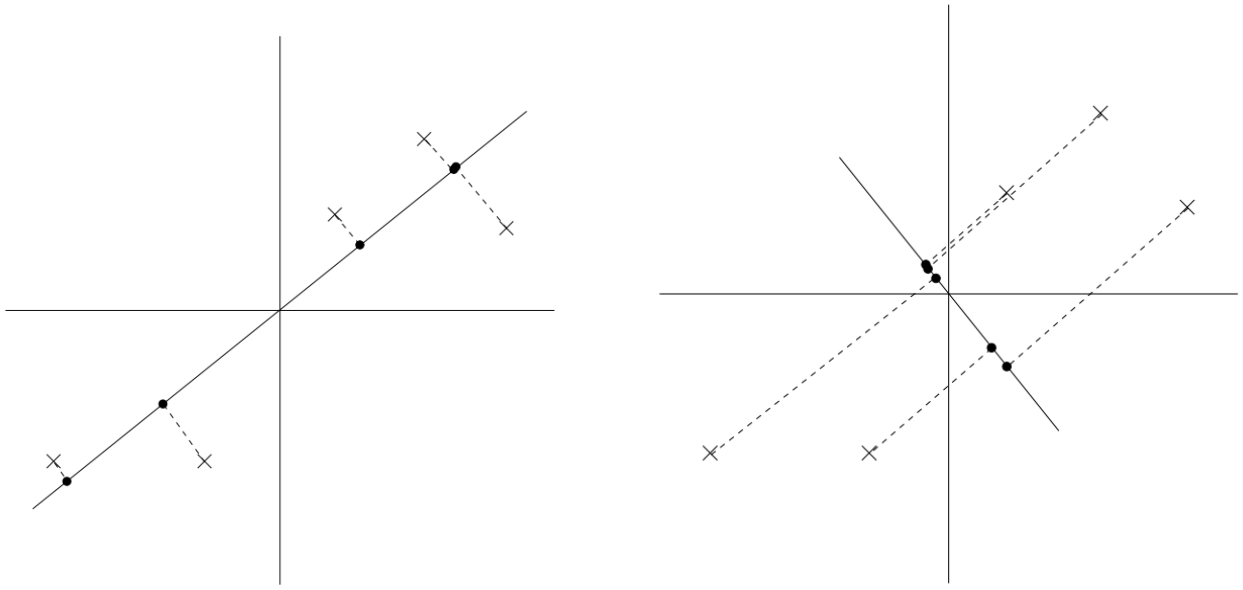
$$Q_i(z^{(i)}) := p(z^{(i)}|x^{(i)};\theta).$$

(M-step) Set

$$\theta := \arg\max_\theta \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})}.$$

- This is the more general form of the EM algorithm, without assuming that each is a gaussian.
- "The reason we don't just find the values of the parameters that maximize $\sum_i log(p(x, z;\theta))$ is that that is computationally intractable, that's why we do the E step, M step way instead" - andrew ng, no proof.
- the above is like doing GDA without knowing labels. Similarly, if we apply EM formulation to naive bayes' without knowing the labels, we get a E step and a M step.
- Andrew talks about something called factor analysis and mentions some math but doesn't go too much into the proof. we will leave out factor analysis here. But basically, factor analysis is a statistical method used to describe variability among observed, correlated variables in terms of a potentially lower number of unobserved variables called factors. For example, it is possible that variations in six observed variables mainly reflect the variations in two unobserved (underlying) variables. Factor analysis searches for such joint variations in response to unobserved latent variables. The observed variables are modelled as linear combinations of the potential factors, plus "error" terms. Factor analysis aims to find independent latent variables.
- Principal Component Analysis (PCA):
  - Given $x_1, ...x_m$, where $x_i \epsilon R^n$, reduce it to k-dimensional data ($k < n$)

- For example, lets say x_i has some components that are functions of each other (like height in meters and hight in feet, rounded to the nearest integer). It need not be perfectly a multiple of each other. it could be some function + some noise. we want to get the underlying "principal component" that determines both those things thereby reducing the dimension of the data.
- Normalize the data (vectors) to $\vec{0}$ mean and unit variance by doing:
  - find $\vec{\mu}$ and replace each vector (data point) by $\vec{x} - \vec{\mu}$.
  - find variance of each feature σ_j^2. replace each x_j^i by $\frac{x_j^i}{\sigma_j}$ so that each feature has the same variance of 1
- with PCA we will find a lower dimensional vector basis to project onto such that once projected, we maximize the variance of the points. for example, think about the figure on the left vs the figure on the right. the one on the left is such that when projected onto, it enforces larger variance between the points than does the line in the one on the right



- If $||u|| = 1$, vector x_i projected on u has length $x_i^T u$.
- Using this, we want to choose u to maximize $\sum_m (x_i^T u)^2$ subject to the constraint $||u|| = 1$
- But, $\sum_m (x_i^T u)^2 = \sum_m (u^T x_i)(x_i^T u) = u^T (\sum_m x_i x_i^T) u$.
- Lets call $\sum_m x_i x_i^T$ as Σ → this is because this is the definition of covariance matrix.
- A principal eigenvector is the eigenvector that corresponds to the largest eigenvalue.
- Now this is a constrained optimization problem. If we set up the Lagrangian and take its derivative and set it to 0 as usual, we see that the solution is such that $\Sigma u = \lambda u$, which means u is an eigenvector of Σ. Since we're looking for the thing that maximizes $\sum_m (x_i^T u)^2$, the eigenvector that maximizes this is the principal eigenvector (easy to verify this yourself)
- More generally, if we want a top k dimensional subspace instead of 1-dimensional, we would choose the top k eigenvectors.
- because Σ is a symmetric matrix, the eigenvectors are independent and form an orthonormal eigenvector basis for the data.
- The reduced k-dimension representation of the data is $u_1^T x_i, u_2^T x_i ... u_k^T x_i$. The new representation is $y_i \, \epsilon \, R^k$
- Because symmetric matrices have a full set of eigenvectors, if $k = n$ you will just get back the original data in a different coordinate system (different basis).

    - Another interpretation of PCA (which Ng does not prove) is that it chooses a basis onto which to project the data so as to minimize the sum of square differences between the projections and the original point. this means it is basically trying to minimize lost information. This is big!
- Applications of PCA:
    - visualization: to visualize a high dimensional dataset with a 2-d or 3-d plot just to understand the data.
    - compression: minimize loss remember?
    - in ML, sometimes we just want to reduce the dimensionality of data and work with a lower dimensional representation. PCA is one way to do what is called dimensionality reduction. This speeds up training and inference with very less loss in accuracy. Reducing number of features also helps reduce overfitting, so PCA is helpful here too.
    - anomaly detection: maybe you have a dataset which can be PCA'ed and the loss is less, but if you detect a new data with high loss, then that might be worth flagging for review.
    - Matching/distance calculations: lets say we want to do face recognition. during training, we can take a bunch of face images and find an eigenvector basis for those images. then given the image of two faces, if we want to tell if they are the same person, then we project them onto the eigenvector subspace and find the distance between them in that subspace instead of finding the distance in the original space. This is the equivalent of leaving out things that don't matter (like backgrounds) that presumably aren't captured in the eigenvector representation so the distance in the eigenvector space captures similarity after "subtracting" things like background. These eigenvector basis for the faces are called eigenfaces.
- In PCA, Because Σ can be a super high dimensional matrix (million x million), computing eigenvectors isn't always feasible. However, there are efficient ways to compute the SVD. So doing PCA (computing eigenvectors of Σ) is equivalent to computing SVD of X where X is the matrix that represents the training data. Since there are efficient ways to compute SVD, we can get around computing a crazy high dimension matrix's eigenvectors directly.
- Andrew talks about a problem of using PCA with text features (super high dimensional features where each x_i^j is 0 or 1 depending on if a document i has word j). when you do similarity measures using the cosine of the angle between documents, if similar meaning but different words are present, then don't contribute to the notion of similarity in the dot product while finding cosine. Andrew talks about using PCA to get around this in what is called latent semantic indexing (LSI) but doesn't go into any details.
- Independent Component Analysis (ICA):
    - This is used to find "independent" components that make up the given data. Andrew cites the cocktail party problem.
    - Lets say we have speakers and microphones. Each speaker outputs an independent component but we only have microphone recording data for multiple microphones.
      Assuming $m_{ji} = \sum_k A_k s_{ki}$, that is, microphone j's output at time i can be modeled by some sum of linear combination of each of the speaker's outputs. we want to find the inverse of A so that we can recover the speaker output from the microphone output.
    - Andrew mentions some cases under which we can't find a unique solution (a few exist) and walks thru some simple sample math for ICA which we won't go into here. The math assumes some simplistic probability models on what is output. More interesting applications have a bit more complicated math but the problem formulation looks similar → we have a bunch of observed data from different points (microphones), and produced from a set of independent components (speakers). goal is to take output of microphones and recover output of speakers.