CPSC 340: Machine Learning and Data Mining

Recommender Systems

Spring 2022 (2021W2)

Admin

- A5 due this Friday (midnight)
- Final exam format will be similar to the midterm. Online (Canvas).

Last Few Lectures: Latent-Factor Models

- We've been discussing latent-factor models of the form:

$$f(W,Z) = \sum_{i=1}^{n}\sum_{j=1}^{d}(\langle w^j, z_i\rangle - x_{ij})^2$$

- We get different models under different conditions:
  - K-means: each z_i has one '1' and the rest are zero.
  - Least squares: we only have one variable (d=1) and the z_i are fixed.
  - PCA: no restrictions on W or Z.
    - Orthogonal PCA (usual case): the rows w_c have norm 1 and inner products of zero.
  - NMF: all elements of W and Z are non-negative.

Variations on Latent-Factor Models

- We can use all our tricks for linear regression in this context:

$$f(W,Z) = \sum_{i=1}^{n}\sum_{j=1}^{d} |\langle w^j, z_i\rangle - x_{ij}| + \frac{\lambda_1}{2}\sum_{i=1}^{n}\sum_{c=1}^{k} z_{ic}^2 + \frac{\lambda_2}{2}\sum_{j=1}^{d}\sum_{c=1}^{k}|w_{cj}|$$

- Absolute loss gives robust PCA that is less sensitive to outliers.
- We can use L2-regularization.
  - Though only reduces overfitting if we regularize both 'W' and 'Z'.
- We can use L1-regularization to give sparse latent factors/features.
- We can use logistic/softmax/Poisson losses for discrete x_{ij}.
- Can use change of basis to learn non-linear latent-factor models.

bonus!

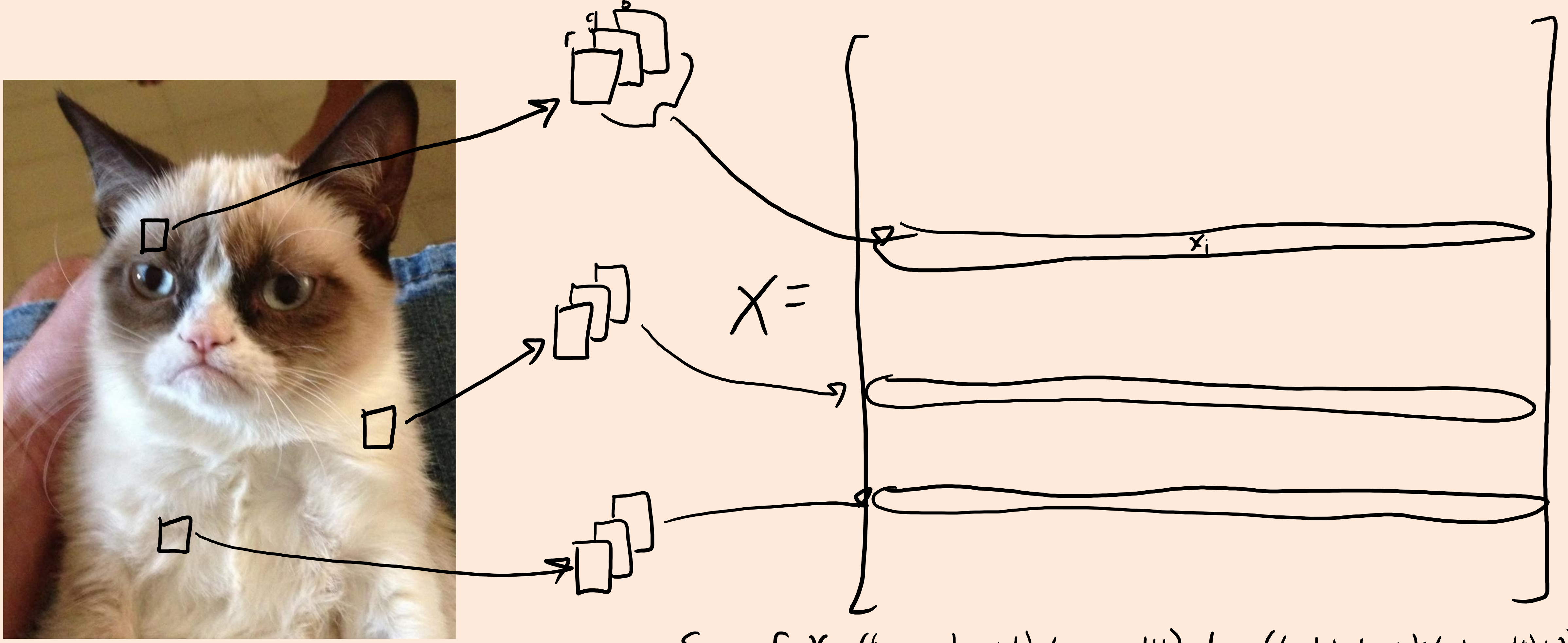
Application: Image Restoration

http://www.jmlr.org/papers/volume11/mairal10a/mairal10a.pdf

bonus!

Latent-Factor Models for Image Patches

- Consider building latent-factors for general image patches:

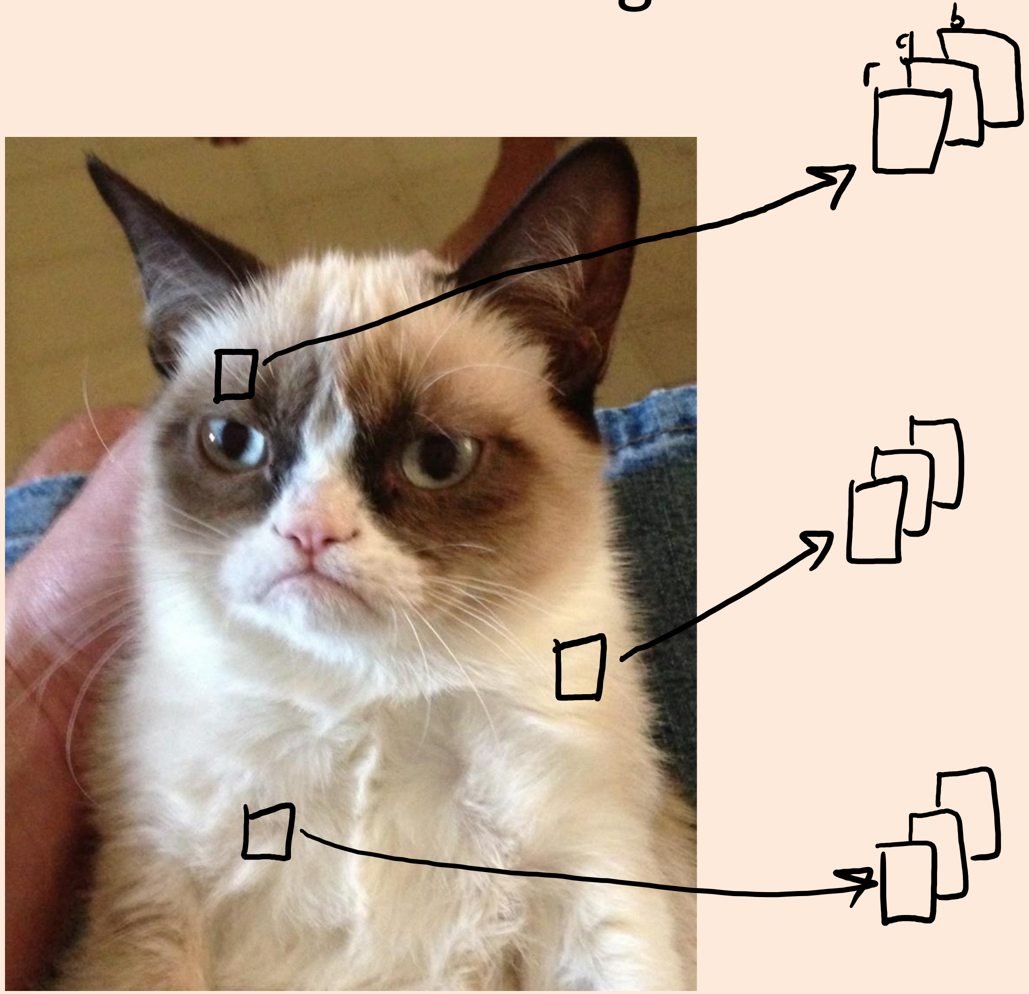


Size of X: (image height)×(image width) by ((patch height)×(patch width)×3)

bonus!

Latent-Factor Models for Image Patches

- Consider building latent-factors for general image patches:



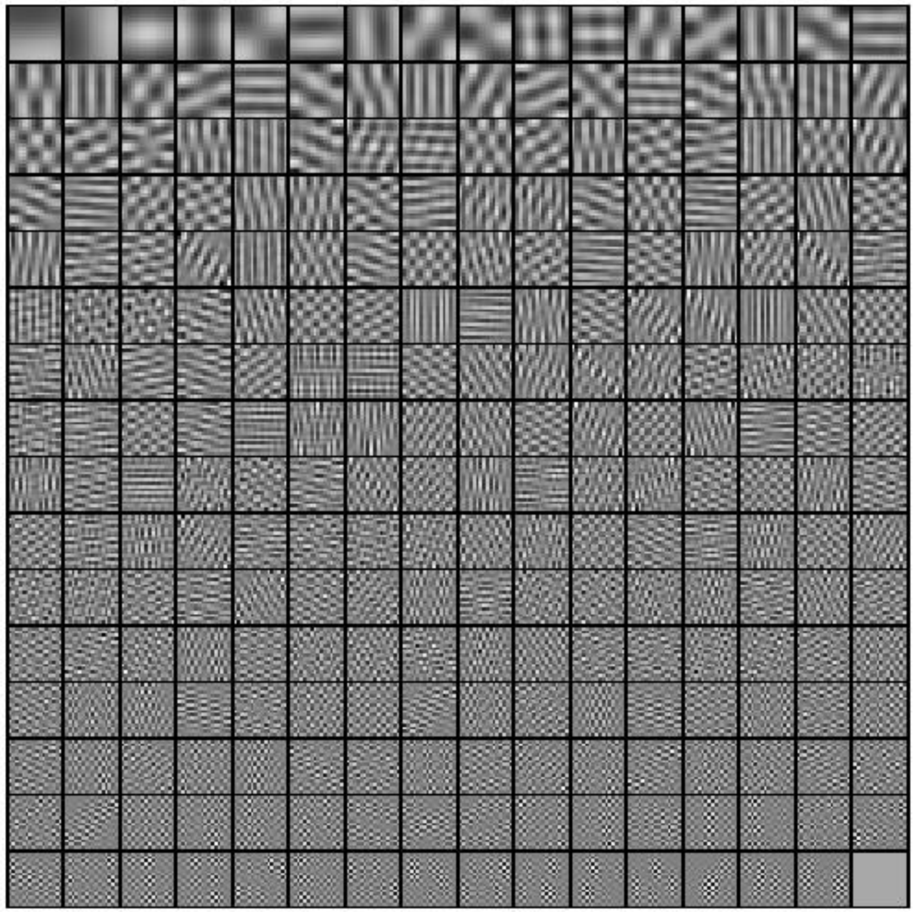
Typical pre-processing:

1. Usual variable centering
2. “Whiten” patches.

(remove correlations - bonus)

bonus!

Latent-Factor Models for Image Patches

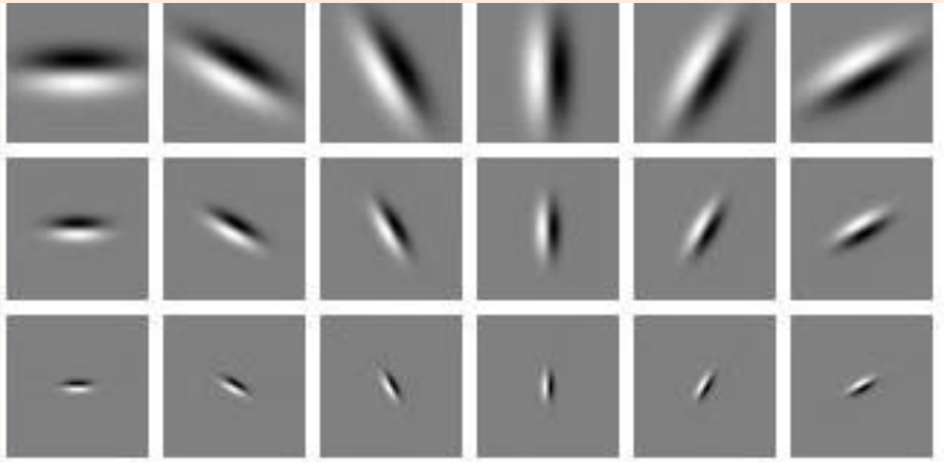


(b) Principal components.

Orthogonal bases don't seem right:

- Few PCs do almost everything.
- Most PCs do almost nothing.

We believe "simple cells" in visual cortex use:



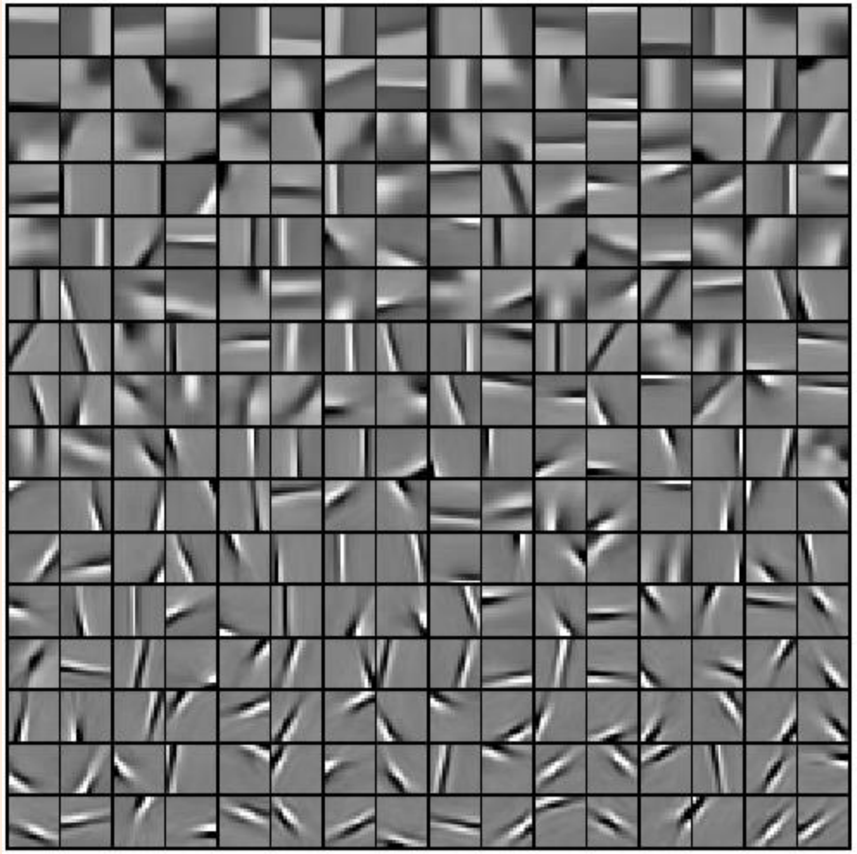
'Gabor' filters

http://lear.inrialpes.fr/people/mairal/resources/pdf/review_sparse_arxiv.pdf
http://stackoverflow.com/questions/16059462/comparing-textures-with-opencv-and-gabor-filters

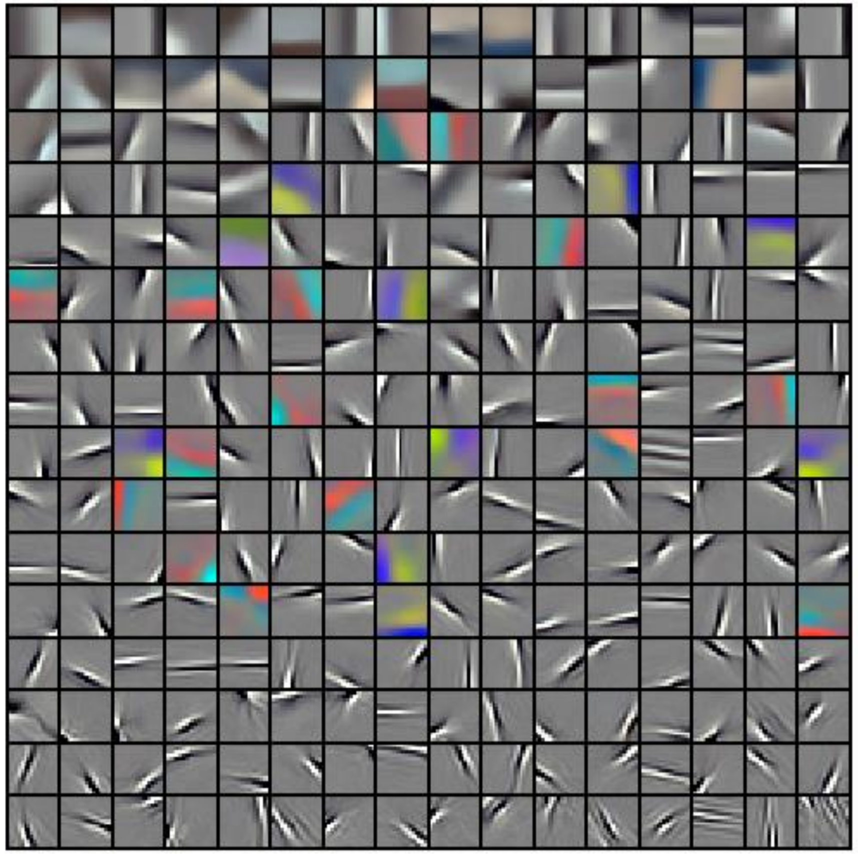
bonus!

Latent-Factor Models for Image Patches

- Results from a "sparse" (non-orthogonal) latent factor model:



(a) With centering - gray.



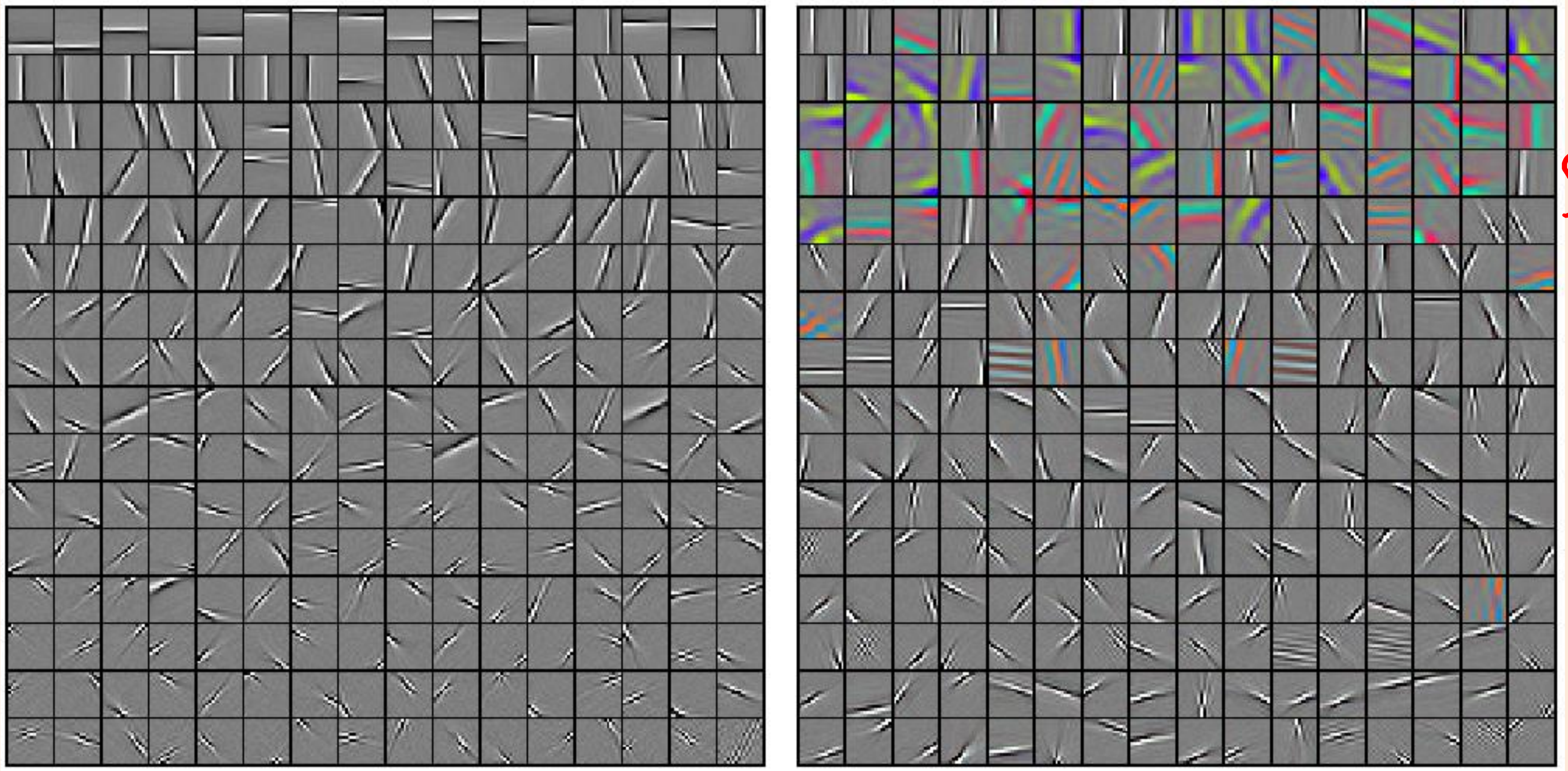
(b) With centering - RGB.

http://lear.inrialpes.fr/people/mairal/resources/pdf/review_sparse_arxiv.pdf

bonus!

Latent-Factor Models for Image Patches

- Results from a "sparse" (non-orthogonal) latent-factor model:



(c) With whitening - gray. (d) With whitening - RGB.

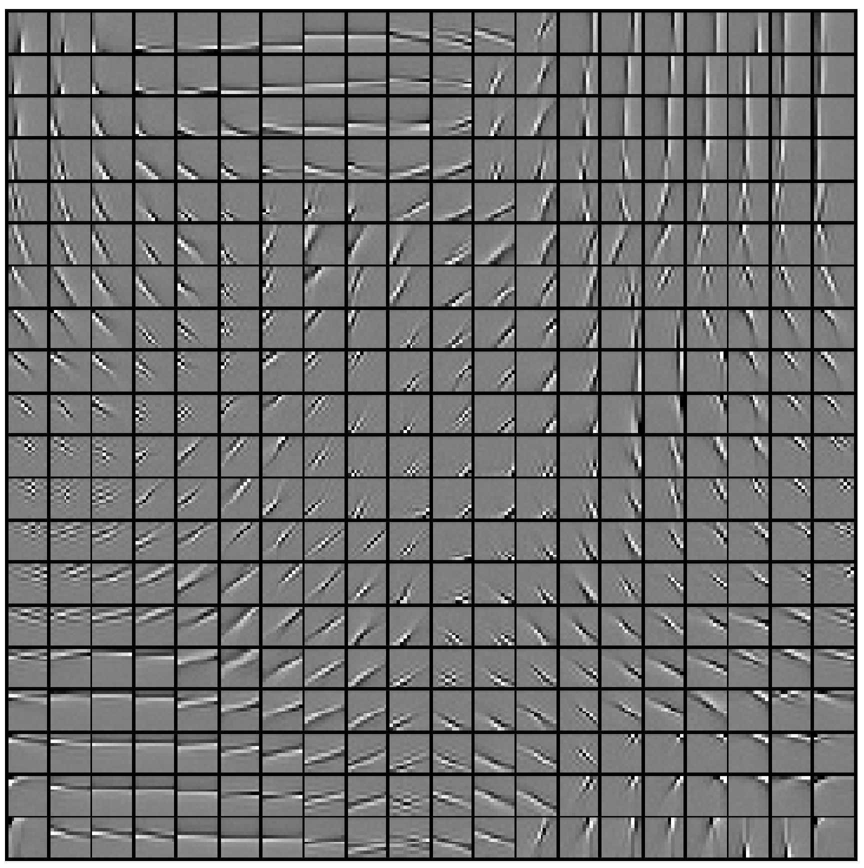
"colour opponency"

http://lear.inrialpes.fr/people/mairal/resources/pdf/review_sparse_arxiv.pdf

bonus!

Recent Work: Structured Sparsity

- Basis learned with a variant of "structured sparsity":



(b) With 4 × 4 neighborhood.

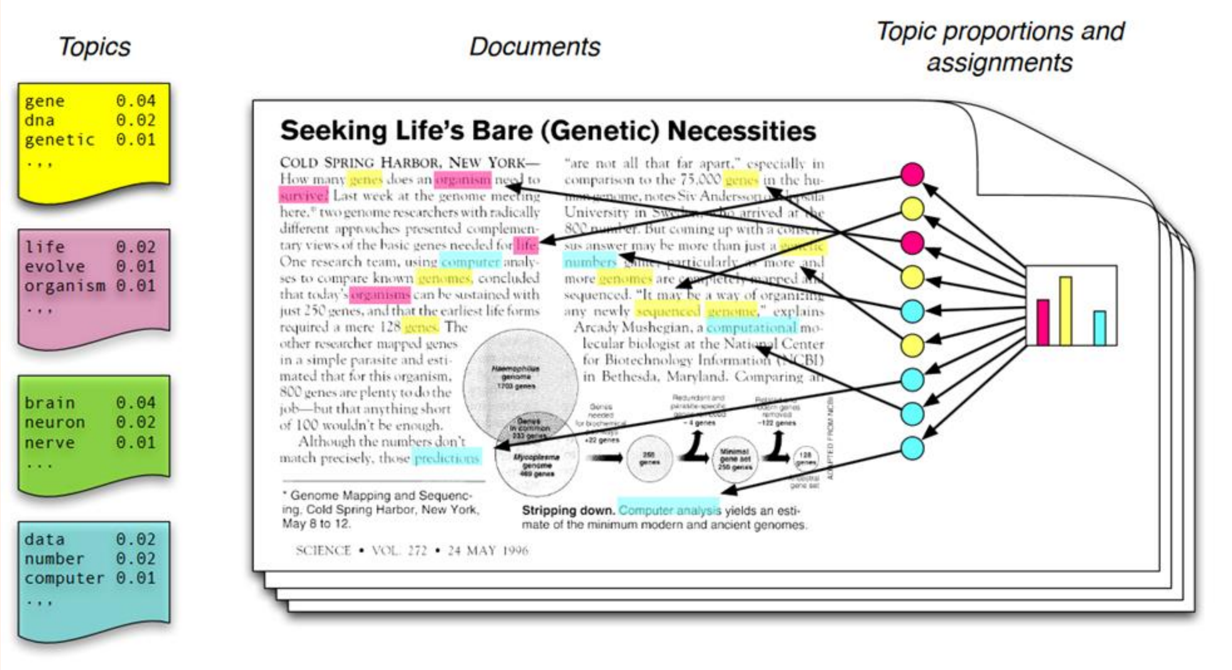
Similar to "cortical columns" theory in visual cortex.

http://lear.inrialpes.fr/people/mairal/resources/pdf/review_sparse_arxiv.pdf

bonus!

Beyond NMF: Topic Models

- For modeling data as combinations of non-negative parts, NMF has largely been replaced by "topic models".
  - A "fully-Bayesian" model where sparsity arises naturally.
  - Most popular example is called "latent Dirichlet allocation" (CPSC 440).



(pause)

Recommender System Motivation: Netflix Prize

- Netflix Prize:
  - 100M ratings from 0.5M users on 18k movies.
  - Grand prize was $1M for first team to reduce squared error by 10%.
  - Started on October 2nd, 2006.
  - Netflix's system was first beat October 8th.
  - 1% error reduction achieved on October 15th.
  - Steady improvement after that.
    - ML methods soon dominated.
  - One obstacle was 'Napoleon Dynamite' problem:
    - Some movie ratings seem very difficult to predict.
    - Should only be recommended to certain groups.

Lessons Learned from Netflix Prize

- Prize awarded in 2009:
  - Ensemble method that averaged 107 models.
  - Increasing diversity of models more important than improving models.

- Winning entry (and most entries) used collaborative filtering:
  - Methods that only looks at ratings, not features of movies/users.
- A simple collaborative filtering method that does really well (7%):
  - "Regularized matrix factorization". Now adopted by many companies.

http://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest/?_r=0

Motivation: Other Recommender Systems

- Recommender systems are now everywhere:
  - Music, news, books, jokes, experts, restaurants, friends, dates, etc.
- Main types of approaches:

1. Content-based filtering.
   - Supervised learning:
     - Extract features x_i of users and items, building model to predict rating y_i given x_i.
     - Apply model to prediction for new users/items.
   - Example: G-mail's "important messages" (personalization with "local" features).
2. Collaborative filtering.
   - "Unsupervised" learning (have label matrix 'Y' but no features):
     - We only have labels y_{ij} (rating of user 'i' for movie 'j').
   - Example: Amazon recommendation algorithm.

Collaborative Filtering Problem

- Collaborative filtering is 'filling in' the user-item matrix:

$$Y = \text{user}\left\{\underbrace{\begin{bmatrix} ? & 4 & 3 & 2 & 3 & 3 \\ 2 & 1 & ? & 5 & ? & 5 \\ ? & 1 & ? & 5 & 5 & 5 \\ 2 & 3 & 3 & ? & ? & ? \end{bmatrix}}_{\text{movie}}\right.$$

"Green Lantern"

"Ryan Reynolds"

- We have some ratings available with values {1, 2, 3, 4, 5}.
- We want to predict ratings "?" by looking at available ratings.

Collaborative Filtering Problem

- Collaborative filtering is 'filling in' the user-item matrix:

$$Y = \text{user}\left\{\underbrace{\begin{bmatrix} ? & 4 & 3 & 2 & 3 & 3 \\ 2 & 1 & ? & 5 & ? & 5 \\ ? & 1 & ? & 5 & 5 & 5 \\ 2 & 3 & 3 & ? & ? & ? \end{bmatrix}}_{\text{movie}}\right.$$

"Green Lantern"

"Ryan Reynolds"

Similar user

Similar movie

- What rating would "Ryan Reynolds" give to "Green Lantern"?
  - Why is this not completely hopeless? It *could* be anything.
  - But we may have similar users and movies.

Matrix Factorization for Collaborative Filtering

- Our standard latent-factor model for entries in matrix 'Y':

$$\underset{n\times d}{Y} \approx \underset{n\times k}{Z}\,\underset{k\times d}{W} \qquad y_{ij} \approx \langle w^j, z_i \rangle$$

- User 'i' has latent features z_i.
  - $\to$ z_{ic} could mean "likes Nicolas Cage"
- Movie 'j' has latent features w^j.
  - $\to$ w_{jc} could mean "has Nicholas Cage"
- Our loss functions sums over available ratings 'R':

$$f(Z,W) = \sum_{(i,j)\in R} \left(\langle w^j, z_i \rangle - y_{ij}\right)^2 + \frac{\lambda_1}{2}\|Z\|_F^2 + \frac{\lambda_2}{2}\|W\|_F^2$$

- And we add L2-regularization to both types of features.
  - Basically, this is regularized PCA on the available entries of Y.
  - Typically fit with SGD.
- This simple method gives you a 7% improvement on the Netflix problem.

bonus!

Adding Global/User/Movie Biases

- Our standard latent-factor model for entries in matrix 'Y':

$$\hat{y}_{ij} = \langle w^j, z_i \rangle$$

- Sometimes we don't assume the y_{ij} have a mean of zero:
  - We could add bias β reflecting average overall rating:

$$\hat{y}_{ij} = \beta + \langle w^j, z_i \rangle$$

  - We could also add a user-specific bias β_i and item-specific bias β_j.

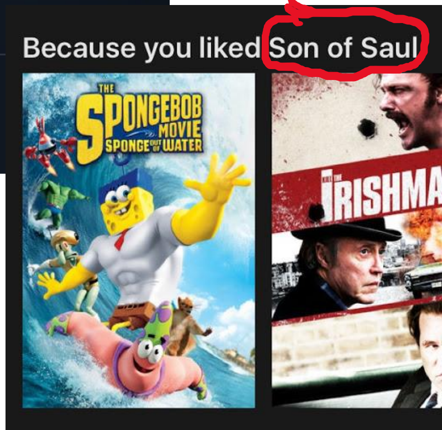
$$\hat{y}_{ij} = \beta + \beta_i + \beta_j + \langle w^j, z_i \rangle$$

    - Some users are more generous, and some movies are just better.
    - These might also be regularized.

Beyond Accuracy in Recommender Systems

- Winning system of Netflix Challenge was never adopted.
- Other issues important in recommender systems:
  - Diversity: how different are the recommendations?
    - If you like ‘Battle of Five Armies Extended Edition’, recommend ‘Battle of Five Armies’?
    - Even if you really really like Star Wars, you might want non-Star-Wars suggestions.
  - Persistence: how long should recommendations last?
    - If you keep not clicking on ‘Justice League’, should it go away?
  - Trust: tell user *why* you made a recommendation.
  - Social recommendation: what did your friends watch?
  - Freshness: people tend to get more excited about *new/surprising* things.
    - Collaborative filtering does not predict well for new users/movies.
      - New movies don’t yet have ratings, and new users haven’t rated anything.

an intense Holocaust drama



Content-Based vs. Collaborative Filtering

- Our latent-factor approach to collaborative filtering (Part 4):

$$\hat{y}_{ij} = \langle \underbrace{w^j}_{\text{"hidden" features of movie}}, \underbrace{z_i}_{\text{"hidden" features of user}} \rangle$$

  – Learns about each user/movie, but can't predict on new users/movies.

- A linear model approach to content-based filtering (Part 3):

$$\hat{y}_{ij} = w^T x_{ij}$$

Our usual $y_i = w^T x_i$ supervised learning setup.

  – Here x_{ij} is a **vector of features** for the movie/user.
    - Usual supervised learning setup: 'y' would contain all the y_{ij}, X would have x_{ij} as rows.
  – Can predict on new users/movies, but can't learn about each user/movie.

bonus!

Hybrid Approaches

- Hybrid approaches combine content-based/collaborative filtering:
  - SVDfeature (won "KDD Cup" in 2011 and 2012).

$$\hat{y}_{ij} = \underbrace{\beta}_{\text{Average rating across all users/movies}} + \underbrace{\beta_i}_{\text{Average rating for user 'i'}} + \underbrace{\beta_j}_{\text{Average for movie 'j'}} + \underbrace{w^T x_{ij}}_{\text{Linear model based on user/movie features } x_{ij}} + \underbrace{\langle w^j, z_i \rangle}_{\text{Latent features } z_i \text{ for user 'i' and latent features } w_j \text{ for movie 'j'}}$$

Extra factors we learn for specific users and movies.

Standard supervised learning: can predict for new users/movies

  - Note that x_{ij} is a feature *vector*. Also, 'w' and 'w^j' are different parameters.

bonus!

Stochastic Gradient for SVDfeature

- Common approach to fitting SVDfeature is stochastic gradient.
- Previously you saw stochastic gradient for supervised learning:
  - Choose a random example 'i'
  - Update parameters 'w' using gradient of example 'i'
- Stochastic gradient for SVDfeature (formulas as bonus):
  - Choose a random user 'i' and a random product 'j'
  - Update β, β_i, β_j, w, z_i, and w^j based on their gradient for this user-product.

Updated every time

bonus!

Social Regularization

- Many recommenders are now connected to social networks.
  - "Login using your Facebook account".

- Often, people like similar movies to their friends.

- Recent recommender systems use social regularization.
  - Add a "regularizer" encouraging friends' weights to be similar:

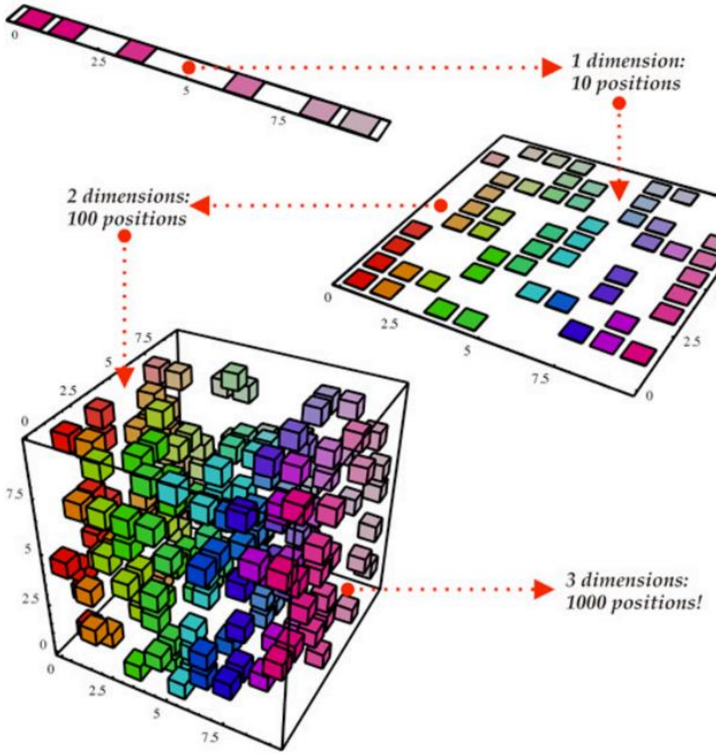
$$\frac{\lambda}{2} \sum_{(i,j) \in \text{"friends"}} \|z_i - z_j\|^2$$

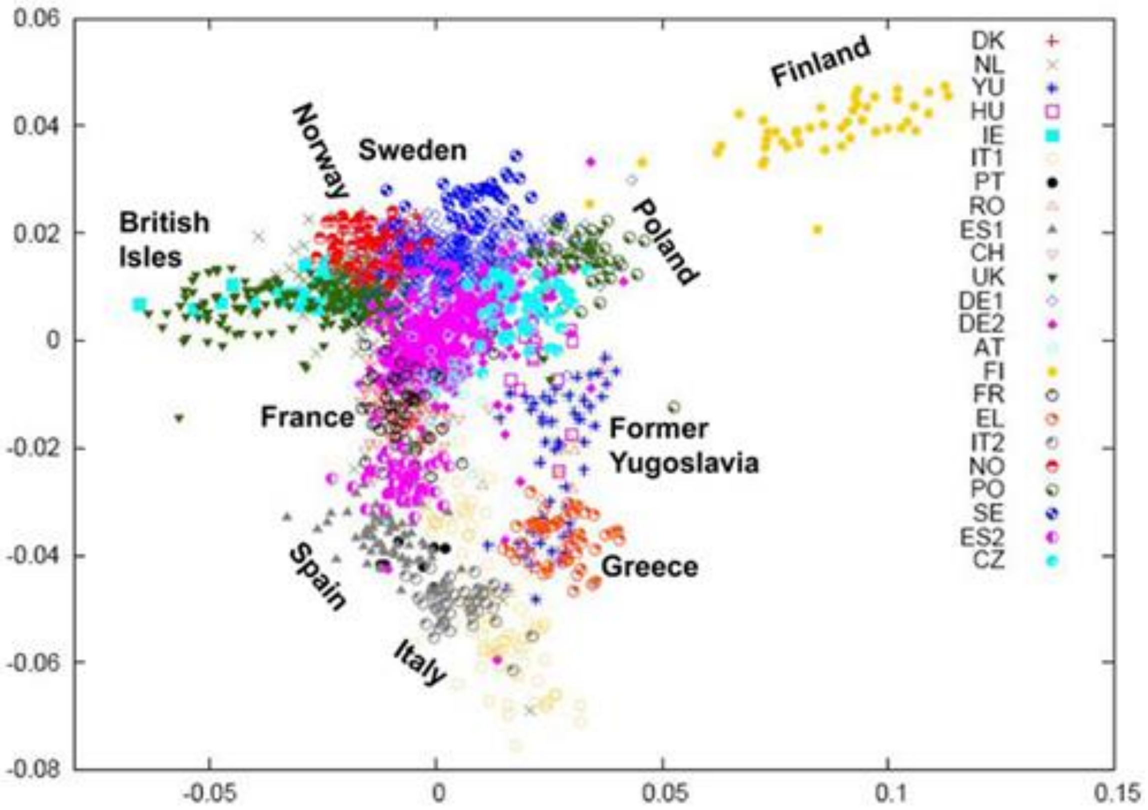
  - If we get a new user, recommendations are based on friend's preferences.

(pause)

Latent-Factor Models for Visualization

- PCA takes features x_i and gives k-dimensional approximation z_i.
- If k is small, we can use this to visualize high-dimensional data.

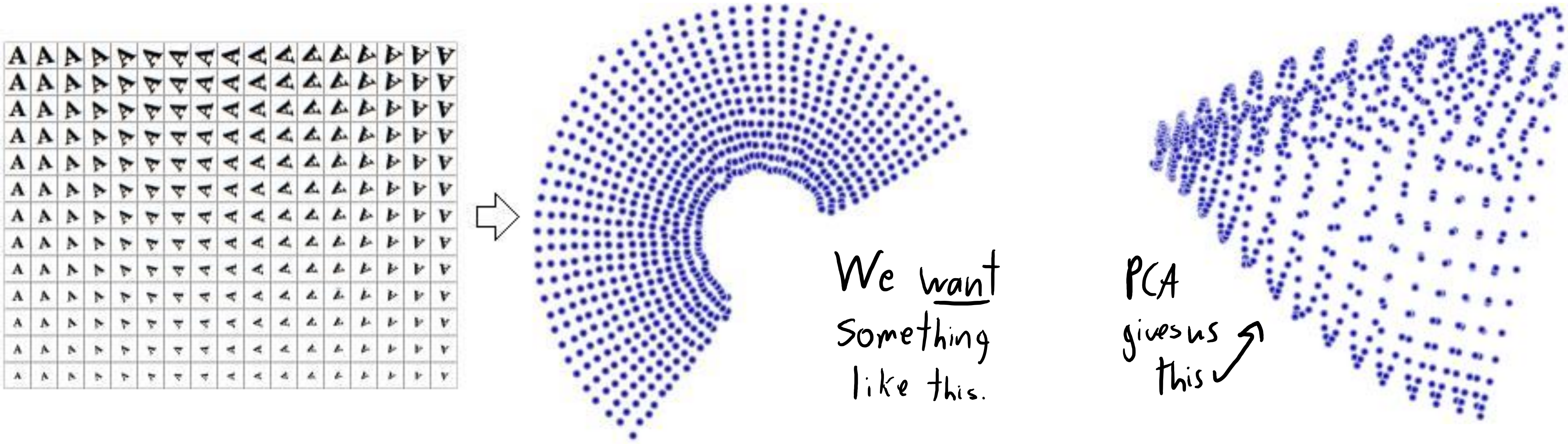




http://www.turingfinance.com/artificial-intelligence-and-statistics-principal-component-analysis-and-self-organizing-maps/
http://scienceblogs.com/gnxp/2008/08/14/the-genetic-map-of-europe/

Motivation for Non-Linear Latent-Factor Models

- But PCA is a parametric linear model
- PCA may not find obvious low-dimensional structure.



- We could use change of basis or kernels: but still need to pick basis.

https://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction

Multi-Dimensional Scaling

- PCA for visualization:
  - We're using PCA to get the location of the z_i values.
  - We then plot the z_i values as locations in a scatterplot.
- Multi-dimensional scaling (MDS) is a crazy idea:
  - Let's directly optimize the pixel locations of the z_i values.
    - "Gradient descent on the points in a scatterplot".
  - Needs a "cost" function saying how "good" the z_i locations are.
    - Traditional MDS cost function:

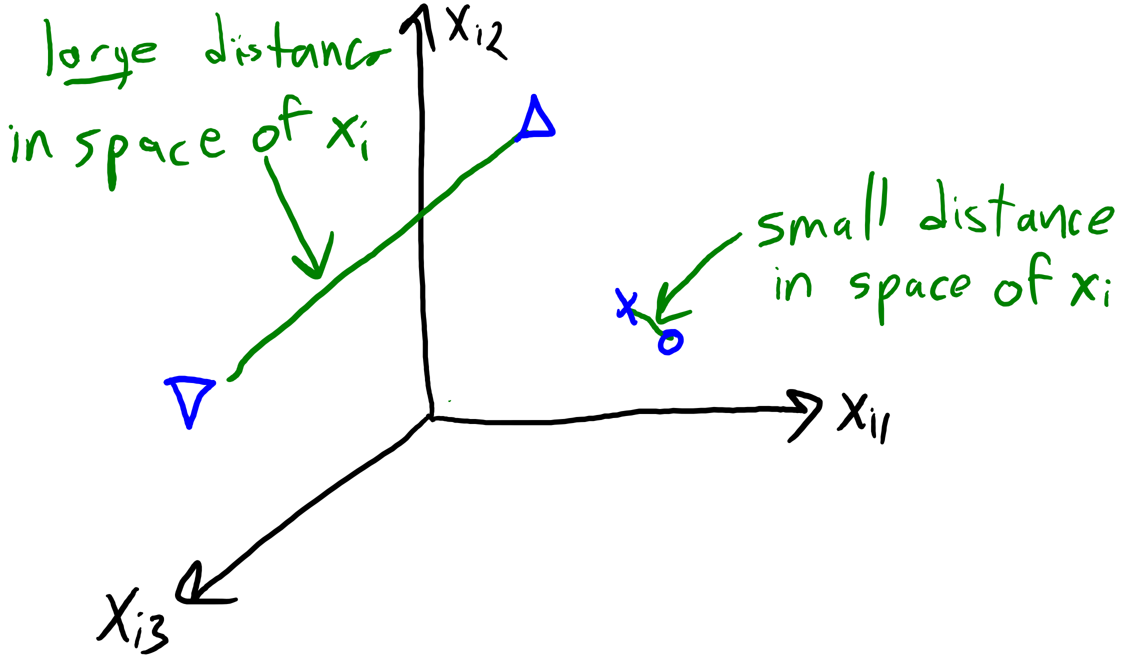
$$f(Z) = \underbrace{\sum_{i=1}^{n}\sum_{j=i+1}^{n}}_{\text{sum over pairs of examples}} \left(\underbrace{\|z_i - z_j\|}_{\text{distance in scatterplot}} - \underbrace{\|x_i - x_j\|}_{\text{Distance between points in original 'd' dimensions}} \right)^2$$

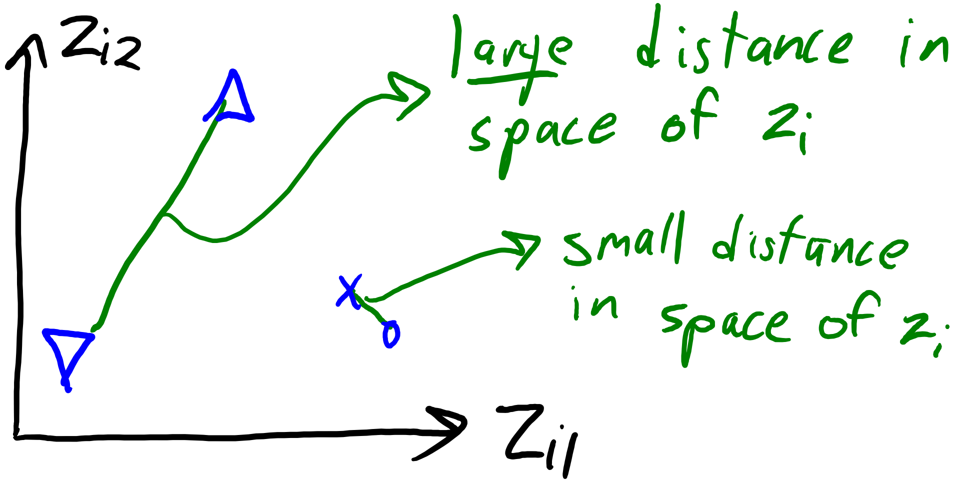
"Try to make scatterplot distances match high-dimensional distance"

Multi-Dimensional Scaling

- Multi-dimensional scaling (MDS):
  - Directly optimize the final locations of the z_i values.

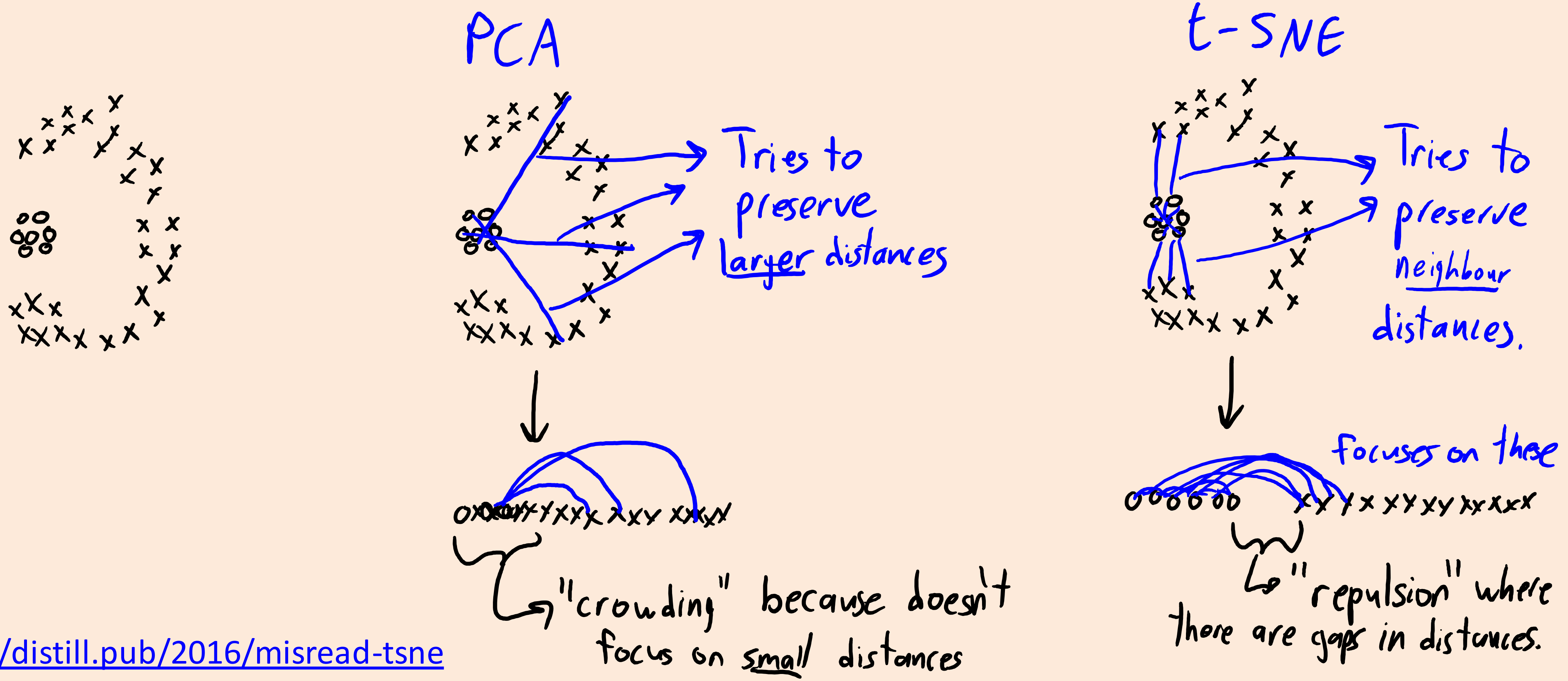
$$f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\|z_i - z_j\| - \|x_i - x_j\| \right)^2$$





t-Distributed Stochastic Neighbour Embedding

- One key idea in t-SNE:
  - Focus on distance to "neighbours"(allow large variance in other distances)



Good demo: https://distill.pub/2016/misread-tsne

Summary

- Recommender systems try to recommend products.
- Collaborative filtering tries to fill in missing values in a matrix.
  - Matrix factorization is a common approach.
- Multi-dimensional scaling is a non-parametric latent-factor model.
  - Big space of variants that we didn’t have time to go into.

- Next time: the long-awaited start of deep learning.

bonus!

Digression: “Whitening”

- With image data, features will be very redundant.
  - Neighbouring pixels tend to have similar values.
- A standard transformation in these settings is “whitening”:
  - Rotate the data so features are uncorrelated.
  - Re-scale the rotated features so they have a variance of 1.

- Using SVD approach to PCA, we can do this with:
  - Get ‘W’ from SVD (usually with k=d).
  - $Z = XW^T$ (rotate to give uncorrelated features).
  - Divide columns of ‘Z’ by corresponding singular values (unit variance).

- Details/discussion here.

Motivation for Topic Models

bonus!

- Want a model of the "factors" making up documents.
  - Instead of latent-factor models, they're called topic models.
  - The canonical topic model is latent Dirichlet allocation (LDA).

Suppose you have the following set of sentences:

- I like to eat broccoli and bananas.
- I ate a banana and spinach smoothie for breakfast.
- Chinchillas and kittens are cute.
- My sister adopted a kitten yesterday.
- Look at this cute hamster munching on a piece of broccoli.

What is latent Dirichlet allocation? It's a way of automatically discovering **topics** that these sentences contain. For example, given these sentences and asked for 2 topics, LDA might produce something like

- **Sentences 1 and 2**: 100% Topic A
- **Sentences 3 and 4**: 100% Topic B
- **Sentence 5**: 60% Topic A, 40% Topic B
- **Topic A**: 30% broccoli, 15% bananas, 10% breakfast, 10% munching, … (at which point, you could interpret topic A to be about food)
- **Topic B**: 20% chinchillas, 20% kittens, 20% cute, 15% hamster, … (at which point, you could interpret topic B to be about cute animals)

  - "Topics" could be useful for things like searching for relevant documents.

http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/

Term Frequency – Inverse Document Frequency

bonus!

- In information retrieval, classic word importance measure is TF-IDF.
- First part is the term frequency tf(t,d) of term 't' for document 'd'.
  - Number of times "word" 't' occurs in document 'd', divided by total words.
  - E.g., 7% of words in document 'd' are "the" and 2% of the words are "Lebron".
- Second part is document frequency df(t,D).
  - Compute number of documents that have 't' at least once.
  - E.g., 100% of documents contain "the" and 0.01% have "LeBron".
- TF-IDF is tf(t,d)*log(1/df(t,D)).

bonus!

Term Frequency – Inverse Document Frequency

- The TF-IDF statistic is tf(t,d)*log(1/df(t,D)).
  - It's high if word 't' happens often in document 'd', but isn't common.
  - E.g., seeing "LeBron" a lot it tells you something about "topic" of article.
  - E.g., seeing "the" a lot tells you nothing.

- There are *many* variations on this statistic.
  - E.g., avoiding dividing by zero and all types of "frequencies".

- Summarizing 'n' documents into a matrix X:
  - Each row corresponds to a document.
  - Each column gives the TF-IDF value of a particular word in the document.

bonus!

Latent Semantic Indexing

- TF-IDF features are very redundant.
  - Consider TF-IDFs of "LeBron", "Durant", "Harden", and "Kobe".
  - High values of these typically just indicate topic of "basketball".

- We can probably compress this information quite a bit.

- Latent Semantic Indexing/Analysis:
  - Run latent-factor model (like PCA or NMF) on TF-IDF matrix X.
  - Treat the principal components as the "topics".
  - Latent Dirichlet allocation is a variant that avoids weird df(t,D) heuristic.

bonus!

SVDfeature with SGD: the gory details

Objective: $\frac{1}{2}\sum_{(i,j)\in R} (\underbrace{\hat{y}_{ij} - y_{ij}}_{r_{ij}})^2$ with $\hat{y}_{ij} = \beta + \beta_i + \beta_j + w^T x_{ij} + (w^j)^T z_i$

Update based on random (i,j):

$$\underbrace{\begin{aligned}\beta &= \beta - \alpha r_{ij}\\ \beta_i &= \beta_i - \alpha r_{ij}\\ \beta_j &= \beta_j - \alpha r_{ij}\end{aligned}}_{\text{Updates are the same,}}$$

but 'β' is always update while β_i and β_j are only updated for the specific user + product

$w = w - \alpha r_{ij} x_{ij}$ ← Updated every time.

$$\left.\begin{aligned} z_i &= z_i - \alpha r_{ij} w^j \\ w^j &= w^j - \alpha r_{ij} z_i \end{aligned}\right\}$$

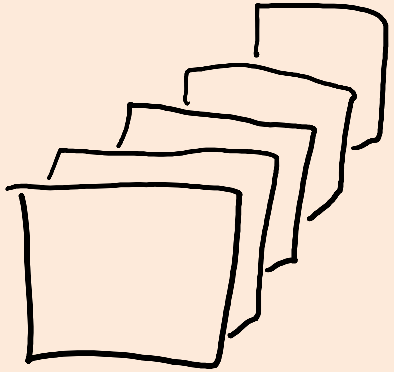
Updated for specific user and product.

(Adding regularization adds an extra term)

bonus!

Tensor Factorization

- Tensors are higher-order generalizations of matrices:

Scalar $\alpha = \underset{1\times 1}{[\ ]}$ Vector $a = [\ ]_{d\times 1}$ Matrix $A = \underset{d\times d}{[\ ]}$ Tensor $A =$  $d\times d\times d$

- Generalization of matrix factorization is tensor factorization:

$$y_{ijm} \approx \sum_{c=1}^{k} w_{jc} z_{ic} v_{mc}$$

- Useful if there are other relevant variables:
  - Instead of ratings based on {user,movie}, ratings based {user,movie,group}.
  - Useful if you have groups of users, or if ratings change over time.

bonus!

Field-Aware Matrix Factorization

- Field-aware factorization machines (FFMs):
  - Matrix factorization with multiple z_i or w_c for each example or part.
  - You choose which z_i or w_c to use based on the value of feature.
- Example from "click through rate" prediction:
  - E.g., predict whether "male" clicks on "nike" advertising on "espn" page.
  - A previous matrix factorization method for the 3 factors used:

$$w_{espn} w_{nike} + w_{espn} w_{male} + w_{nike} w_{male}$$

  - FFMs could use:

$$w_{espn}^{A} w_{nike}^{P} + w_{espn}^{G} w_{male}^{P} + w_{nike}^{G} w_{male}^{A}$$

    - wespnA is the factor we use when multiplying by a an advertiser's latent factor.
    - wespnG is the factor we use when multiplying by a group's latent factor.
- This approach has won some Kaggle competitions (link), and has shown to work well in production systems too (link).

bonus!

Warm-Starting

- We've used data {X,y} to fit a model.
- We now have new training data and want to fit new and old data.

- Do we need to re-fit from scratch?

- This is the warm starting problem.
  - It's easier to warm start some models than others.

bonus!

Easy Case: K-Nearest Neighbours and Counting

- K-nearest neighbours:
  - KNN just stores the training data, so just store the new data.
- Counting-based models:
  - Models that base predictions on frequencies of events.
  - E.g., naïve Bayes.
  - Just update the counts: $p(\text{"vicodin"} \mid \text{"spam"}) = \frac{\text{count of \{vicodin, spam\} in \underline{new and old} data}}{\text{count of "spam" in \underline{new and old} data}}$
  - Decision trees with fixed rules: just update counts at the leaves.

bonus!

Medium Case: L2-Regularized Least Squares

- L2-regularized least squares is obtained from linear algebra:

$$w = (X^T X + \lambda I)^{-1}(X^T y)$$

  - Cost is $O(nd^2 + d^3)$ for 'n' training examples and 'd' features.
- Given one new point, we need to compute:
  - $X^T y$ with one row added, which costs $O(d)$.
  - Old $X^T X$ plus $x_i x_i^T$, which costs $O(d^2)$.
  - Solution of linear system, which costs $O(d^3)$.
  - So cost of adding 't' new data point is $O(td^3)$.
- With "matrix factorization updates", can reduce this to $O(td^2)$.
  - Cheaper than computing from scratch, particularly for large d.

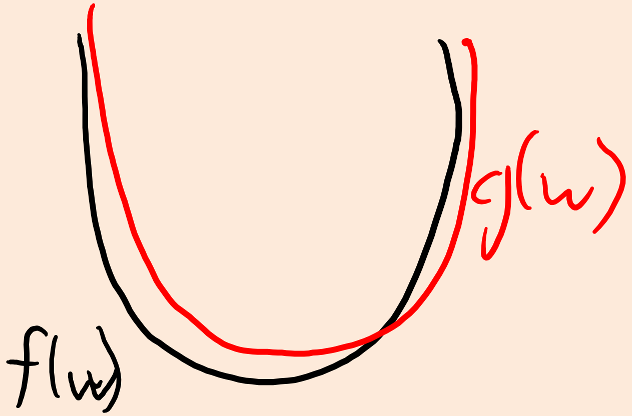
bonus!

Medium Case: Logistic Regression

- We fit logistic regression by gradient descent on a convex function.
- With new data, convex function f(w) changes to new function g(w).

$$f(w) = \sum_{i=1}^{n} f_i(w) \qquad\qquad g(w) = \sum_{i=1}^{n+1} f_i(w)$$

- If we don't have much more data, 'f' and 'g' will be "close".
  - Start gradient descent on 'g' with minimizer of 'f'.
  - You can show that it requires fewer iterations.



bonus!

Hard Cases: Non-Convex/Greedy Models

- For decision trees:
  - "Warm start": continue splitting nodes that haven't already been split.
  - "Cold start": re-fit everything.
- Unlike previous cases, this won't in general give same result as re-fitting:
  - New data points might lead to different splits higher up in the tree.
- Intermediate: usually do warm start but occasionally do a cold start.
- Similar heuristics/conclusions for other non-convex/greedy models:
  - K-means clustering.
  - Matrix factorization (though you can continue PCA algorithms).