Collaborative Filtering

Matrix Factorization Approach

Collaborative filtering algorithms

- Common types:
  - Global effects
  - Nearest neighbor
  - Matrix factorization
  - Restricted Boltzmann machine
  - Clustering
  - Etc.

Optimization

- Optimization is an important part of many machine learning methods.

- The thing we're usually optimizing is the loss function for the model.
  - For a given set of training data **X** and outcomes **y**, we want to find the model parameters **w** that minimize the total loss over all **X, y**.

Loss function

- Suppose target outcomes come from set *Y*
  - Binary classification: *Y* = { 0, 1 }
  - Regression: *Y* = $\Re$ (real numbers)
- A loss function maps decisions to costs:
  - $L(y_i, \hat{y}_i)$ defines the penalty for predicting $\hat{y}_i$ when the true value is y_i.
- Standard choice for classification:
  0/1 loss (same as misclassification error)

$$L_{0/1}(y_i, \hat{y}_i) = \begin{cases} 0 & \text{if } y_i = \hat{y}_i \\ 1 & \text{otherwise} \end{cases}$$

- Standard choice for regression:
  squared loss

$$L(y_i, \hat{y}_i) = (\hat{y}_i - y_i)^2$$

Least squares linear fit to data

- Calculate sum of squared loss (SSL) and determine **w**:

$$\mathrm{SSL} = \sum_{j=1}^{N} (y_j - \sum_{i=0}^{d} w_i \cdot x_i)^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathrm{T}} \cdot (\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\mathbf{y} = \text{vector of all training responses } y_j$$

$$\mathbf{X} = \text{matrix of all training samples } \mathbf{x}_j$$

$$\mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

$$\hat{y}_t = \mathbf{w} \cdot \mathbf{x}_t \qquad \text{for test sample } \mathbf{x}_t$$

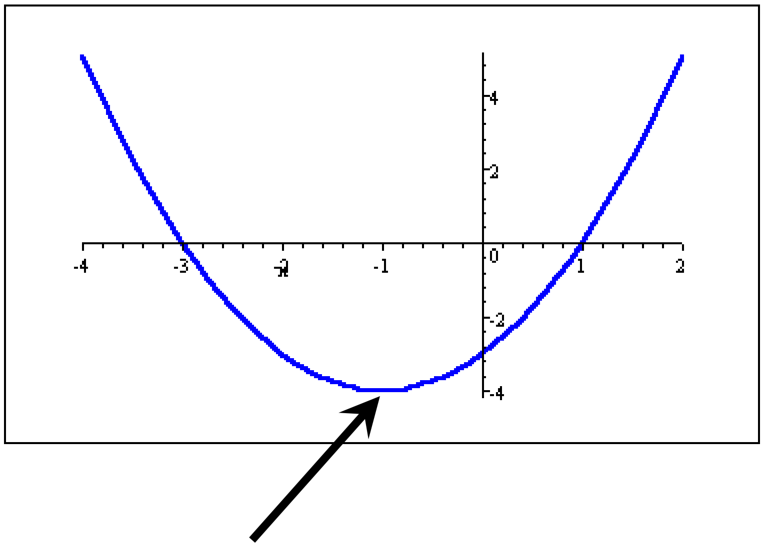
- Can prove that this method of determining **w** minimizes SSL.

Optimization

- Simplest example - quadratic function in 1 variable:

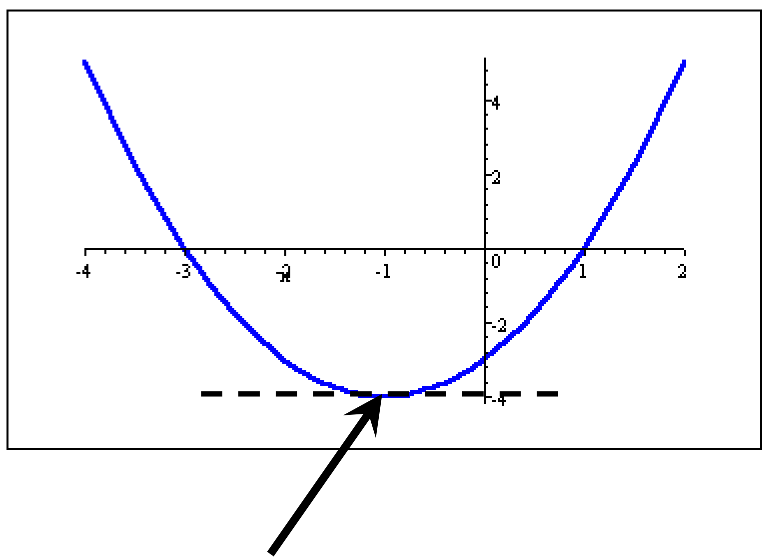
$$f(\ x\ ) = x^2 + 2x - 3$$

- Want to find value of x where $f(\ x\ )$ is minimum



Optimization

- This example is simple enough we can find minimum directly
  - Minimum occurs where slope of curve is 0
  - First derivative of function = slope of curve
  - So set first derivative to 0, solve for *x*



Optimization

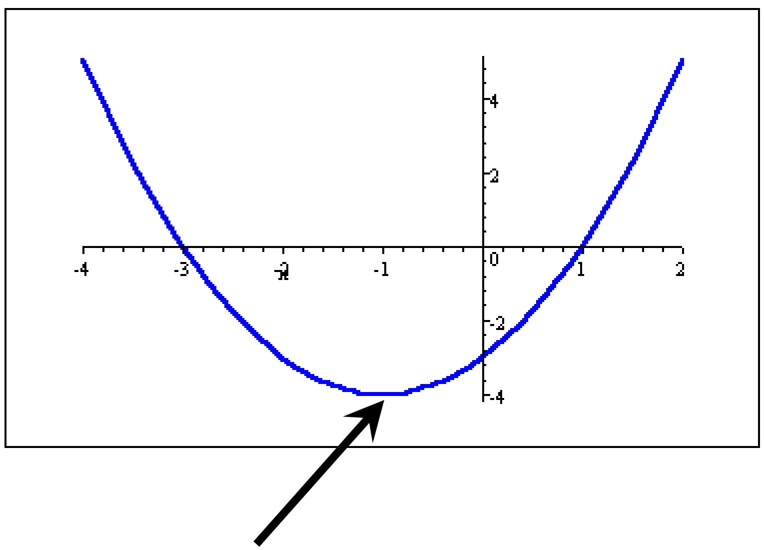
$$f(x) = x^2 + 2x - 3$$

$$f(x) / dx = 2x + 2$$

$$2x + 2 = 0$$

$$x = -1$$

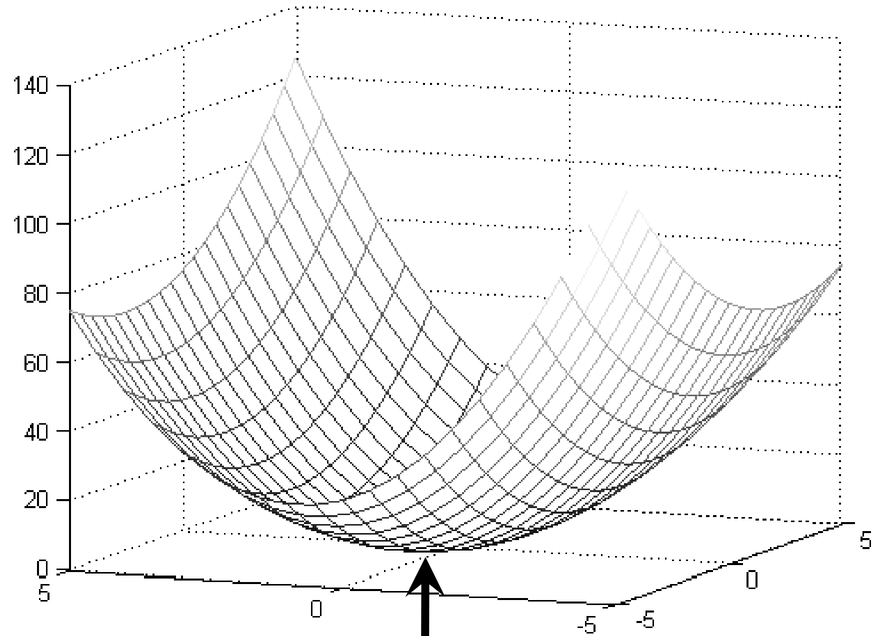
is value of x where $f(x)$ is minimum



Optimization

- Another example - quadratic function in 2 variables:

$$f(\mathbf{x}) = f(x_1, x_2) = x_1^2 + x_1 x_2 + 3x_2^2$$



- $f(\mathbf{x})$ is minimum where gradient of $f(\mathbf{x})$ is zero in all directions

Optimization

- Gradient is a vector
  - Each element is the slope of function along direction of one of variables
  - Each element is the partial derivative of function with respect to one of variables

$$\nabla f(\mathbf{x}) = \nabla f(x_1, x_2, \ldots, x_d) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \frac{\partial f(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_d} \end{bmatrix}$$

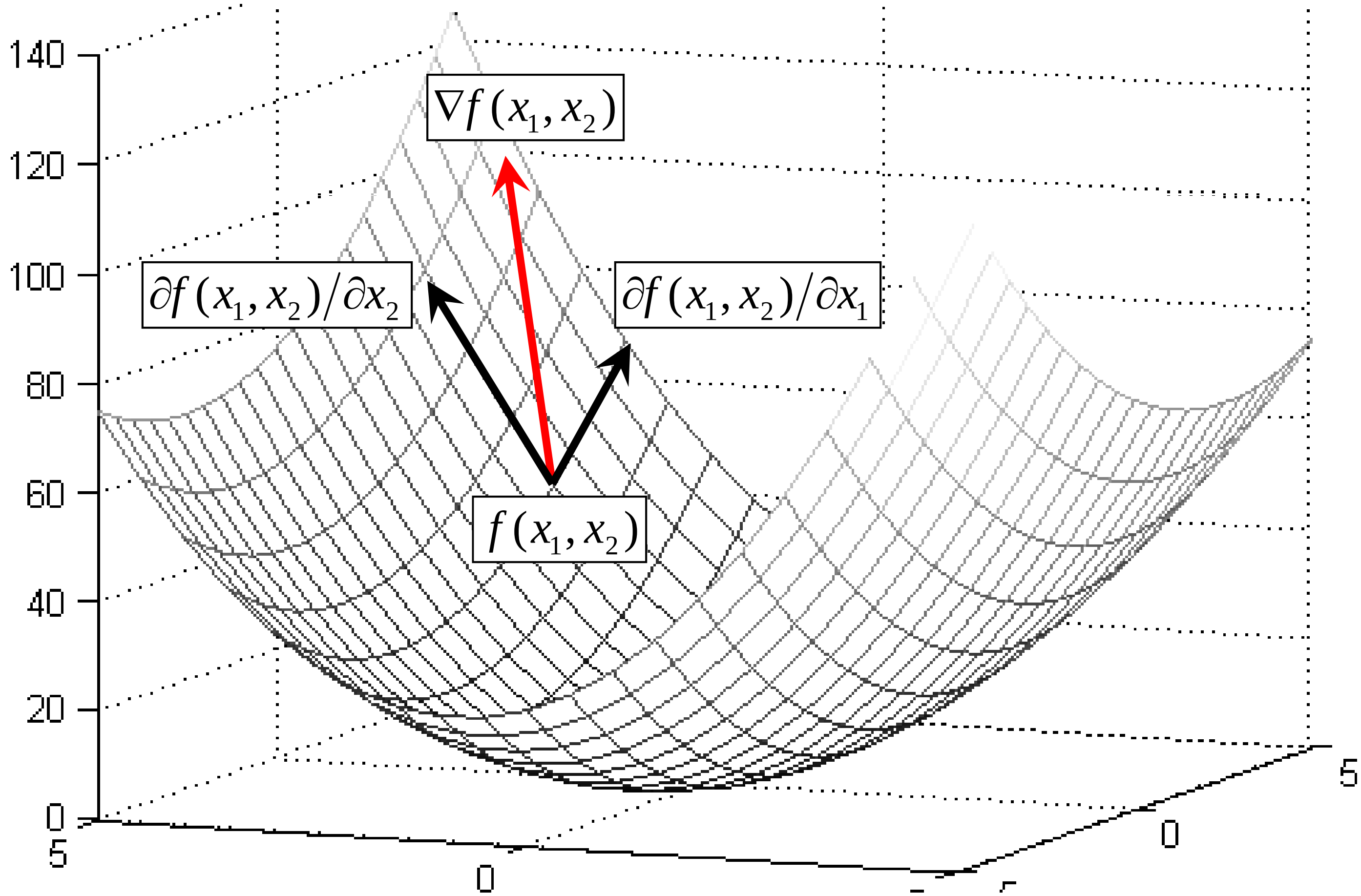
  - Example:

$$f(\mathbf{x}) = f(x_1, x_2) = x_1^2 + x_1 x_2 + 3x_2^2$$

$$\nabla f(\mathbf{x}) = \nabla f(x_1, x_2) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \frac{\partial f(\mathbf{x})}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 + x_2 & x_1 + 6x_2 \end{bmatrix}$$

Optimization

- Gradient vector points in direction of steepest ascent of function



Optimization

- This two-variable example is still simple enough that we can find minimum directly

$$f(x_1, x_2) = x_1^{\,2} + x_1 x_2 + 3x_2^{\,2}$$
$$\nabla f(x_1, x_2) = \begin{bmatrix} 2x_1 + x_2 & x_1 + 6x_2 \end{bmatrix}$$

  - Set both elements of gradient to 0
  - Gives two linear equations in two variables
  - Solve for x_1, x_2

$$2x_1 + x_2 = 0 \qquad\qquad x_1 + 6x_2 = 0$$
$$x_1 = 0 \qquad\qquad x_2 = 0$$

Optimization

- Finding minimum directly by closed form analytical solution often difficult or impossible.
  - Quadratic functions in many variables
    - system of equations for partial derivatives may be ill-conditioned
    - example: linear least squares fit where redundancy among features is high
  - Other convex functions
    - global minimum exists, but there is no closed form solution
    - example: maximum likelihood solution for logistic regression
  - Nonlinear functions
    - partial derivatives are not linear
    - example: $f(x_1, x_2) = x_1(\sin(x_1 x_2)) + x_2^2$
    - example: sum of transfer functions in neural networks

Optimization

- Many approximate methods for finding minima have been developed
  - Gradient descent
  - Newton method
  - Gauss-Newton
  - Levenberg-Marquardt
  - BFGS
  - Conjugate gradient
  - Etc.

Gradient descent optimization

- Simple concept: follow the gradient *downhill*
- Process:
  1. Pick a starting position: $\mathbf{x}^0 = (x_1, x_2, \ldots, x_d)$
  2. Determine the descent direction: $- \nabla f(\mathbf{x}^t)$
  3. Choose a learning rate: η
  4. Update your position: $\mathbf{x}^{t+1} = \mathbf{x}^t - \eta \cdot \nabla f(\mathbf{x}^t)$
  5. Repeat from 2) until stopping criterion is satisfied
- Typical stopping criteria
  - $\nabla f(\mathbf{x}^{t+1}) \sim 0$
  - some validation metric is optimized

Gradient descent optimization

Slides thanks to Alexandre Bayen

(CE 191, Univ. California, Berkeley, 2006)

http://www.ce.berkeley.edu/~bayen/ce191www/lecturenotes/lecture10v01_descent2.pdf

Gradient descent optimization

Example in MATLAB

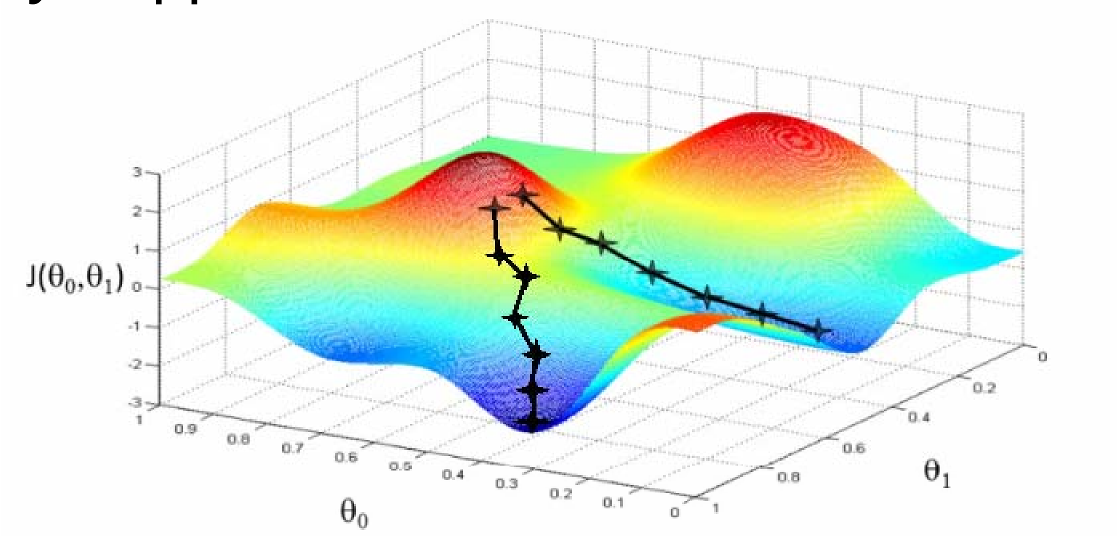
Find minimum of function in two variables:

$$y = x_1^2 + x_1 x_2 + 3x_2^2$$

http://www.youtube.com/watch?v=cY1YGQQbrpQ

Gradient descent optimization

- Problems:
  - Choosing step size
    - too small → convergence is slow and inefficient
    - too large → may not converge
  - Can get stuck on “flat” areas of function
  - Easily trapped in local minima



Stochastic gradient descent

Stochastic (definition):

1. involving a random variable
2. involving chance or probability; probabilistic

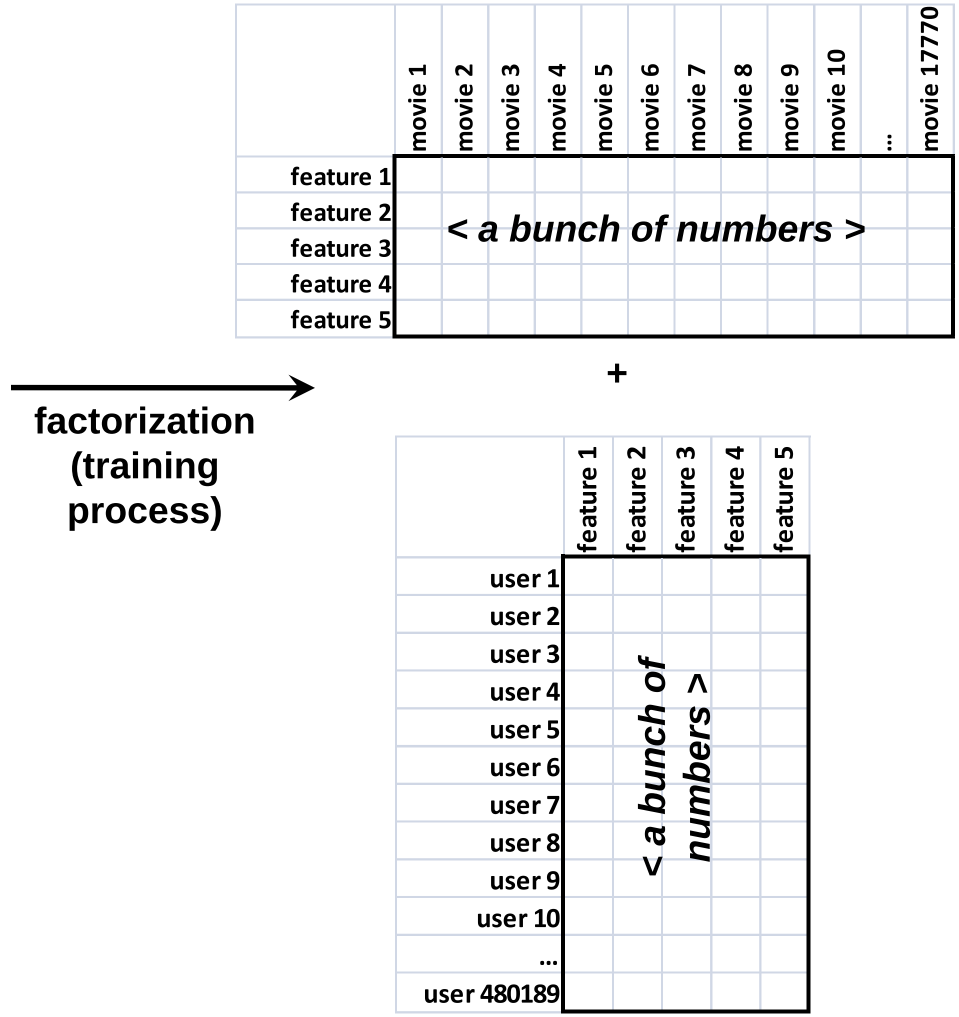
Stochastic gradient descent

- Application to training a machine learning model:
  1. Choose one sample from training set
  2. Calculate loss function for that single sample
  3. Calculate gradient from loss function
  4. Update model parameters a single step based on gradient and learning rate
  5. Repeat from 1) until stopping criterion is satisfied
- Typically entire training set is processed multiple times before stopping.
- Order in which samples are processed can be fixed or random.

Matrix factorization in action

	movie 1	movie 2	movie 3	movie 4	movie 5	movie 6	movie 7	movie 8	movie 9	movie 10	…	movie 17770
user 1			1		2							3
user 2		2		3	3			4				
user 3							5	3		4		
user 4	2				3			2				2
user 5		4				5			3			4
user 6			2									
user 7			2					4	2	3		
user 8	3	4				4						
user 9									3			
user 10			1		2							2
…												
user 480189		4			3			3				

training data



Matrix factorization in action

	movie 1	movie 2	movie 3	movie 4	movie 5	movie 6	movie 7	movie 8	movie 9	movie 10	…	movie 17770
feature 1												
feature 2												
feature 3												
feature 4												
feature 5												

+

	feature 1	feature 2	feature 3	feature 4	feature 5
user 1					
user 2					
user 3					
user 4					
user 5					
user 6					
user 7					
user 8					
user 9					
user 10					
…					
user 480189					

multiply and add features (dot product) for desired < *user*, *movie* > prediction

	movie 1	movie 2	movie 3	movie 4	movie 5	movie 6	movie 7	movie 8	movie 9	movie 10	…	movie 17770
user 1			1		2							3
user 2		2		3	3			4				
user 3							5	3		4		
user 4	2				3			2				2
user 5		4				5			3			4
user 6			2									
user 7			2					4	2	3		
user 8	3	4				4	?					
user 9									3			
user 10			1		2							2
…												
user 480189		4			3			3				

Matrix factorization

- Notation
  - Number of users = I
  - Number of items = J
  - Number of factors per user / item = F
  - User of interest = i
  - Item of interest = j
  - Factor index = f

- User matrix $\boldsymbol{U}$ dimensions = I x F
- Item matrix $\boldsymbol{V}$ dimensions = J x F

Matrix factorization

- Prediction $\hat{r}_{ij}$ for < *user*, *item* > pair *i*, *j* :

$$\hat{r}_{ij} = \sum_{f=1}^{F} U_{if} \cdot V_{jf}$$

- Loss for prediction where true rating is r_{ij}:

$$L(r_{ij}, \hat{r}_{ij}) = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{f=1}^{F} U_{if} \cdot V_{jf})^2$$

  – Using squared loss; other loss functions possible
  – Loss function contains *F* model variables from $\boldsymbol{U}$, and *F* model variables from $\boldsymbol{V}$

Matrix factorization

- Gradient of loss function for sample $< i, j >$:

$$\frac{\partial L(r_{ij}, \hat{r}_{ij})}{\partial U_{if}} = \frac{\partial (r_{ij} - \sum_{f=1}^{F} U_{if} \cdot V_{jf})^2}{\partial U_{if}} = -2(r_{ij} - \sum_{f=1}^{F} U_{if} \cdot V_{jf})V_{jf}$$

$$\frac{\partial L(r_{ij}, \hat{r}_{ij})}{\partial V_{jf}} = \frac{\partial (r_{ij} - \sum_{f=1}^{F} U_{if} \cdot V_{jf})^2}{\partial V_{jf}} = -2(r_{ij} - \sum_{f=1}^{F} U_{if} \cdot V_{jf})U_{if}$$

  – for $f = 1$ to F

Matrix factorization

- Let's simplify the notation:

$$\text{let } e = r_{ij} - \sum_{f=1}^{F} U_{if} \cdot V_{jf} \qquad \text{(the prediction error)}$$

$$\frac{\partial L(r_{ij}, \hat{r}_{ij})}{\partial U_{if}} = \frac{\partial e^2}{\partial U_{if}} = -2eV_{jf}$$

$$\frac{\partial L(r_{ij}, \hat{r}_{ij})}{\partial V_{jf}} = \frac{\partial e^2}{\partial V_{jf}} = -2eU_{if}$$

  – for f = 1 to F

Matrix factorization

- Set learning rate = η
- Then the factor matrix updates for sample $< i, j >$ are:

$$U_{if} = U_{if} + 2\eta e V_{jf}$$
$$V_{jf} = V_{jf} + 2\eta e U_{if}$$

  – for f = 1 to F

Matrix factorization

SGD for training a matrix factorization:

1. Decide on F = dimension of factors
2. Initialize factor matrices with small random values
3. Choose one sample from training set
4. Calculate loss function for that single sample
5. Calculate gradient from loss function
6. Update $2 \cdot F$ model parameters a single step using gradient and learning rate
7. Repeat from 3) until stopping criterion is satisfied

Matrix factorization

- Must use some form of regularization (usually L_2):

$$L(r_{ij}, \hat{r}_{ij}) = (r_{ij} - \sum_{f=1}^{F} U_{if} \cdot V_{jf})^2 + \lambda \sum_{f=1}^{F} U_{if}{}^2 + \lambda \sum_{f=1}^{F} V_{jf}{}^2$$

- Update rules become:

$$U_{if} = U_{if} + 2\eta(eV_{jf} - \lambda U_{if})$$
$$V_{jf} = V_{jf} + 2\eta(eU_{if} - \lambda V_{jf})$$

  – for f = 1 to F

Stochastic gradient descent

- Random thoughts …
  - Samples can be processed in small batches instead of one at a time $\rightarrow$ batch gradient descent
  - We'll see stochastic / batch gradient descent again when we learn about neural networks (as back-propagation)