COMPUTER SCIENCE
INDIANA UNIVERSITY
School of Informatics and Computing
Bloomington

Matrix factorization for representation learning

Reminders/Comments

- Project discussion on canvas
- Assignment marking and discussion with AIs

Thought question

- When training in Naive Bayes Classifier, what are strategies to deal with missing data? We could choose to discard the data with missing features, but might end up discard all the data with nothing left. Inspired by assignment 2, question 4(B), is there a way that we may add some kind of regularizer to solve the missing data issue?
  - At training time, the distribution over each feature is learned separately
  - At prediction time, the main importance is relative probability values for the classes; by ignoring missing value, its like we are marginalizing over that value
  - Depends on why the data is missing: common assumption is that the data is missing at random (i.e., not biased)

Handling missing features

- Naive Bayes assumption makes this simpler
- For regression models (e.g., logistic regression), more difficult. Some options are:
  - replacing the missing values with the mean value for the attribute
  - replacing the missing value with the weighted mean of k nearest neighbors (most similar k other instances)
  - many more suggestions…
- Unsupervised learning (with matrix factorization) will be another option we will discuss
  - e.g. matrix completion

Thought questions

- I have one general problem, why we don't apply our link function f() on all elements of y. And then run linear regression? If we have a vector X and a goal y that we want to learn w such that exp(X^T w) becomes y. Why we don't learn w from a linear regression of X^Tw becoming ln(y)!?

- General related questions: how do we know what transfer to pick and the loss for GLMS? And why do we pick exponential family distributions? What about other distributions?

- General answer:
  - exponential family distributions are great because the negative log-likelihood under iid data is a convex function
  - the specification of p(y | x) as an exponential family distribution tells us what the transfer has to be to get a convex function

Neural networks summary

- Scalar output, two-layer neural network

$$\frac{\partial L(\hat{y}, y)}{\partial \mathbf{W}_{ij}^{(l)}} \text{ where } l = 1, 2$$

- Vector output, two-layer neural network

$$\frac{\partial L(\hat{\mathbf{y}}, \mathbf{y})}{\partial \mathbf{W}_{ij}^{(l)}} \text{ where } l = 1, 2$$

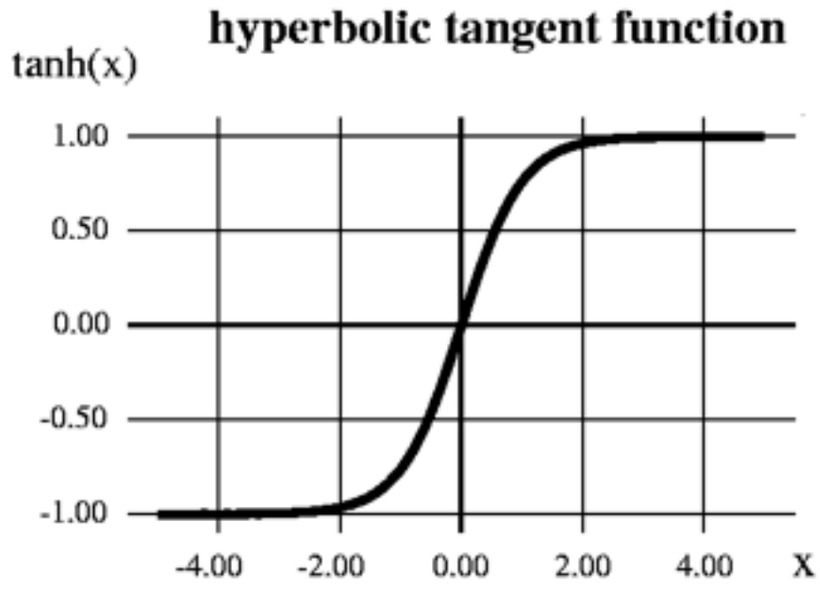
- Vector output, three-layer neural network

$$\frac{\partial L(\hat{\mathbf{y}}, \mathbf{y})}{\partial \mathbf{W}_{ij}^{(l)}} \text{ where } l = 1, 2, 3$$

- Notes help convert to matrix notation, but unnecessary for understanding; start these exercises with partial derivatives and later (if you want) convert to matrix notation

Tanh and rectified linear

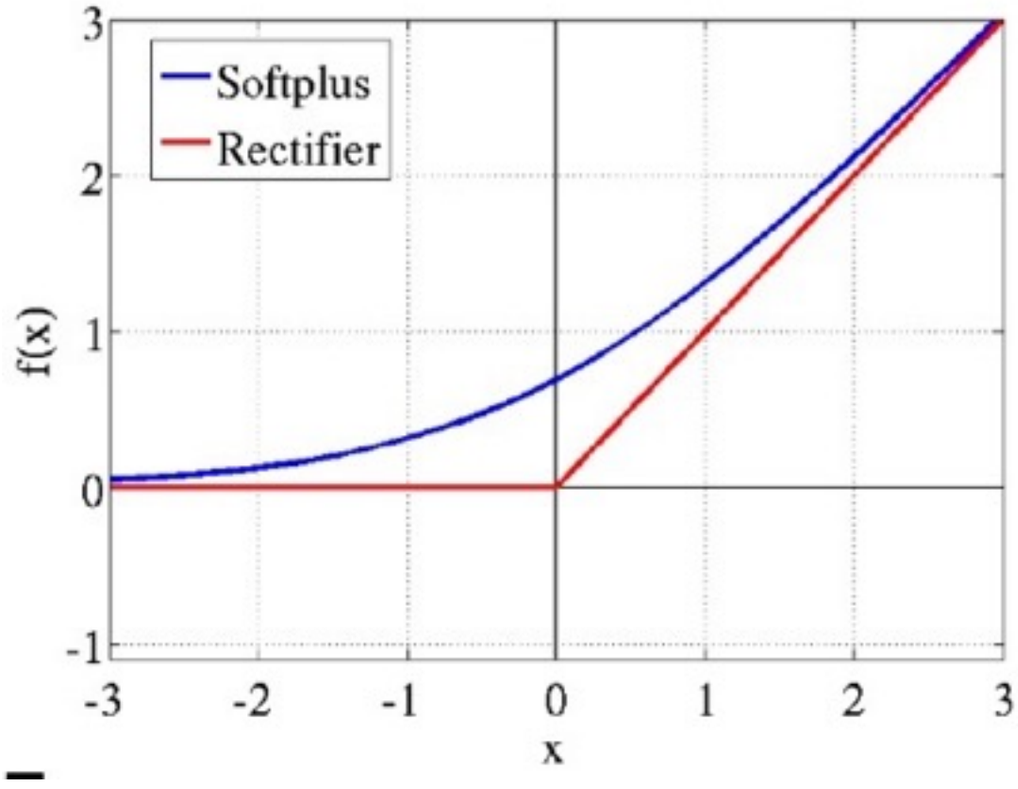
- Two more popular transfers are tanh and rectified linear
- Tanh is balanced around 0, which seems to help learning
  - usually preferred to sigmoid
- Rectified linear



- Binary threshold function (perceptron): less used, some notes for this approach: http://www.cs.indiana.edu/~predrag/classes/2015springb555/9.pdf

Rectified linear

- Rectified(x) = max(0, x)
  - Non-differentiable point at 0
  - Commonly gradient is 0 for x <= 0, else 1
- Softplus(x) = ln(1+e^{x})
- Recall our variable is sum w_i x_i
- Common strategy: still use sigmoid (or tanh) with cross-entropy in the last output layer, and use rectified linear units in the interior

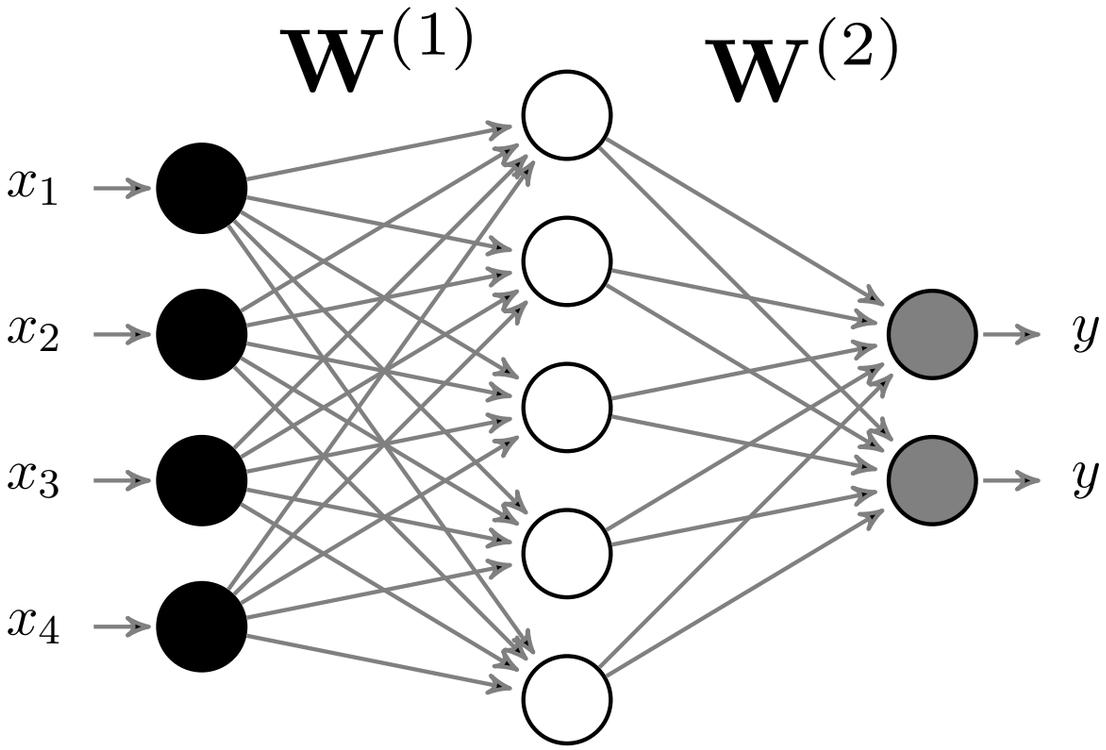


Selecting loss functions

- If we pick a sigmoid transfer for the last layer, what is the loss?
  - cross-entropy (logistic regression loss)
- If we pick a tanh transfer for the last layer, what is the loss?
  - entropic loss (similar to cross-entropy)
  - this is the matching loss for tanh; see http://papers.nips.cc/paper/1610-linear-hinge-loss-and-average-margin.pdf
- If we pick rectified linear for the last layer, what is the loss?
  - no longer has a matching loss
  - usually pick least-squares loss, or do not use it as last layer

Representation learning

Neural network

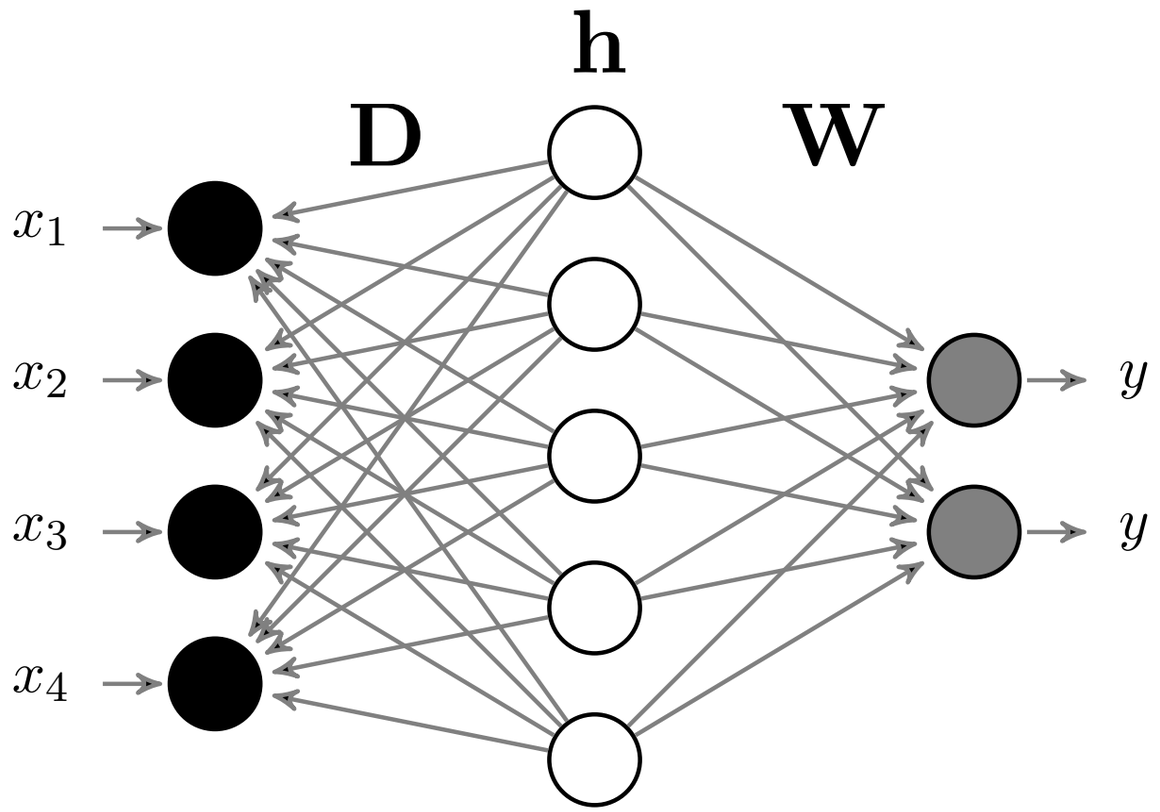


$$\mathbf{W}^{(1)} \in \mathbb{R}^{k \times d}, \mathbf{W}^{(2)} \in \mathbb{R}^{m \times k}$$

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

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

Regularized factor model



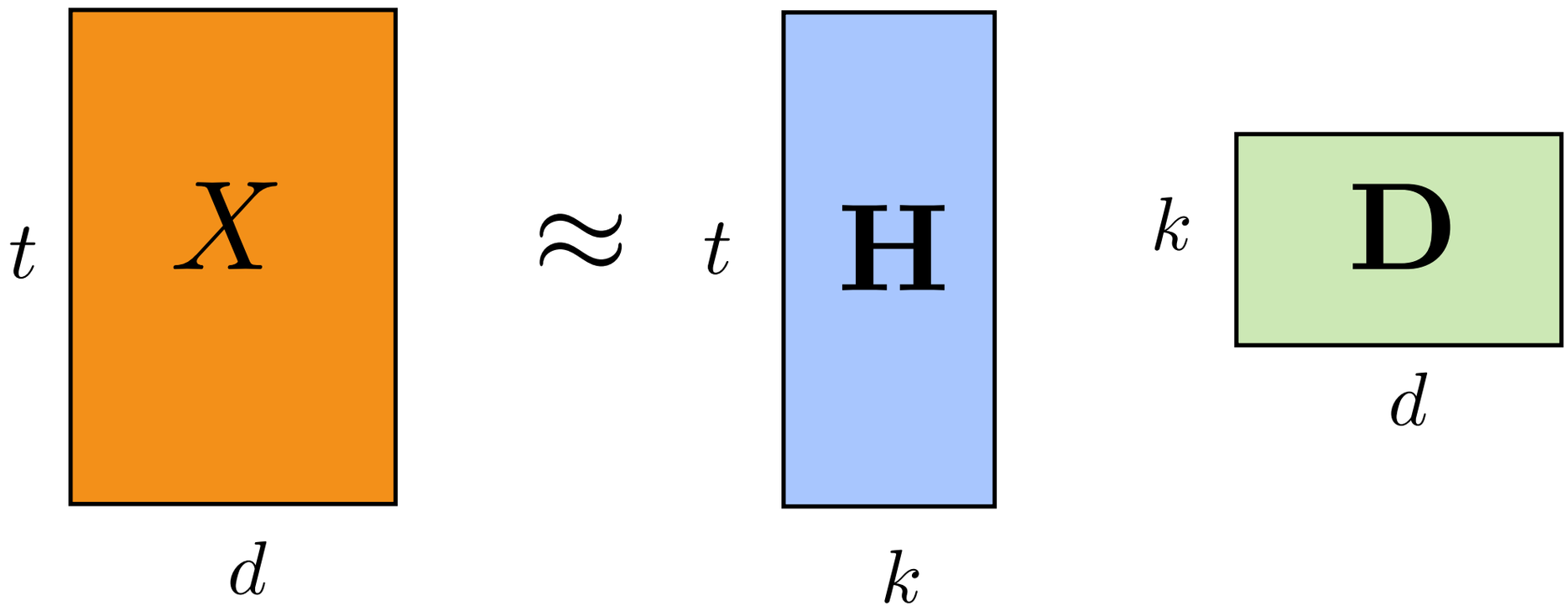
$$\mathbf{D} \in \mathbb{R}^{k \times d}, \mathbf{W} \in \mathbb{R}^{k \times m}$$

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

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

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

Unsupervised RFMs

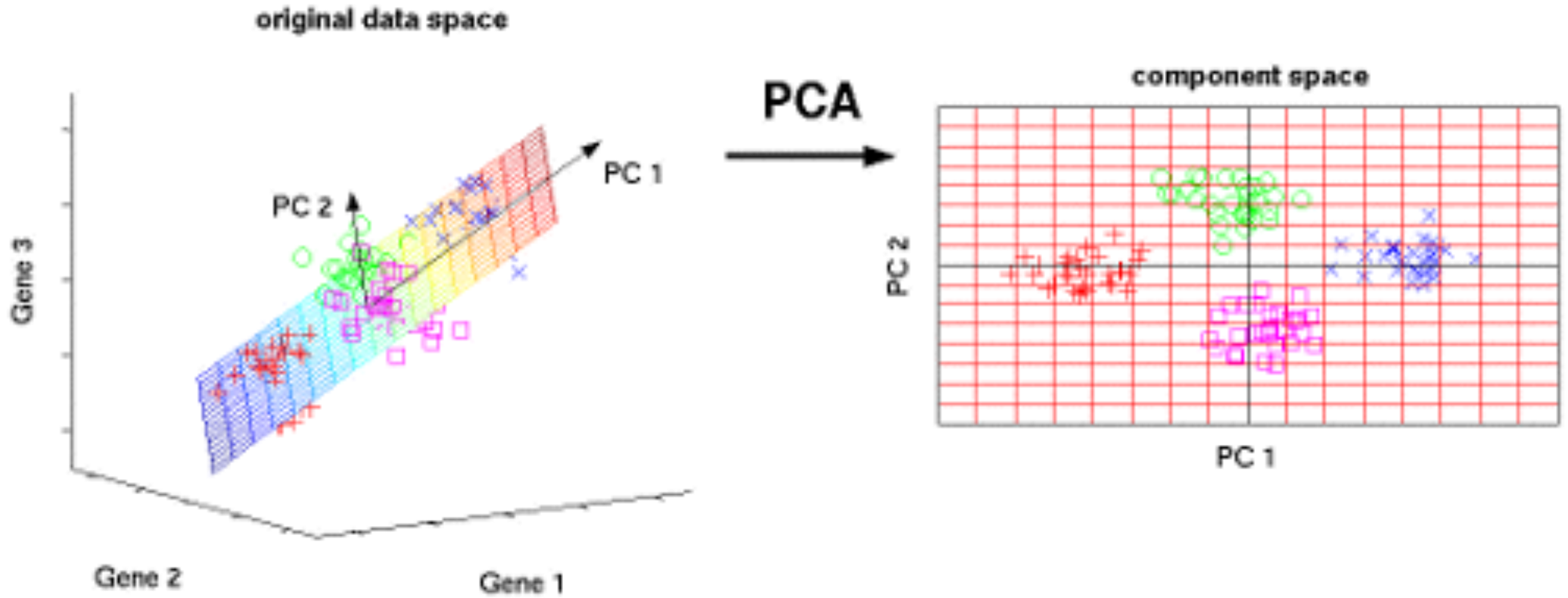


If k < d, then we obtain dimensionality reduction (PCA)

Principal components analysis

- New representation is k left singular vectors that correspond to k largest singular values
  - equivalent to k eigenvectors of covariance X^T X, that correspond to k largest eigenvalues

- Not the same as selecting k features, but rather projecting features into lower-dimensional space

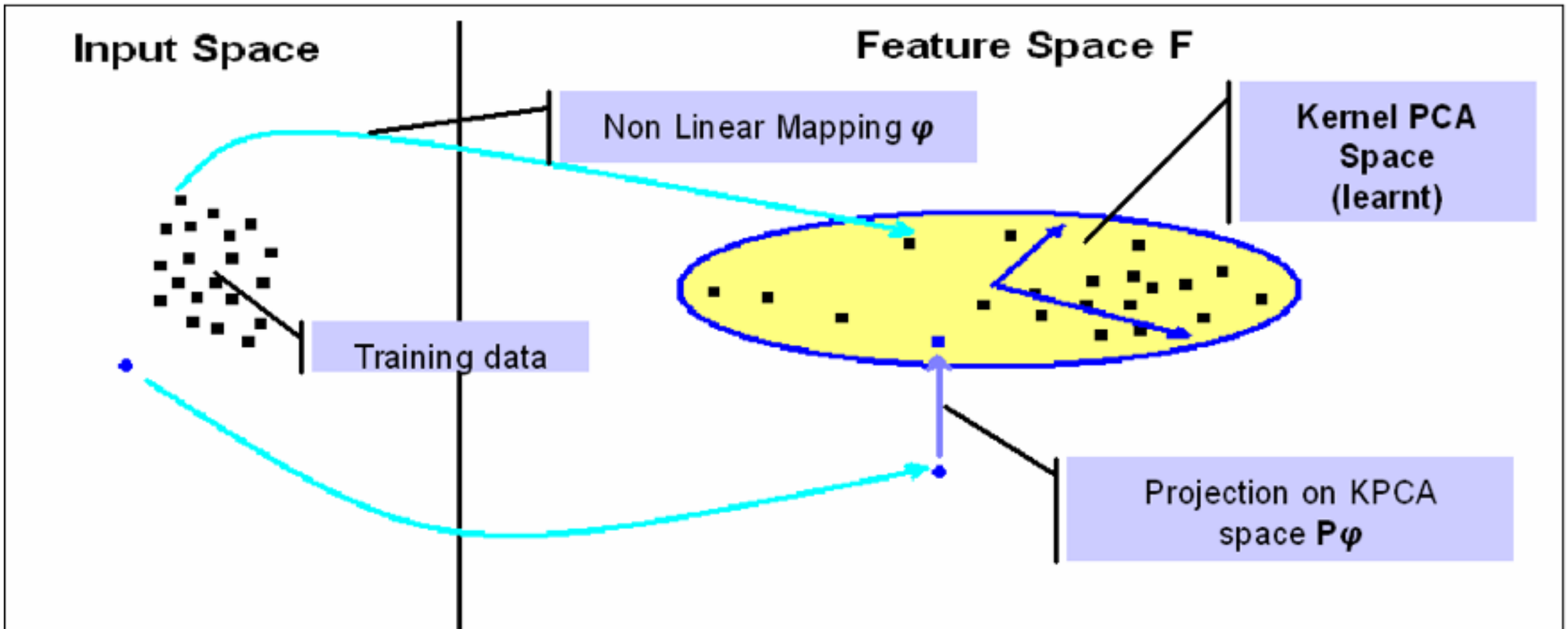


Kernel PCA

- Non-linear dimensionality reduction: PCA, but on kernel matrix K
- **Step 1:** expand/transform original features into kernel features

$$\mathbf{X} \in \mathbb{R}^{T \times d} \rightarrow \mathbf{K} \in \mathbb{R}^{T \times T} \text{ where } \mathbf{K}_{ij} = k(\mathbf{X}_{i:}, \mathbf{X}_{j:})$$

- **Step 2:** perform PCA on kernel matrix (dimensionality reduction), to get new non-linear features (rather than just linear projection)



Linear kernel PCA

- Linear kernel PCA is equivalent to PCA
- Linear kernel:

$$\mathbf{X}\mathbf{X}^\top \text{ where } k(\mathbf{X}_{i:}, \mathbf{X}_{j:}) =< \mathbf{X}_{i:}, \mathbf{X}_{j:} >= \mathbf{X}_{i:}\mathbf{X}_{j:}^\top$$

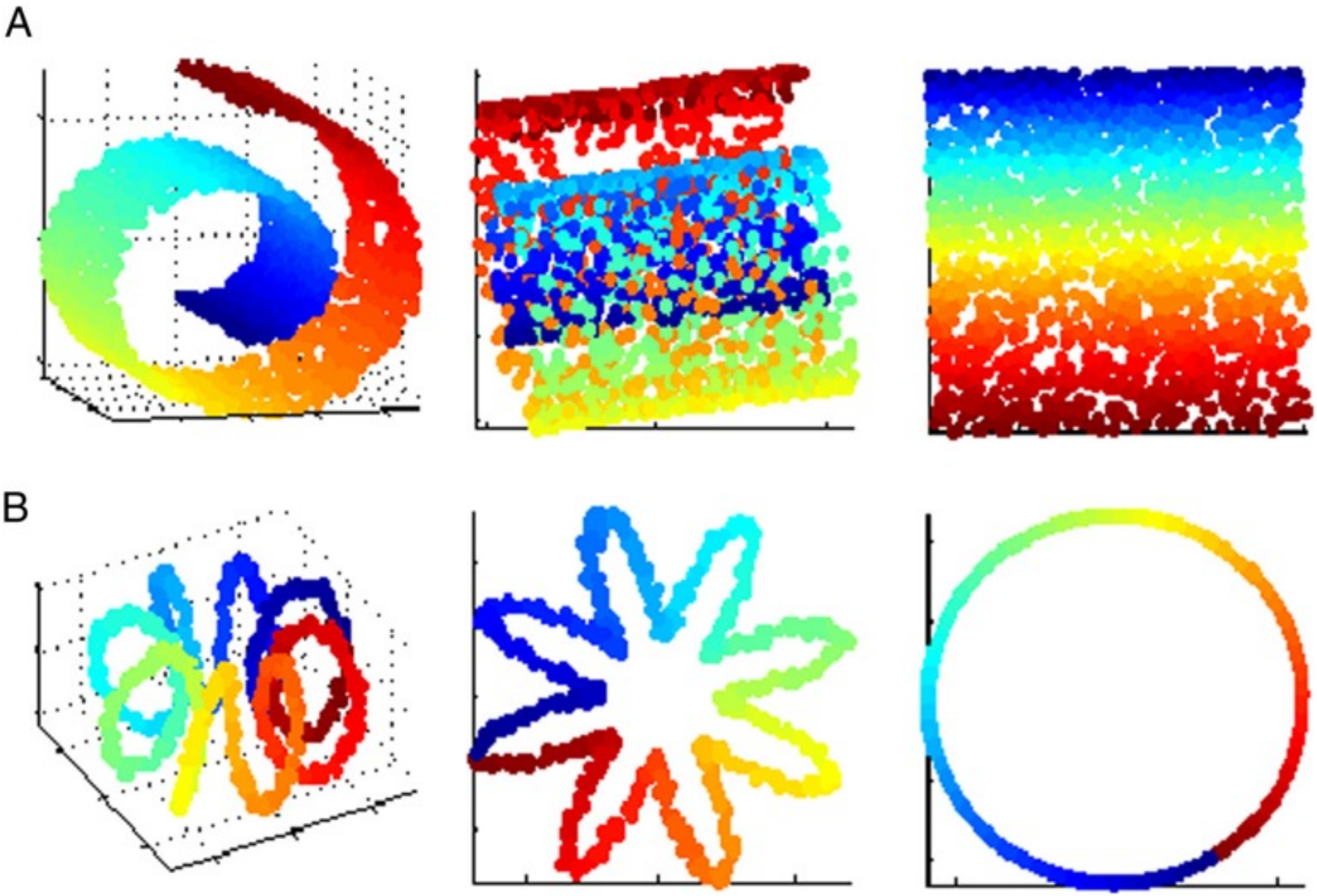
- How could you show that they give the same representation H (i.e., same principal components)?
- Hint: recall that the principal component are the left singular vectors of the matrix

Isomap

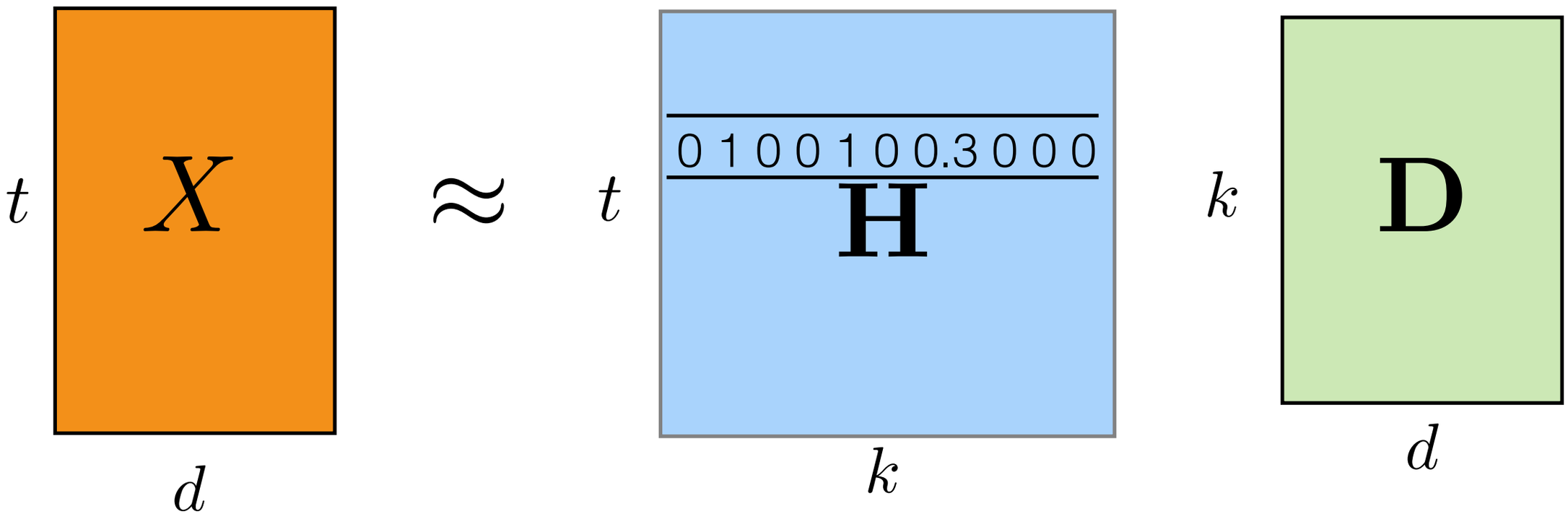
- Non-linear dimensionality reduction using a specific kernel
  - graph kernel: similarity is shortest path between two nodes, after creating a neighborhood graph
- New kernel “unfolds” the data (better preserves distances than the distance used for PCA)
- PCA uses a dot-product similarity

Isomap vs PCA

Data PCA ISOMAP



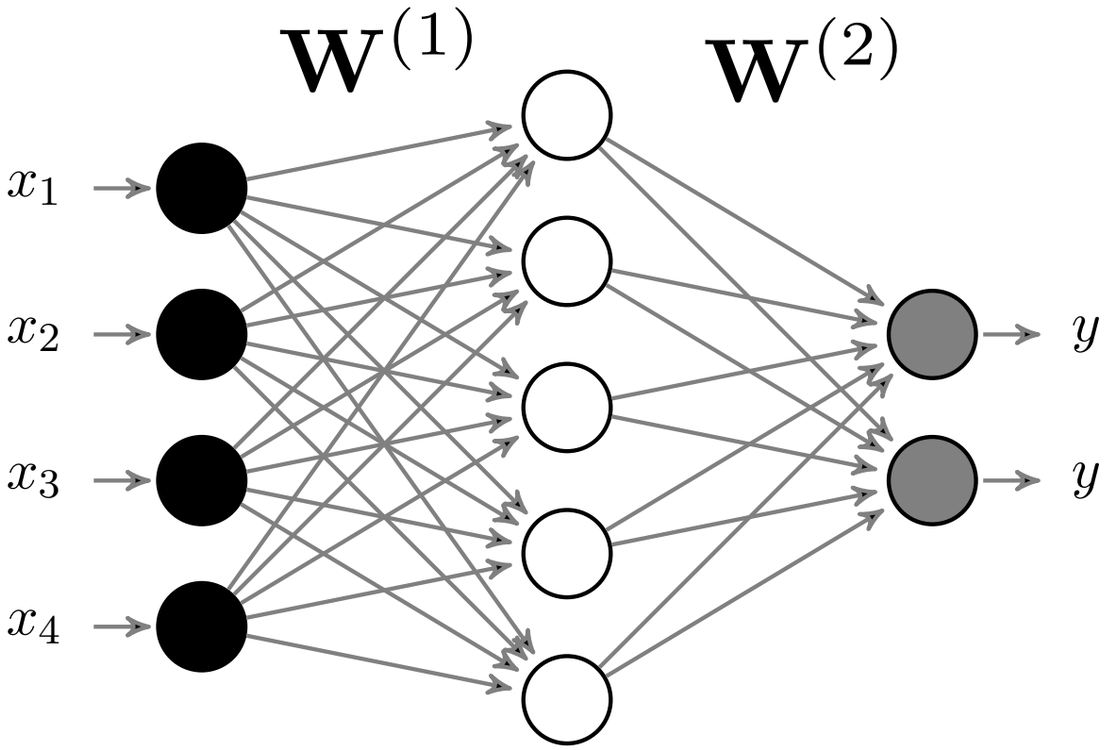
Sparse coding



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

Representation learning

Neural network

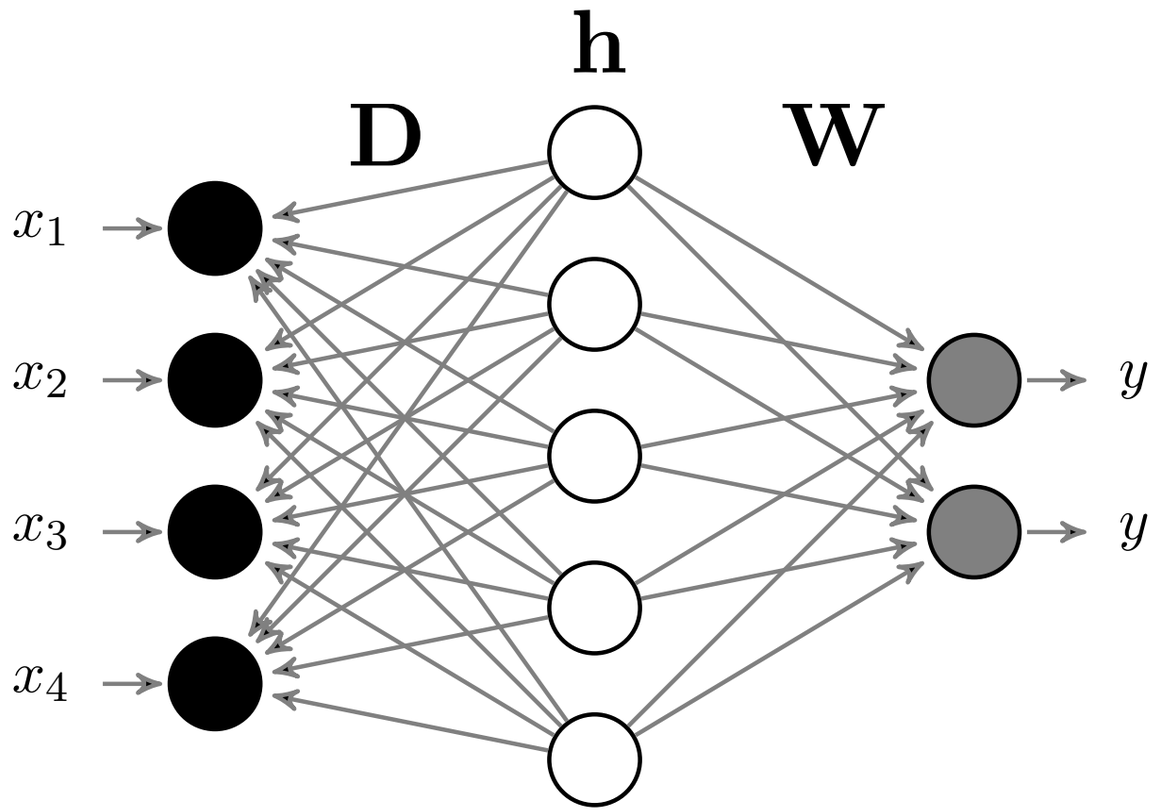


$$\mathbf{W}^{(1)} \in \mathbb{R}^{k \times d}, \mathbf{W}^{(2)} \in \mathbb{R}^{m \times k}$$

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

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

Regularized factor model



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

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

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

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

Whiteboard

- General formulation for regularized factor models
  - using regularizers to encode properties
  - formulating supervised models