Matrix factorization for representation learning

Reminders/Comments

- Speed of learners:
  - Sources of slowness: for-loops
  - In naive Bayes, for example, mostly have to loop through samples; however, using vector addition within might speed things up
- A more explicit Mini-project specification added to schedule
  - Gives better feedback about marking
  - Note: initial draft (due November 28) should be an almost complete final draft—only thing that can be missing is some results
- I will provide you will practice questions for the final

Neural networks summary

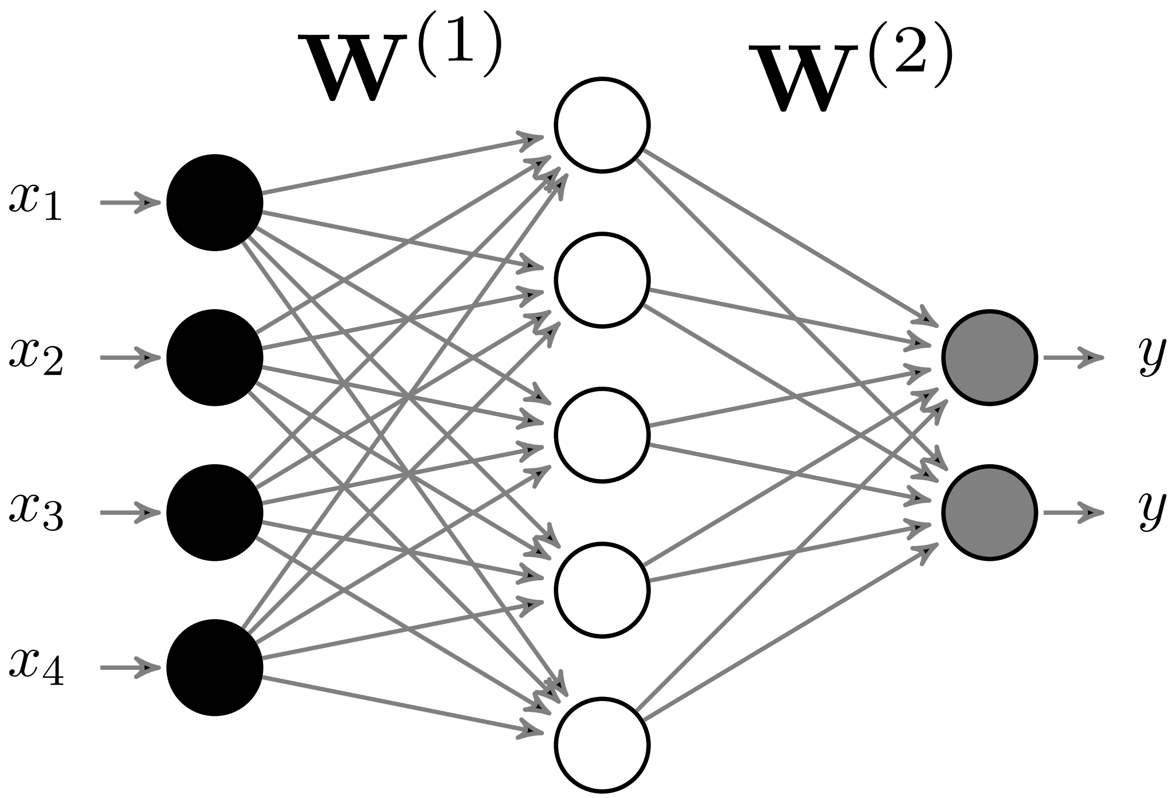
- Discussed basics, including
- Basic architectures (fully connected layers with activations like sigmoid, tanh, and relu)
- How to choose the output loss
  - i.e., still using the GLM formulation
- Learning strategy: gradient descent (called back-propagation)
- Basic regularization strategies
- After reading week, will discuss more advanced topics (for fun)

How else can we learn the representation?

- Discussed how learning can be done in simple ways even for "fixed representations"
  - e.g., learn the centres for radial basis function networks
  - e.g., learn the bandwidths for Gaussian kernel
- Discussed less constrained representation learning setting with neural networks
  - though still quite constrained in our architecture, not just learning any representation
- In general, this problem has been tackled for a long time in the field of unsupervised learning
  - where the goal is to analyze the underlying structure in the data

Representation learning

Neural network



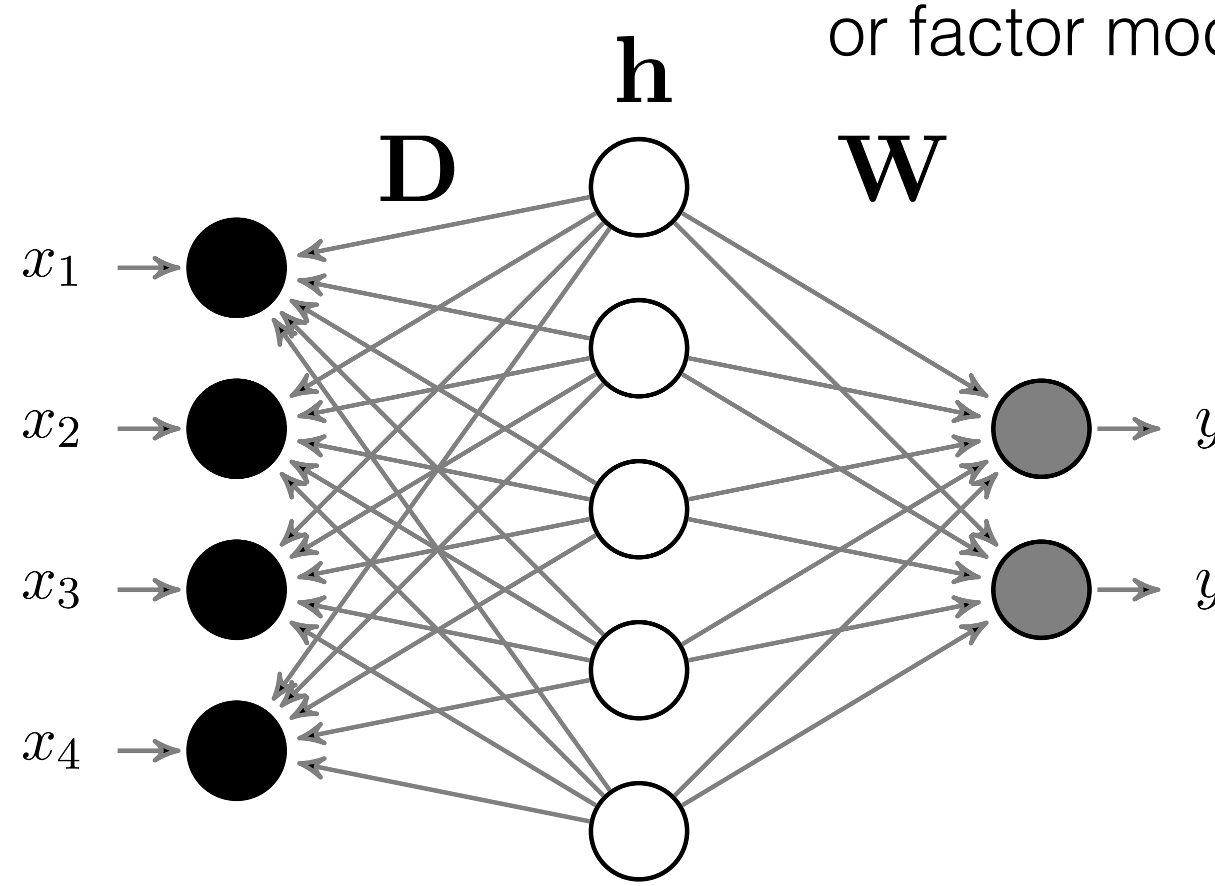
$$\mathbf{W}^{(1)} \in \mathbb{R}^{k\times d}, \mathbf{W}^{(2)} \in \mathbb{R}^{m\times k}$$

$$d = 4, k = 5, m = 2$$

$$\hat{\mathbf{y}} = f_2(\mathbf{W}^{(2)} f_1(\mathbf{W}^{(1)}\mathbf{x}))$$

Dictionary Learning models

or factor models



$$\mathbf{D} \in \mathbb{R}^{k\times d}, \mathbf{W} \in \mathbb{R}^{k\times m}$$

$$d = 4, k = 5, m = 2$$

$$\hat{\mathbf{y}} = f_2(\mathbf{h}\mathbf{W})$$

$$\mathbf{h} = \arg\min_{\mathbf{h}\in\mathbb{R}^{1\times k}} L_x(\mathbf{h}\mathbf{D}, \mathbf{x})$$

Using factorizations

- Many unsupervised learning and semi-supervised learning problems can be formulated as factorizations
    - PCA, kernel PCA, sparse coding, clustering, etc.
- Also provides an way to embed more complex items into a shared space using co-occurence
    - e.g., matrix completion for Netflix challenge
    - e.g., word2vec

Intuition (factor analysis)

- Imagine you have test scores from 10 subjects (topics), for 1000 students

- As a psychologist, you hypothesize there are two kinds of intelligence: verbal and mathematical

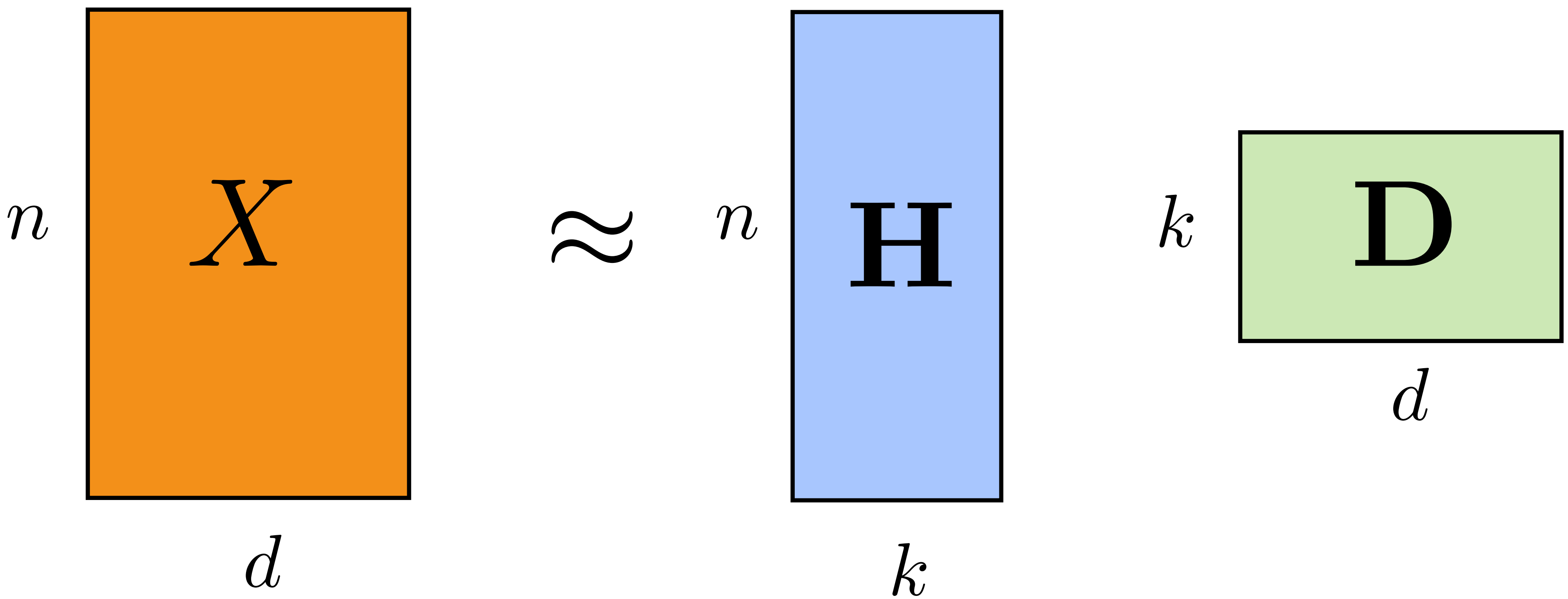
- You cannot observe these factors (hidden variables)

- Instead, you would like to see if these two factor explain the data, where x is the vector of test scores of a student

- Want to find: $x = d1\ h1 + d2\ h2$, where d1 and d2 are vectors h1 = verbal intelligence and h2 = mathematical intelligence

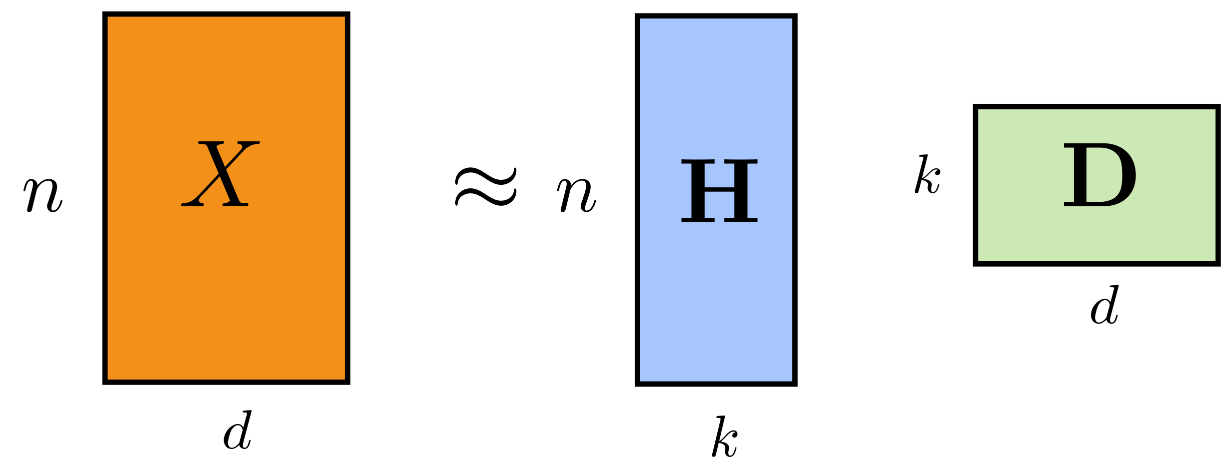
- Having features h1 and h2 would give a compact, intuitive model

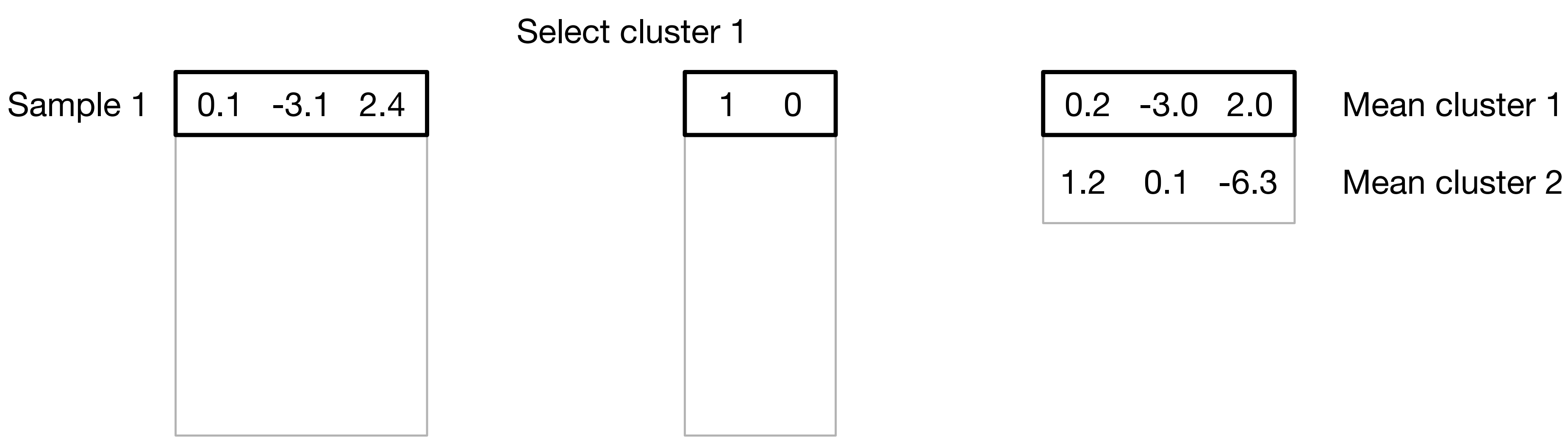
Matrix factorization



If k < d, then we obtain dimensionality reduction (PCA)

Example: K-means





$$\left\|\mathbf{x} - \sum_{i=1}^{2} 1\left(\mathbf{x} \text{ in cluster } i\right) \mathbf{d}_i\right\|_2^2 = \|\mathbf{x} - \mathbf{h}\mathbf{D}\|_2^2$$

where $\mathbf{h} = [1 \quad 0]$ or $\mathbf{h} = [0 \quad 1]$ and $\mathbf{D} = [\mathbf{d}_1 \ ; \ \mathbf{d}_2]$.

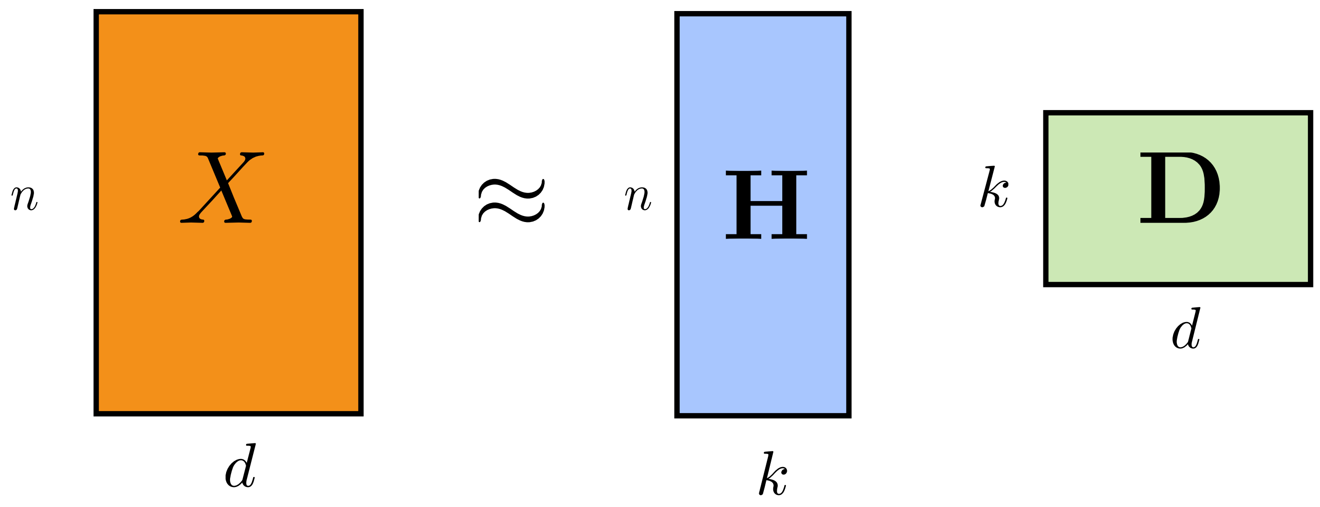
Dimensionality reduction

- If set inner dimension k < d, obtain dimensionality reduction

- Recall that the product of two matrices H and D has rank at most the minimum rank of H and D

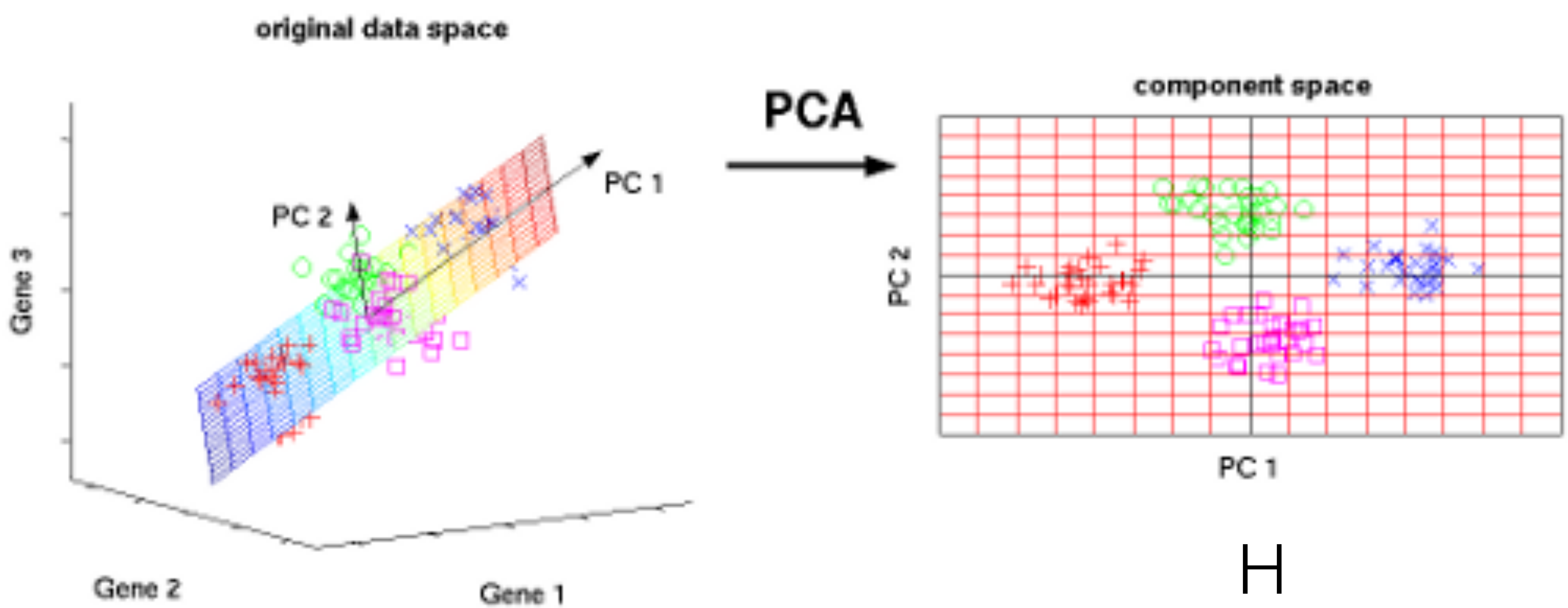
$$\text{rank}(\mathbf{HD}) \leq \min(\text{rank}(\mathbf{H}), \text{rank}(\mathbf{D}))$$

- Even if d = 1000, if we set k = 2, then we get a reconstruction of X that is only two-dimensional
  - we could even visualize the data! How?



Principal components analysis

- New representation is k left singular vectors that correspond to k largest singular values
  - i.e., for each sample x, the corresponding k-dimensional h is the rep
- Not the same as selecting k features, but rather projecting features into lower-dimensional space



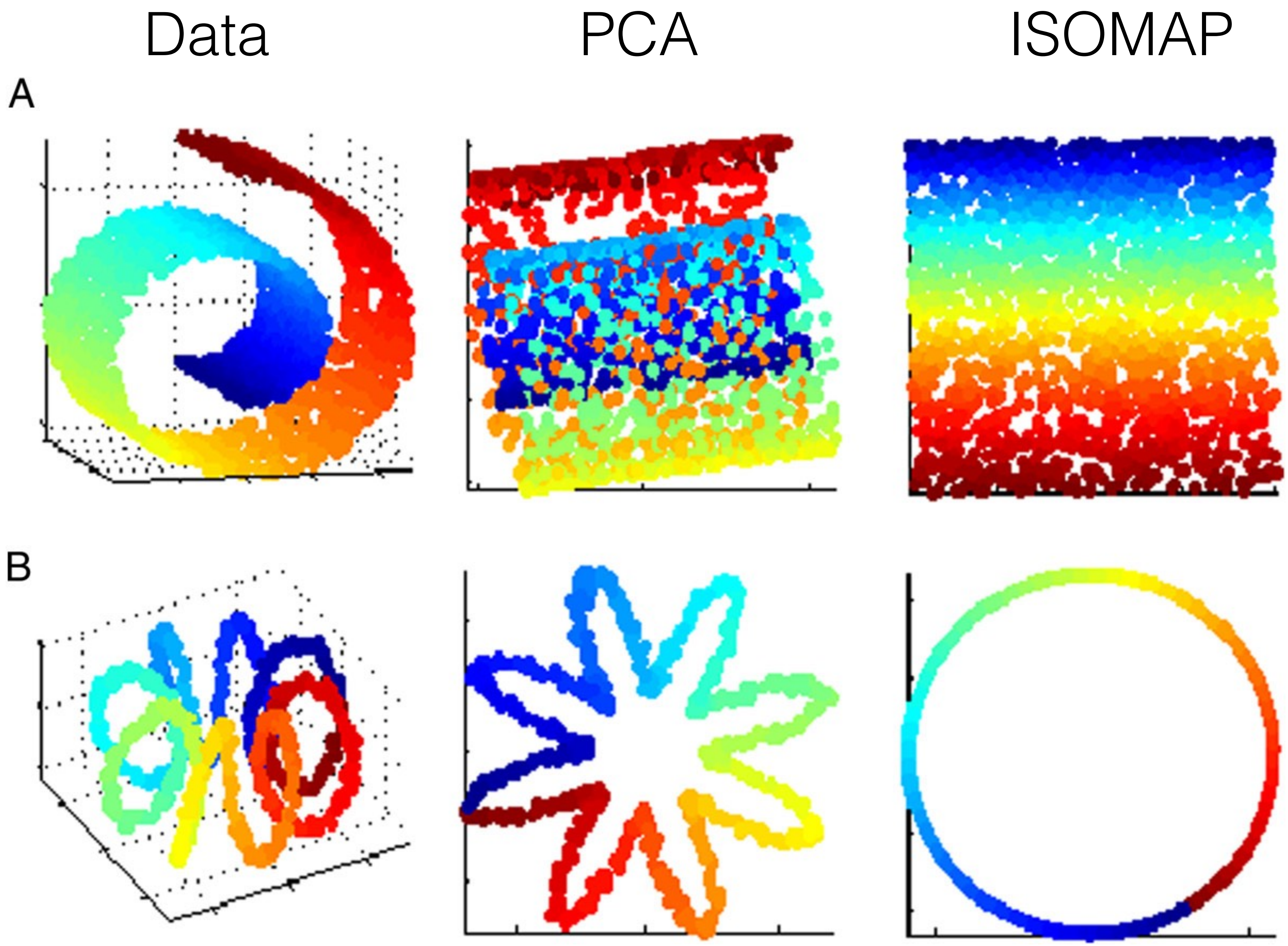
Do these make useful features?

- Before we were doing (huge) nonlinear expansions
- PCA takes input features and reduces the dimension
- This constrains the model, cannot be more powerful
- Why could this be helpful?
  - Constraining the model is a form of regularization: could promote generalization
  - Sometimes have way too many features (e.g., someone overdid their nonlinear expansion, redudant features), want to extract key dimensions and remove redundancy and noise
  - Can be helpful for simply analyzing the data, to choose better models

What if the data does not lie on a plane?

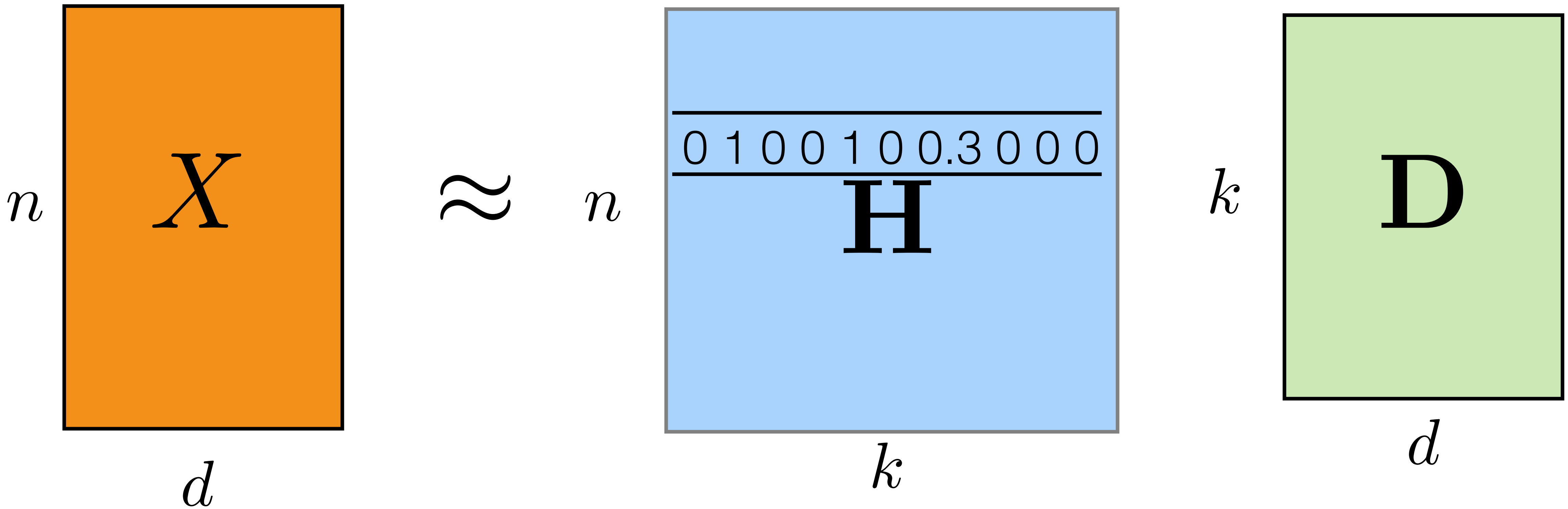
- Can do non-linear dimensionality reduction
- Interestingly enough, many non-linear dimensionality reduction techniques correspond to PCA, but first by taking a nonlinear transformation of the data with a (specialized) kernel
  - Isomap, Laplacian eigenmaps, LLE, etc.
- Can therefore extract a lower-dimensional representation on a curved manifold, can better approximate input data in a low-dimensional space
  - which would be hard to capture on a linear surface

Isomap vs PCA



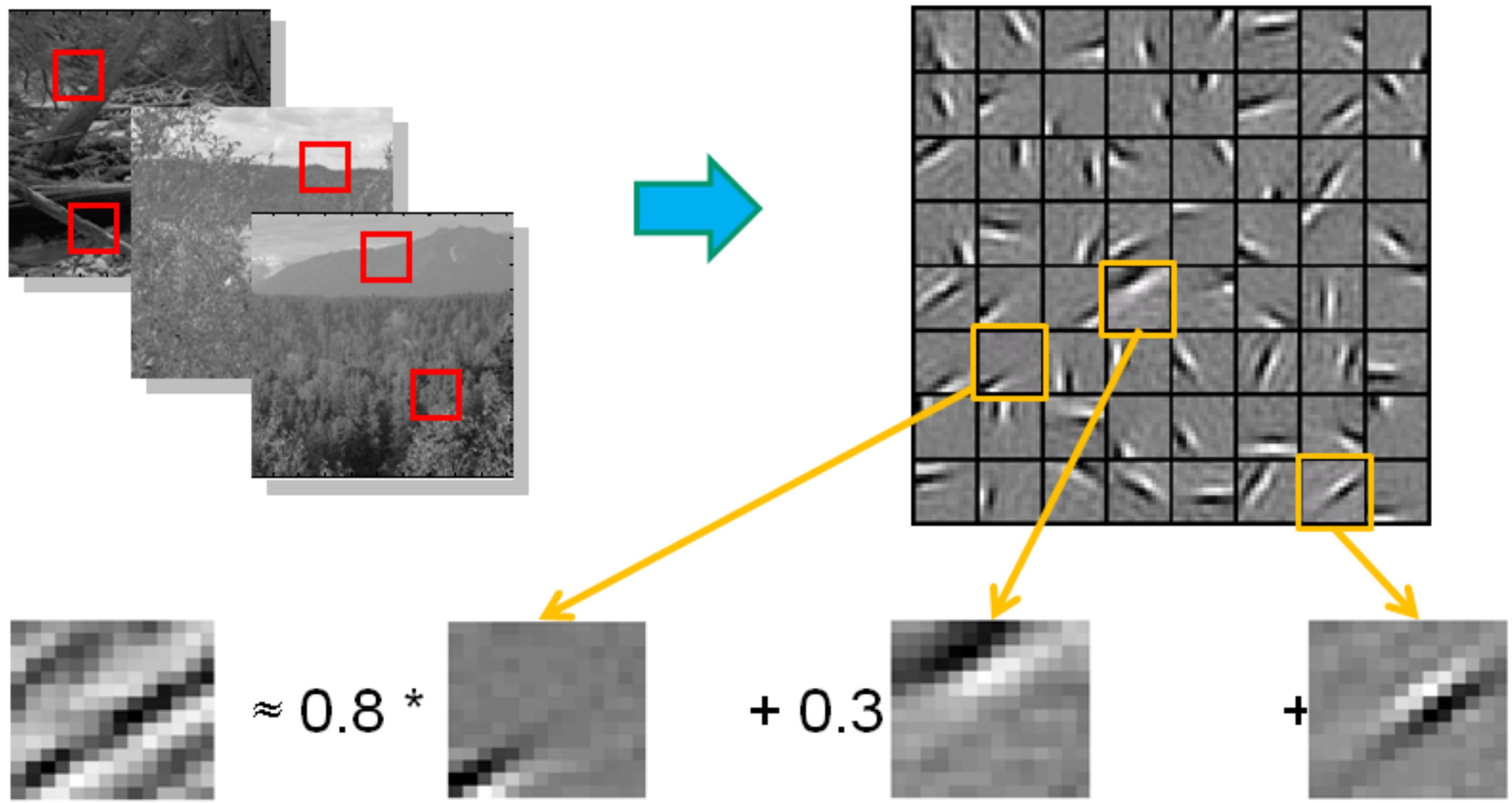
*Note: you don't need to know Isomap, just using it as an example

Sparse coding



- For sparse representation, usually k > d
- Many entries in new representation are zero

Sparse coding illustration



$[a_1, \ldots, a_{64}] = [0, 0, \ldots, 0, \mathbf{0.8}, 0, \ldots, 0, \mathbf{0.3}, 0, \ldots, 0, \mathbf{0.5}, 0]$
(feature representation)

Slide credit: Andrew Ng

Compact & easily interpretable

Whiteboard

- We'll look at more examples of matrix factorizations later
- Let's look at now how to solve for these representations
    - for some settings we will have a closed-form solution (e.g., PCA)
    - for most setting, we will again have to derive an iterative update
- Finish up example with auto-encoder

l1 regularizer for sparse coding

- Why does the l1 regularizer give sparse representations?
  - behaves like the l0 regularizer
- What about the l2 regularizer?
  - the l2 regularizer prefers to more uniformly squash values
  - in fact, picking an l2 regularizer on both H and D ends up corresponding to PCA (subspace representations) —> the interaction of having an l2 on both seems to prefer to zero out entire rows of H and columns of D (relaxed rank PCA)

l1 regularizer and l0 regularizer

$$\ell_0(\mathbf{w}) = \sum_{i=1}^{d} 1(w_i \neq 0) = \#\text{ non-zero entries}$$

$$\ell_1(\mathbf{w}) = \sum_{i=1}^{d} |w_i|$$

- l1 regularizer in practice behaves similarly to l0 regularizer
- Before we used it for feature selection
  - regularized weights w in Xw = y
- Here we are using it on a matrix, so again we are doing feature selection, but separately for each sample

$$\|\mathbf{H}\|_{1,1} = \sum_{i=1}^{k} \sum_{j=1}^{t} |H_{ij}|$$

For other settings

- If there is no closed form solution, we will do as before: compute the gradient and do gradient descent

- Step 1: Compute gradient with respect to H, for fixed D, update H = H - alpha grad_H

- Step 2: Compute gradient with respect to D, for fixed H, update D = D - alpha grad_D

- Natural question: with neural networks, we updated both W1 and W2 simultaneously; why do we alternate between the two variables here?

Alternating methods

- Alternating steepest descent: step in direction of gradient in alternating fashion
  - seems to have nice time, convergence trade-offs
- Alternating minimization: solve for one variable, with the other fixed, in alternating fashion
  - each step corresponds to a batch gradient descent solution with one of the variables fixed
  - more traditional approach with well-known convergence properties
- Which one you use likely depends on your setting; alternating steepest descent is likely a better way in general, if there are computation time restrictions
- Note: this is related to EM, as we will see later (viterbi EM)

What are the distributional assumptions?

- If try to factorize X into HD, making an assumption that p(x l mu = hD) is Gaussian, with some fixed covariance
  - weighted l2-loss gives a different covariance for each entry
- What if the data is binary (not Gaussian) or Poisson distributed? (or some other distribution)
  - again, we can use generalized linear models to generalize the distribution p(x l hD) to exponential families
  - See e.g., paper on exponential family PCA: “A generalization of principal component analysis to the exponential family”, Collins et al., 2002