COMPUTER SCIENCE

INDIANA UNIVERSITY

School of Informatics and Computing
Bloomington

Matrix factorization for representation learning (continued….)

Reminders/Comments

- Accuracy of learners:
  - Question 3: 58% for logistic regression
  - Speed of learners: with one run through about 1000 samples for training, instantaneous for all algorithms
  - 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
- Package from a fellow classmate:
  - http://sourceforge.net/projects/matlab-proper/?source=directory
- Assignment 3 in two weeks; you are hopefully simultaneously working on the project

Thought question

- In the book, they calculated gradient of log likelihood of *w*. And they mentioned that “*w* can be easily calculated using update rules like newton method.” During HW2, I failed many times gaining optimal value using newton method or gradient descent because they usually went infinity. How can we update *w* preventing infinity error? Just set low value of step (alpha)?
  - If you have a convex loss function, generally optimization is not difficult; if divergence occurring, your first thought should be that you have a bug (in general, this should be your first thought)
  - If you use line-search for batch gradient descent, you will also be fine (with a convex loss function)
  - Stochastic gradient descent more finicky for step-sizes
  - Some functions difficult to optimize: non-smooth, or functions with flat regions

Thought question

- How does the sign of lambda effect regularization? I ran the code for both positive and negative values of it. Positive values give better partition. How do I interpret this?
  - The regularization parameter is always chosen to be positive
  - A negative value for the regularizer means you want solutions that do not match the chosen regularizer; e.g., if l2 regularizer, then adding a negative value means that want larger weights (not smaller)
  - A negative regularization weight also now changes the regularization term to being concave, making the optimization non-convex

Thought question

- In some classifiers, the goal is to find the maximum of a function representing the likelihood or the log-likelihood. In others, the goal is to find the minimum of a function representing the expected value of an error. This makes mathematical and logical sense to me. I have seen some other optimization methods where a negative sign is tacked onto the log-likelihood, and this is described as cross-entropy and is somehow related to Kullback-Leibler divergence. To optimize in this instance is to find the minimum. What's the reason for doing this? At first glance it just seems to me to be taking a convex likelihood function and flipping it upside down, simply transforming the maximization problem into a minimization problem. Is there something gained by formulating the problem in this manner?
  - No, it is equivalent and is simply a matter of preference
  - minimizing KL divergence equivalent to maximum likelihood

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

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

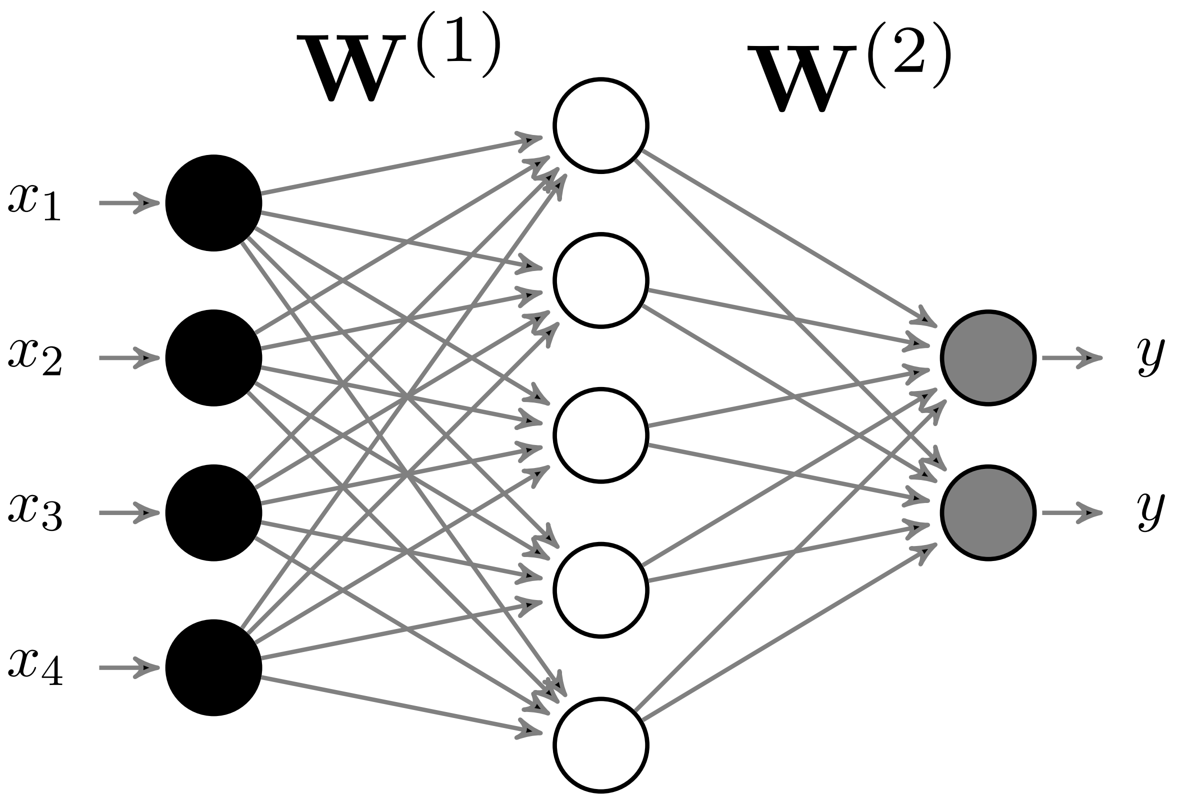
- Vector output, three-layer neural network

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

- 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

Representation learning

Neural network

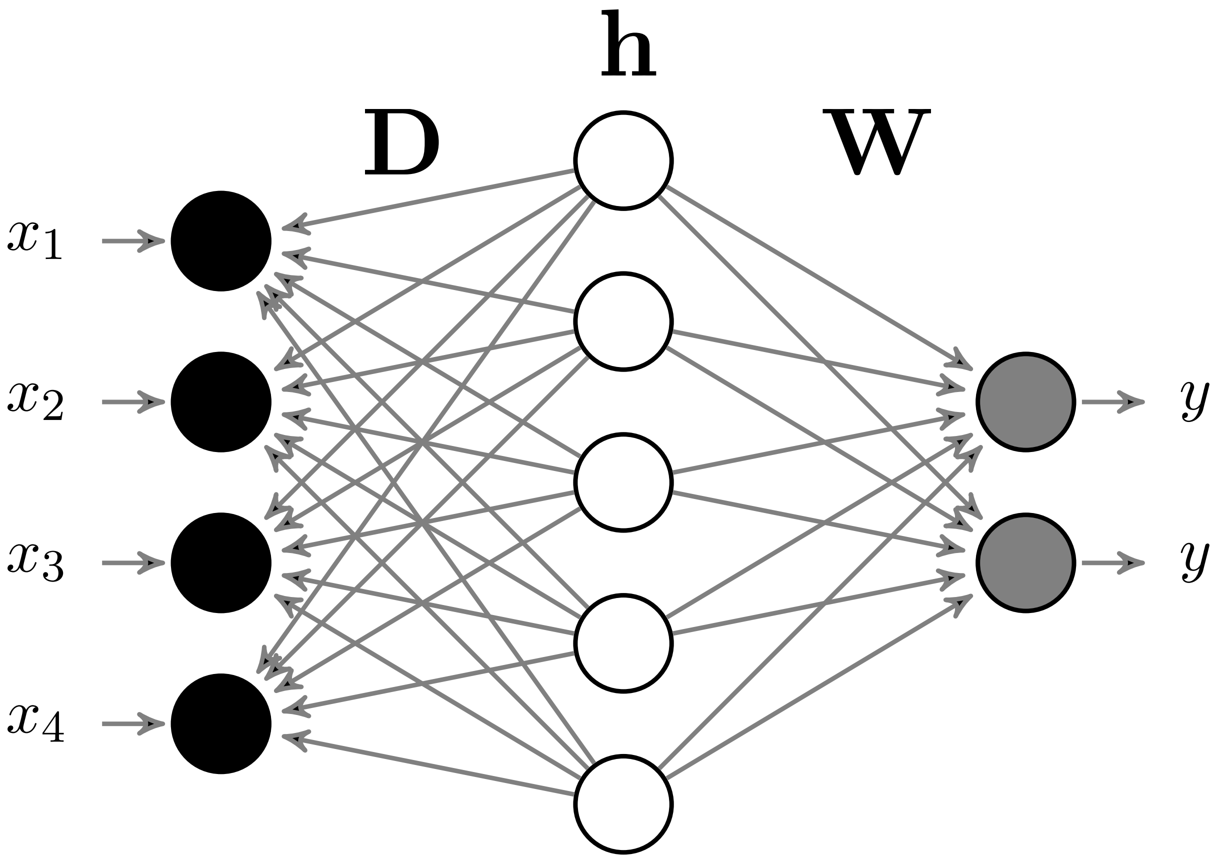


$$\mathbf{W}^{(1)} \in \mathbb{R}^{k \times d}, \mathbf{W}^{(2)} \in \mathbb{R}^{m \times k}$$

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

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

Regularized factor model



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

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

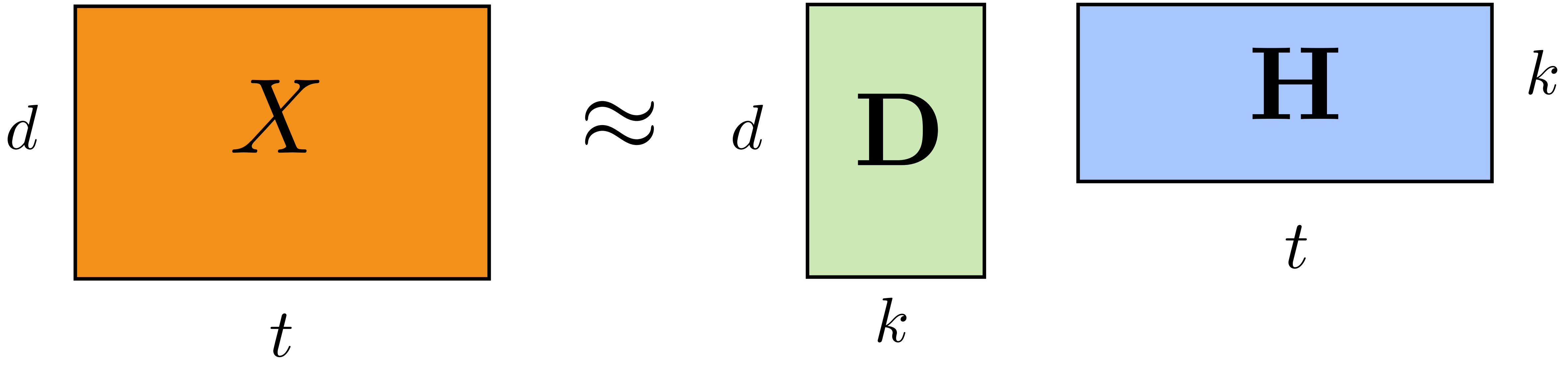
$$\hat{\mathbf{y}} = f_2(\mathbf{h}\mathbf{W})$$

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

Using factorizations

- Many problems can be formulated as factorizations, with different settings
- RFMs a useful case study to get better at specifying objectives
- They have also proven to be a simple but useful tool for solving many problems
    - e.g., matrix completion for Netflix challenge

Unsupervised RFMs



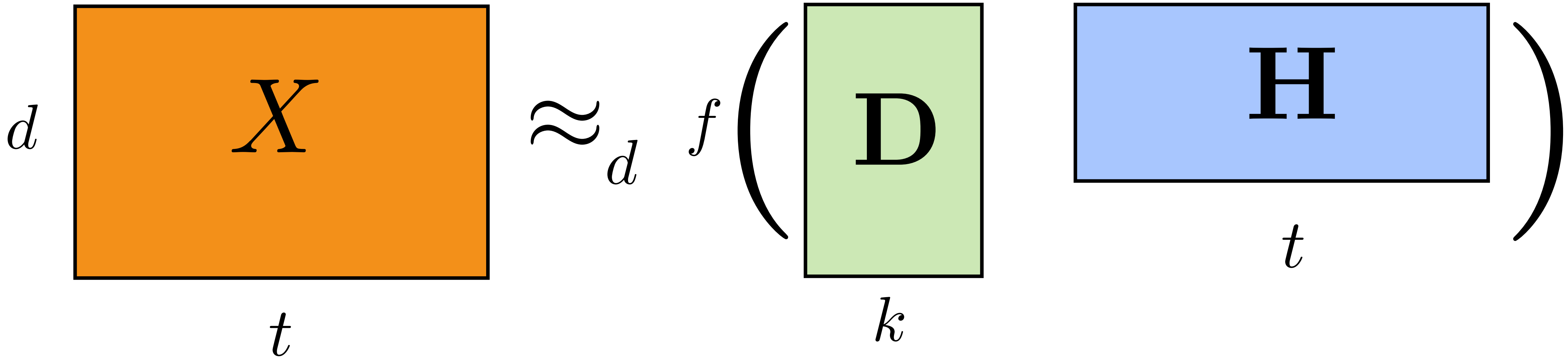
If k < d, then we obtain dimensionality reduction (PCA)

What are the distributional assumptions?

- If try to factorize X into DH, making an assumption that p(x I mu = Dh) 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 I Dh) 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

Exponential family PCA

k < d



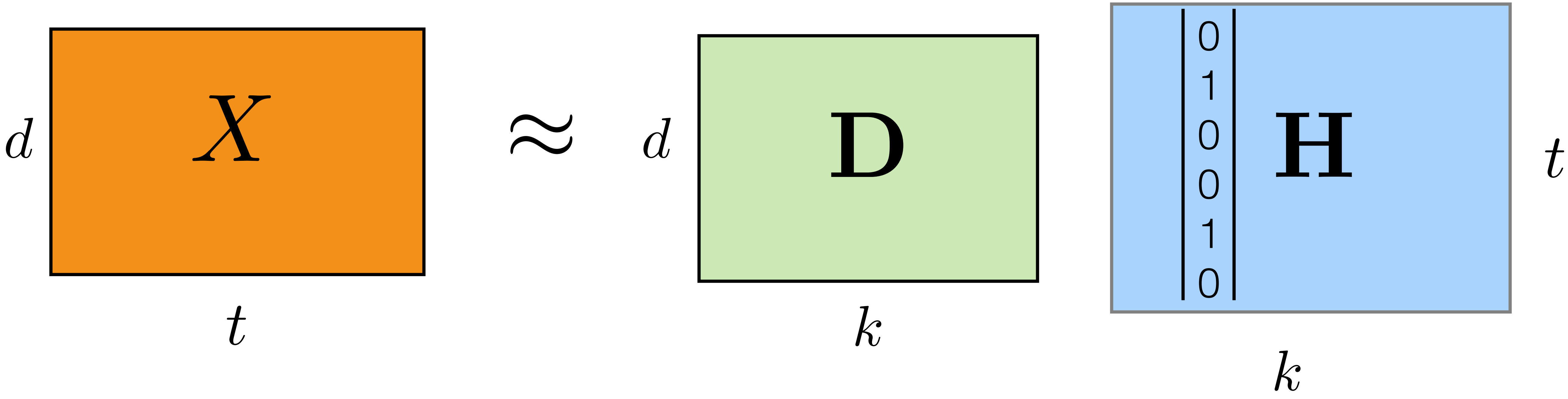
i.e., for each $\mathbf{x} = \mathbf{X}_{:i}$,

$\mathbf{x} \approx f(\mathbf{D}\mathbf{h})$

$\mathbf{h} = \mathbf{H}_{:i}$

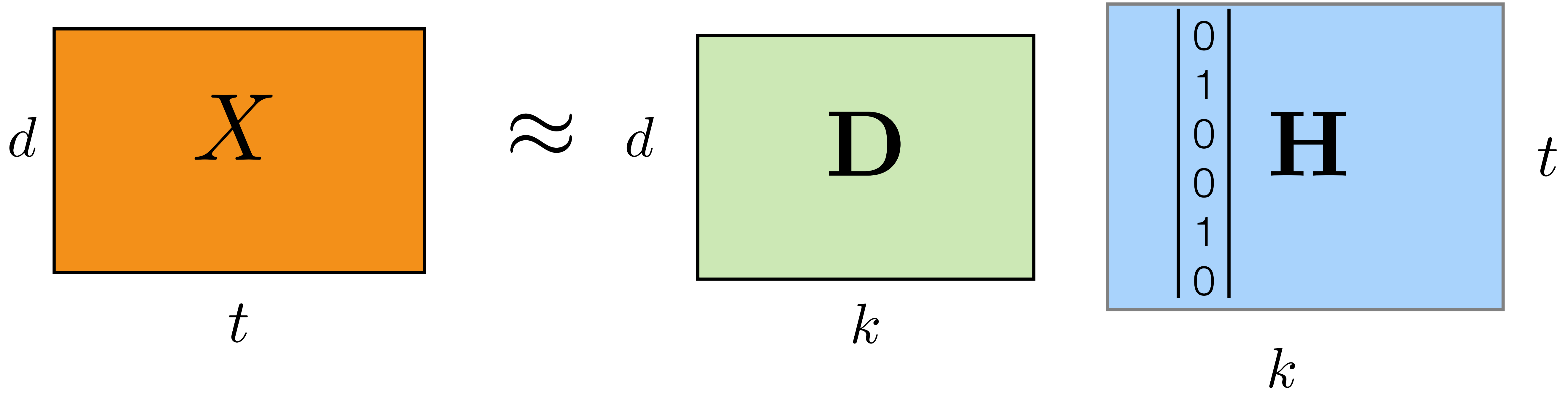
Squared loss generalized to negative log-likelihood of exponential family corresponding to transfer f

Sparse coding



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

Sparse coding



$$L(\hat{\mathbf{x}}, \mathbf{x}) = \|\hat{\mathbf{x}} - \mathbf{x}\|_2^2$$

$$R_H(\mathbf{H}) = \|\mathbf{H}\|_{1,1} = \sum_{i=1}^{T} \|\mathbf{H}_{:i}\|_1$$

prefers to zero entries in H

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

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

Optimizing RFMs

- We have the objective:

$$\min_{D \in \mathbb{R}^{d \times k}, H \in \mathbb{R}^{k \times t}} \sum_{i=1}^{t} L(\mathbf{DH}_{:i}, \mathbf{X}_{:i}) + \lambda R_D(\mathbf{D}) + \lambda R_H(\mathbf{H})$$

- How do we optimize it?
- For several settings, we have closed form solutions
  - PCA, CCA, ISOMAP, …
- For others, we do not
  - sparse coding, exponential family PCA, …

Least-squares loss form

- Well-known solution to

$$\min_{D \in \mathbb{R}^{d \times k}, H \in \mathbb{R}^{k \times t}} \|\mathbf{X} - \mathbf{D}\mathbf{H}\|_F^2$$

- is to take the singular value decomposition of X and set D = U Sigma and H = V^T

- Multiple ways to arrive at this solution
  - Exercise: solve closed form for D first, then plug back in and reduce. You will end up with the optimization

$$\max_{H \in \mathbb{R}^{k \times t} : H^\top H = I} \operatorname{tr}(\mathbf{H}^\top \mathbf{H} \mathbf{X}^\top \mathbf{X})$$

  - The well-known solution to this maximization is the top k eigenvectors of X^T X (i.e., right singular vectors of X)

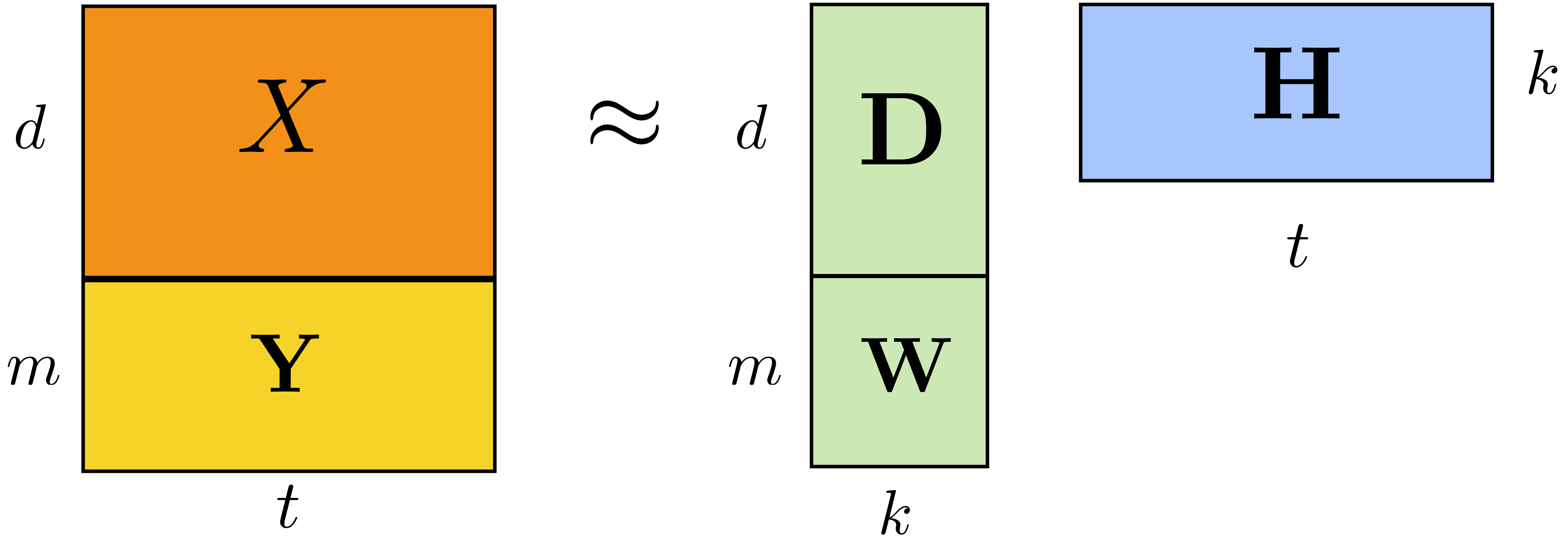
For other settings

- 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)

Supervised RFMs



As with generalized linear models, can use a nonlinear transfer (e.g., sigmoid)

Whiteboard

- Exercise: alternating descent for sparse coding
- Three layer neural network