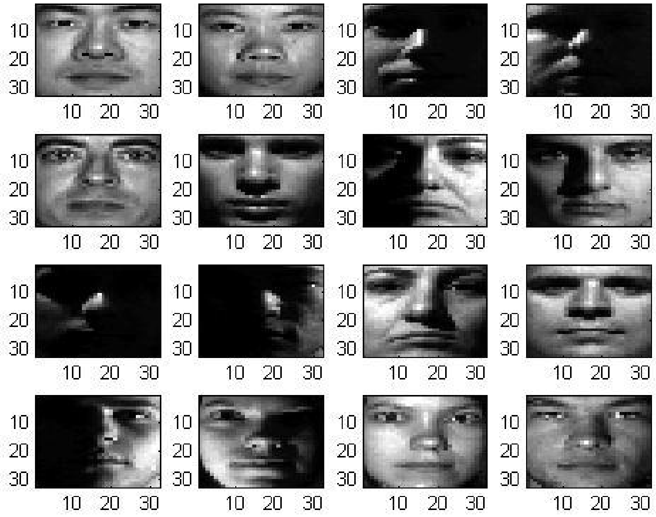
CPSC 340
Assignment 5 Tutorial

Questions 1 to 2.2

Question 1 - Sparse Latent Features

- *example_faces.m* shows an example of using PCA on images
- generates 5 plots:

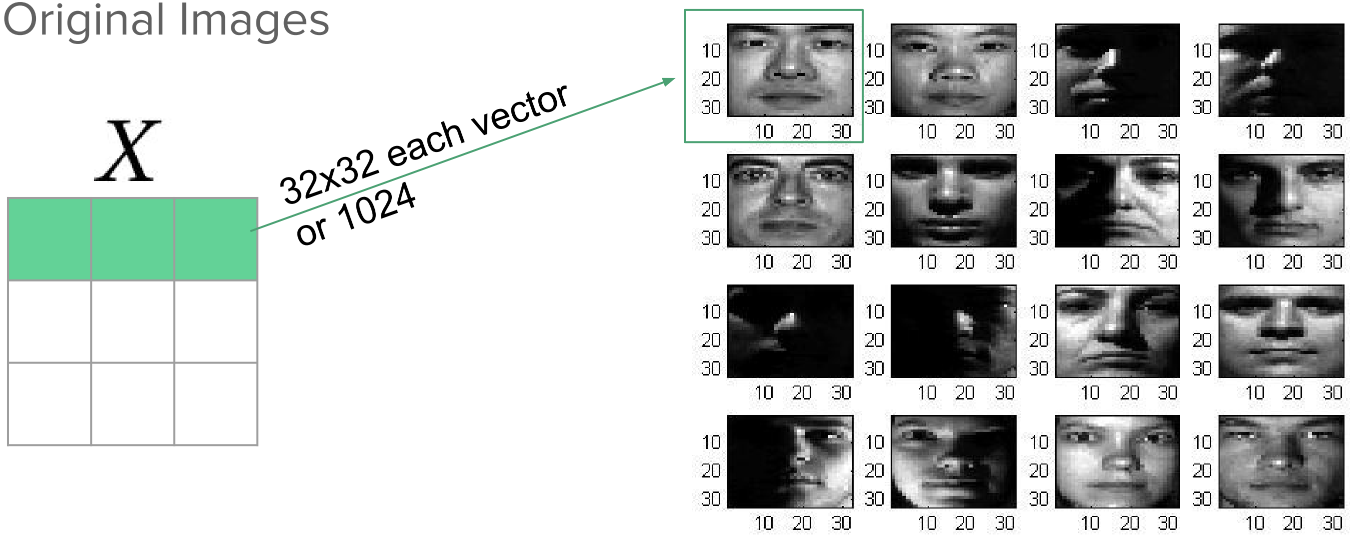
Figure 1 - Original Images



Question 1 - Sparse Latent Features

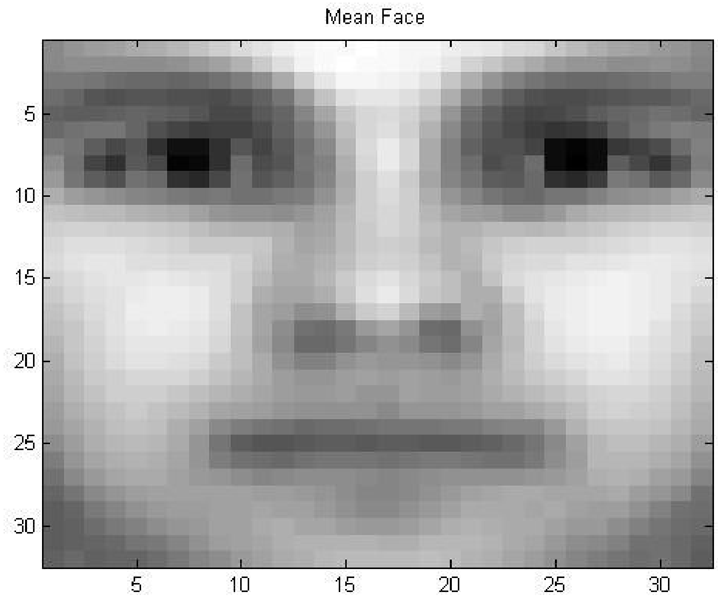
- *example_faces.m* shows an example of using PCA on images
- generates 5 plots:

Figure 1 - Original Images



Question 1 - Sparse Latent Features

- *example_faces.m* - Figure 2: $\boxed{\textit{mu = mean(X);}}$

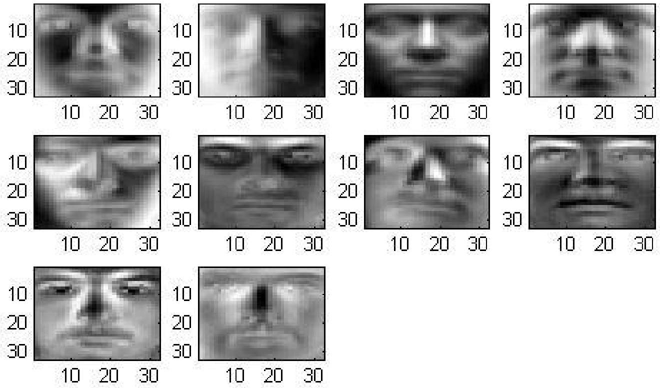


Question 1 - Sparse Latent Features

- *example_faces.m* - Figure 3: $SVD(X) = U, \Sigma, V^T$

$$W = V[1:, 1:k]$$

$$\frac{1}{2}||X - ZW||_2^2$$



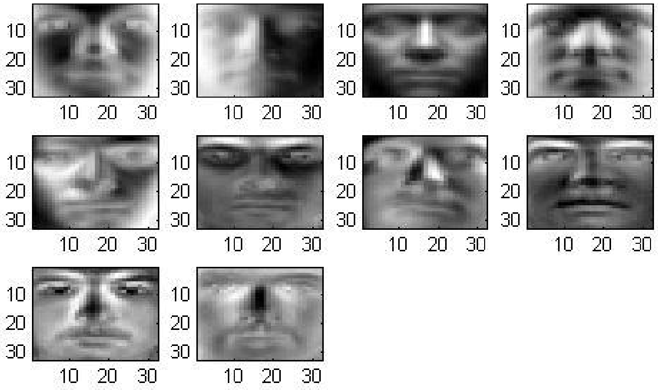
Eigenvectors (Eigenfaces)

Question 1 - Sparse Latent Features

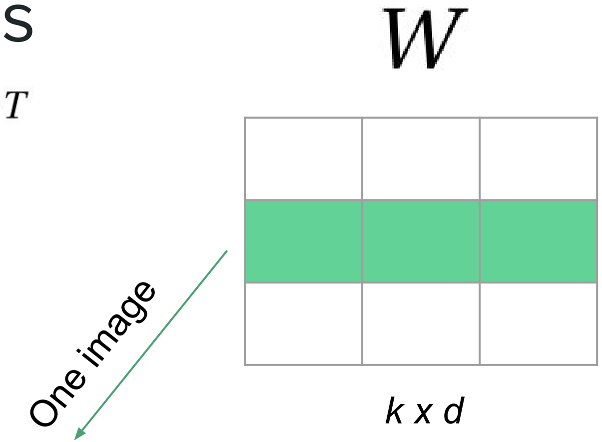
- *example_faces.m* - Figure 3: $SVD(X) = U, \Sigma, V^T$

$$W = V[1:, 1:k]$$

$$\frac{1}{2}||X - Z\boxed{W}||_2^2$$



Eigenvectors (Eigenfaces)



Each eigenvector is 32x32

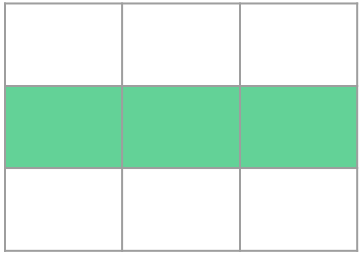
Question 1 - Sparse Latent Features

- *example_faces.m* - Figure 3: $SVD(X) = U, \Sigma, V^T$

$$W = V[1:,1:k]$$

$$\frac{1}{2}||X - Z\boxed{W}||_2^2$$

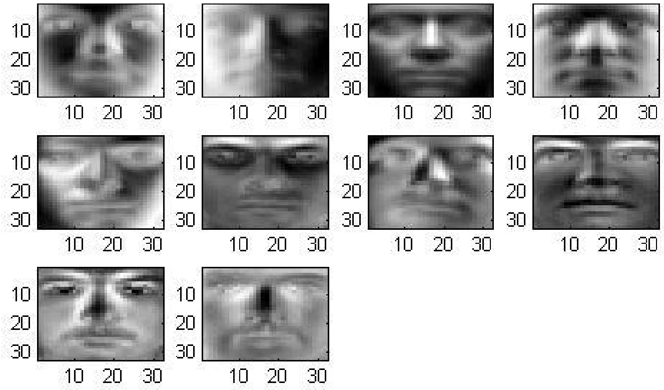
W



k x d

One image

Change the value of “k” to get more eigenvectors



Eigenvectors (Eigenfaces)

Each eigenvector is 32x32

Question 1 - Sparse Latent Features

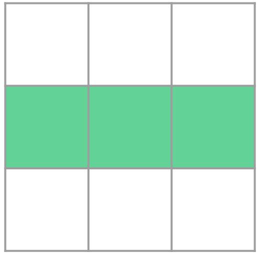
- *example_faces.m* - Figure 4: $\boxed{Z} = \boxed{X}\boxed{W}^T$

Compressed data

Original data

Eigenvectors of $X^T X$

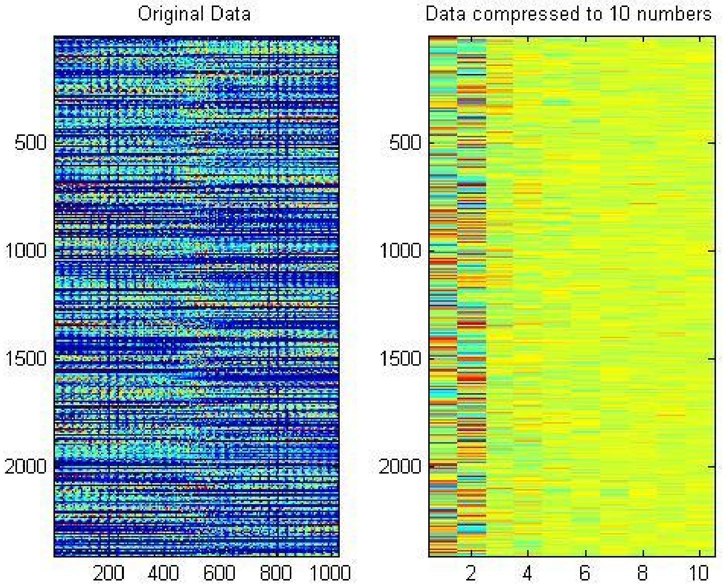
W



k x d

X

n x d



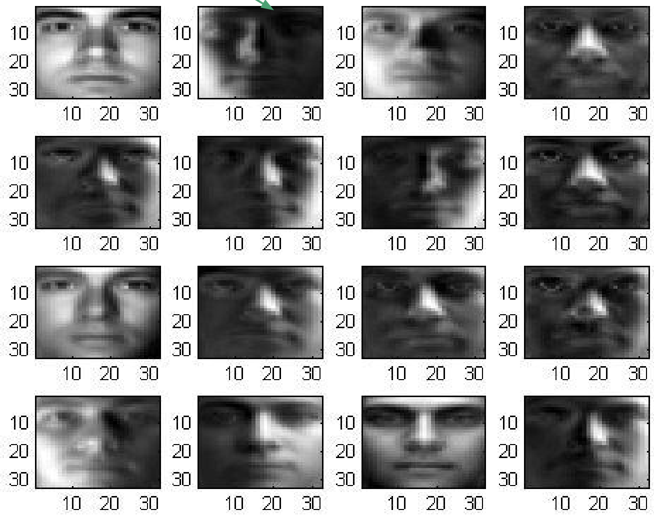
Z

n x k

Question 1 - Sparse Latent Features

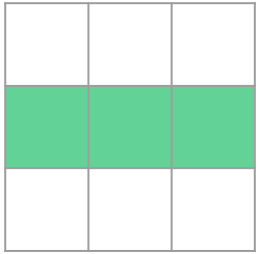
- *example_faces.m* - Figure 5: $\frac{1}{2}||X - ZW||_2^2$

$$\hat{X} = ZW$$



Reconstructed *X*

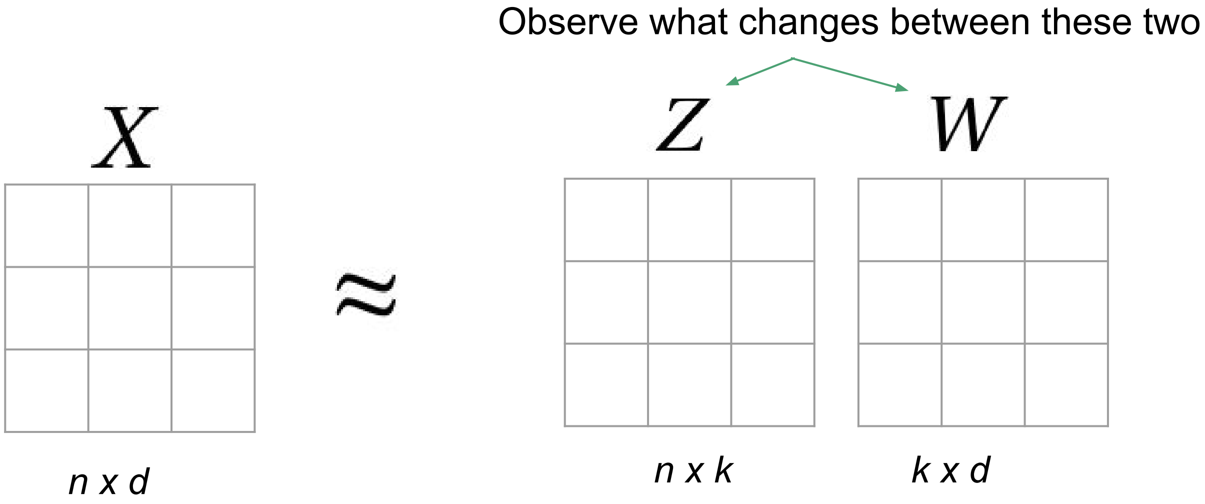
W



k x d

Question 1.1 - Sparse Latent Features

If you re-run the script, you may get different principal components, even though all that changes between runs is the order of the training examples. What is the specific difference between the principal components that are obtained between different runs of the algorithm?



Question 1.2 - Non-Negative Matrix Factorization

non-negative constraints on W. Using *dimRedPCA_alternate* as a template, write a function *dimRedNMF* that implements the non-negative matrix factorization (NMF) model. Hand in your code and hand in a plot of the latent factors (Figure 3) obtained when $k = 100$.

NMF is for optimizing ZW in $\frac{1}{2}||X - ZW||_2^2$
such that Z and W have non-zero terms

```
function [model] = dimRedPCA_alternate(X,k)

[n,d] = size(X);

% Subtract mean
mu = mean(X);
X = X - repmat(mu,[n 1]);

% Initialize W and Z
W = randn(k,d);
Z = randn(n,k);

f = (1/2)*sum(sum((X-Z*W).^2));
for iter = 1:50
    fOld = f;

    % Update Z
    Z(:) = findMin(@funObjZ,Z(:),10,0,X,W);

    % Update W
    W(:) = findMin(@funObjW,W(:),10,0,X,Z);

    f = (1/2)*sum(sum((X-Z*W).^2));
    fprintf('Iteration %d, loss = %.5e\n',iter,f);

    if fOld - f < 1
        break;
    end
end

model.mu = mu;
model.W = W;
model.compress = @compress;
model.expand = @expand;
end
```

Question 1.2 - use NMF

a. Optimization stage

Initialize without negative values

$$\frac{1}{2}||X - ZW||_2^2$$

`findMin` uses gradient descent (no constraints)

Question 1.2 - use NMF

a. Optimization stage

```
function [model] = dimRedPCA_alternate(X,k)

[n,d] = size(X);

% Subtract mean
mu = mean(X);
X = X - repmat(mu,[n 1]);

% Initialize W and Z
W = randn(k,d);
Z = randn(n,k);

f = (1/2)*sum(sum((X-Z*W).^2));
for iter = 1:50
    fOld = f;

    % Update Z
    Z(:) = findMin(@funObjZ,Z(:),10,0,X,W);

    % Update W
    W(:) = findMin(@funObjW,W(:),10,0,X,Z);

    f = (1/2)*sum(sum((X-Z*W).^2));
    fprintf('Iteration %d, loss = %.5e\n',iter,f);

    if fOld - f < 1
        break;
    end
end

model.mu = mu;
model.W = W;
model.compress = @compress;
model.expand = @expand;
end
```

Initialize without negative values

$$\frac{1}{2}||X - ZW||_2^2$$

`findMin` uses gradient descent (no constraints)

Use gradient descent that enforces non-negative parameters (**findMinNN**) instead!

```
function [Z] = compress(model,X)
[t,d] = size(X);
mu = model.mu;
W = model.W;

X = X - repmat(mu,[t 1]);
% We didn't enforce that W was ort
Z = X*W'*inv(W*W');
end
```

Question 1.2 - use NMF

b. Compress function

Computes *Z* with *W* fixed.

Uses least squares - we don't want that!

Use gradient descent that enforces non-negative parameters (**findMinNN)** instead!

- NMF results in sparse matrices for *Z* and *W* since negative values become zero
- However, the compression ratio is poor - the non-negative constraint strongly limits the model power

```
function [model] = dimRedPCA_alternate(X,k)

[n,d] = size(X);

% Subtract mean
mu = mean(X);
X = X - repmat(mu,[n 1]);

% Initialize W and Z
W = randn(k,d);
Z = randn(n,k);

f = (1/2)*sum(sum((X-Z*W).^2));
for iter = 1:50
    fOld = f;

    % Update Z
    Z(:) = findMin(@funObjZ,Z(:),10,0,X,W);

    % Update W
    W(:) = findMin(@funObjW,W(:),10,0,X,Z);

    f = (1/2)*sum(sum((X-Z*W).^2));
    fprintf('Iteration %d, loss = %.5e\n',iter,f);

    if fOld - f < 1
        break;
    end
end

model.mu = mu;
model.W = W;
model.compress = @compress;
model.expand = @expand;
end
```

Question 1.3 -

Use L1 regularization

a. Optimization stage

We can have negative values!

$$\frac{1}{2}||X - ZW||_2^2$$

`findMin` uses gradient descent (no constraints)

Use L1-regularized gradient descent (**findMinL1)** instead!

Question 1.3 - use L1 Regularization

b. Compress function

Computes *Z* with *W* fixed.

```
function [Z] = compress(model,X)
[t,d] = size(X);
mu = model.mu;
W = model.W;

X = X - repmat(mu,[t 1]);
% We didn't enforce that W was ort
Z = X*W'*inv(W*W');
end
```

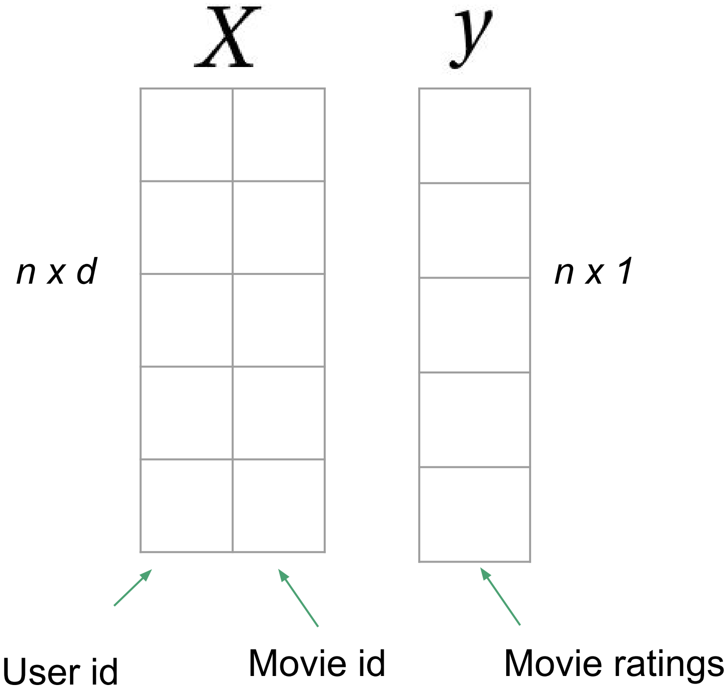
Uses least squares - no constraints!

Use L1-regularized gradient descent (**findMinL1)** instead!

- L1 Regularization results in sparse matrices for *Z* and *W*
- The compression ratio is better for using L1 than for using NMF

Question 2 - Recommender Systems

If you run the function example_movies, it will load a dataset consisting of movie ratings for different users. The vector y contains the ratings, the first column of X contains the user numbers, and the second column of X contains the movie numbers. The script runs several simple baseline methods, and reports their performance on the validation set.

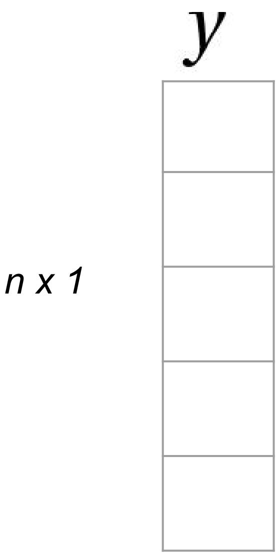


- No features being represented
- But we can extract latent features that represent the relationships between users, movies, and ratings

Question 2.1 - Latent-Factor Model

We have no features for the user/movies, we must predict the labels based on other labels (collaborative filtering). One way to improve on these methods is with a latent-factor model. Consider a model of the form

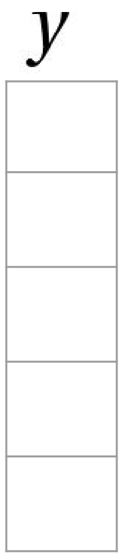
$$y_{um} = b_u + b_m + w_m^T z_u,$$



Question 2.1 - Latent-Factor Model

We have no features for the user/movies, we must predict the labels based on other labels (collaborative filtering). One way to improve on these methods is with a latent-factor model. Consider a model of the form

$$y_{um} = \boxed{b_u} + \boxed{b_m} + \boxed{w_m^T z_u},$$



n x 1

Latent features (represents hidden relationships)
We can extract them through optimization

Bias term, variable for movie '*m*'

Bias term, variable for user '*u*'

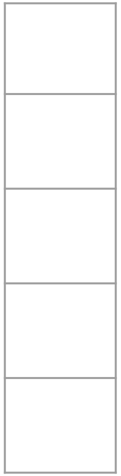
Question 2.1 - Latent-Factor Model

We have no features for the user/movies, we must predict the labels based on other labels (collaborative filtering). One way to improve on these methods is with a latent-factor model. Consider a model of the form

$$y_{um} = \boxed{b_u} + \boxed{b_m} + \boxed{w_m^T z_u},$$

We want this equality to hold!
Use optimization!

y



n x 1

Latent features (represents hidden relationships)
We can extract them through optimization

Bias term, variable for movie '*m*'

Bias term, variable for user '*u*'

Question 2.1 - Latent-Factor Model

Consider training this based on the squared loss function, which means that our error for a particular user u and movie m is given by

$$f(b_u, b_m, w_m, z_u) = \frac{1}{2}(y_{um} - (b_u + b_m + w_m^T z_u))^2.$$

Minimize this objective function

- To optimize this function we can use gradient descent, which involves computing the partial derivatives w.r.t to the **unknown variables**.

Question 2.1 - Latent-Factor Model

$$f(b_u, b_m, w_m, z_u) = \frac{1}{2}(y_{um} - (b_u + b_m + w_m^T z_u))^2.$$

Using the notation $r_{um} = (y_{um} - (b_u + b_m + w_m^T z_u))$, derive the partial derivative of this expression with respect to (i) b_u, (ii) b_m, (iii) $(w_m)_i$ for a particular element i of w_m, and (iv) $(z_u)_i$ for a particular element i of z_u.

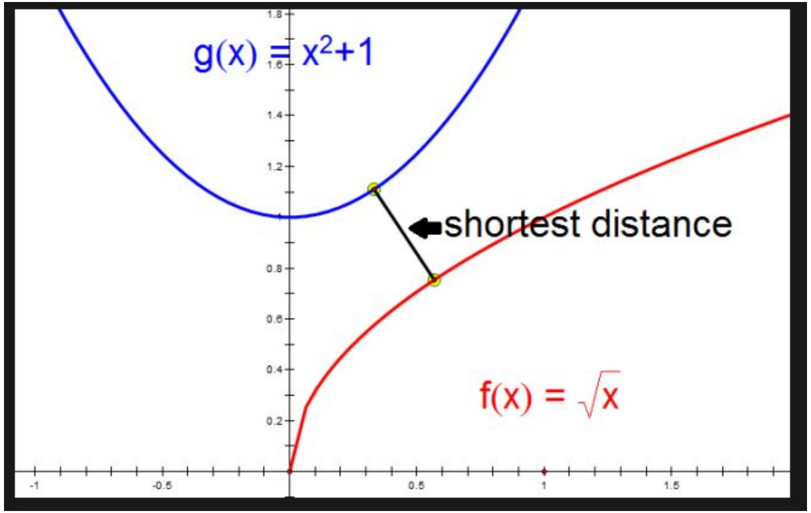
$$\frac{\partial f}{\partial b_u} = ?$$

$$\frac{\partial f}{\partial b_m} = ?$$

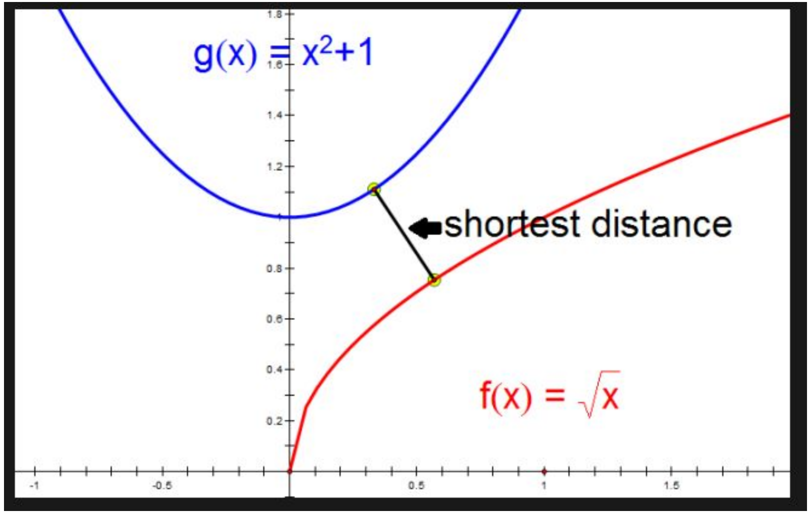
$$\frac{\partial f}{\partial (w_m)_i} = ?$$

$$\frac{\partial f}{\partial (z_u)_i} = ?$$

Question 2.1 - Optimization problem example

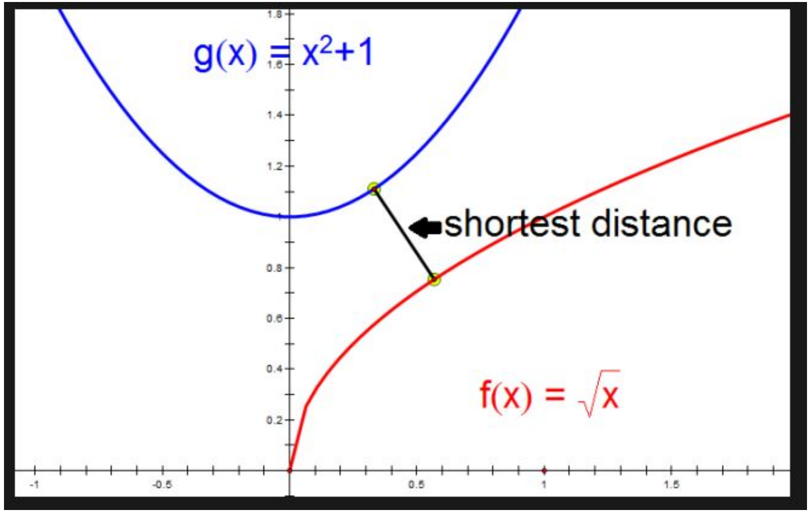


Question 2.1 - Optimization problem example



$$P(x_1, x_2) = \frac{1}{2}(f(x_1) - g(x_2))^2 + \frac{1}{2}(x_1 - x_2)^2$$

Question 2.1 - Optimization problem example



$$P(x_1, x_2) = \frac{1}{2}(f(x_1) - g(x_2))^2 + \frac{1}{2}(x_1 - x_2)^2$$

$$\frac{\partial P}{\partial x_1} = x_1 - x_2 - \frac{1}{\sqrt{x_1}}\left(-\frac{\sqrt{x_1}}{2} + \frac{x_2^2}{2} + \frac{1}{2}\right)$$

$$\frac{\partial P}{\partial x_2} = -x_1 + 2x_2(x_2^2 - \sqrt{x_1} + 1) + x_2$$

Algorithm

1. Initialize random values for x_1, x_2
2. Update x_1, x_2 as follows,

$$x_1 := x_1 - \alpha \frac{\partial P}{\partial x_1}$$

$$x_2 := x_2 - \alpha \frac{\partial P}{\partial x_2}$$

$$\alpha > 0$$

learning rate

Question 2.2 - Gradient descent

- Gradient Descent
  - Uses the complete dataset per iteration
  - Very costly for datasets with over million samples

```
% Optimization
maxIter = 10;
alpha = 1e-4;
for iter = 1:maxIter

    % Compute gradient
    gu = zeros(n,1);
    gm = zeros(d,1);
    gW = zeros(k,d);
    gZ = zeros(n,k);
    for i = 1:nRatings

        % Make prediction for this rating based on current model
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';

        % Add gradient of this prediction to overall gradient
        % (follows from chain rule)
        r = y(i)-yhat;
        gu(u) = gu(u) - r;
        gm(m) = gm(m) - r;
        gW(:,m) = gW(:,m) - r*Z(u,:)';
        gZ(u,:) = gZ(u,:) - r*W(:,m)';
    end

    % Take a small step in the negative gradient directions
    bu = bu - alpha*gu;
    bm = bm - alpha*gm;
    W = W - alpha*gW;
    Z = Z - alpha*gZ;

    % Compute and output function value
    f = 0;
    for i = 1:nRatings
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';
        f = f + (1/2)*(y(i) - yhat)^2;
    end
    fprintf('Iter = %d, f = %e\n',iter,f);
end
```

```
% Optimization
maxIter = 10;
alpha = 1e-4;
for iter = 1:maxIter

    % Compute gradient
    gu = zeros(n,1);
    gm = zeros(d,1);
    gW = zeros(k,d);
    gZ = zeros(n,k);
    for i = 1:nRatings

        % Make prediction for this rating based on current model
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';

        % Add gradient of this prediction to overall gradient
        % (follows from chain rule)
        r = y(i)-yhat;
        gu(u) = gu(u) - r;
        gm(m) = gm(m) - r;
        gW(:,m) = gW(:,m) - r*Z(u,:)';
        gZ(u,:) = gZ(u,:) - r*W(:,m)';
    end

    % Take a small step in the negative gradient directions
    bu = bu - alpha*gu;
    bm = bm - alpha*gm;
    W = W - alpha*gW;
    Z = Z - alpha*gZ;

    % Compute and output function value
    f = 0;
    for i = 1:nRatings
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';
        f = f + (1/2)*(y(i) - yhat)^2;
    end
    fprintf('Iter = %d, f = %e\n',iter,f);
end
```

Question 2.2 - Gradient descent

- Gradient Descent
  - Uses the complete dataset per iteration
  - Very costly for datasets with over million samples

Question 2.2 - Gradient descent

```
% Optimization
maxIter = 10;
alpha = 1e-4;
for iter = 1:maxIter

    % Compute gradient
    gu = zeros(n,1);
    gm = zeros(d,1);
    gW = zeros(k,d);
    gZ = zeros(n,k);
    for i = 1:nRatings

        % Make prediction for this rating based on current model
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';

        % Add gradient of this prediction to overall gradient
        % (follows from chain rule)
        r = y(i)-yhat;
        gu(u) = gu(u) - r;
        gm(m) = gm(m) - r;
        gW(:,m) = gW(:,m) - r*Z(u,:)';
        gZ(u,:) = gZ(u,:) - r*W(:,m)';
    end

    % Take a small step in the negative gradient directions
    bu = bu - alpha*gu;
    bm = bm - alpha*gm;
    W = W - alpha*gW;
    Z = Z - alpha*gZ;

    % Compute and output function value
    f = 0;
    for i = 1:nRatings
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';
        f = f + (1/2)*(y(i) - yhat)^2;
    end
    fprintf('Iter = %d, f = %e\n',iter,f);
end
```

- Gradient Descent
  - Uses the complete dataset per iteration
  - Very costly for datasets with over million samples

```
% Optimization
maxIter = 10;
alpha = 1e-4;
for iter = 1:maxIter

    % Compute gradient
    gu = zeros(n,1);
    gm = zeros(d,1);
    gW = zeros(k,d);
    gZ = zeros(n,k);
    for i = 1:nRatings

        % Make prediction for this rating based on current model
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';

        % Add gradient of this prediction to overall gradient
        % (follows from chain rule)
        r = y(i)-yhat;
        gu(u) = gu(u) - r;
        gm(m) = gm(m) - r;
        gW(:,m) = gW(:,m) - r*Z(u,:)';
        gZ(u,:) = gZ(u,:) - r*W(:,m)';
    end

    % Take a small step in the negative gradient directions
    bu = bu - alpha*gu;
    bm = bm - alpha*gm;
    W = W - alpha*gW;
    Z = Z - alpha*gZ;

    % Compute and output function value
    f = 0;
    for i = 1:nRatings
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';
        f = f + (1/2)*(y(i) - yhat)^2;
    end
    fprintf('Iter = %d, f = %e\n',iter,f);
end
```

Question 2.2 - Gradient descent

- Gradient Descent
  - Uses the complete dataset per iteration
  - Very costly for datasets with over million samples

Accumulates gradients for each sample

Question 2.2 - Gradient descent

- Gradient Descent
  - Uses the complete dataset per iteration
  - Very costly for datasets with over million samples

```
% Optimization
maxIter = 10;
alpha = 1e-4;
for iter = 1:maxIter

    % Compute gradient
    gu = zeros(n,1);
    gm = zeros(d,1);
    gW = zeros(k,d);
    gZ = zeros(n,k);
    for i = 1:nRatings

        % Make prediction for this rating based on current model
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';

        % Add gradient of this prediction to overall gradient
        % (follows from chain rule)
        r = y(i)-yhat;
        gu(u) = gu(u) - r;
        gm(m) = gm(m) - r;
        gW(:,m) = gW(:,m) - r*Z(u,:)';
        gZ(u,:) = gZ(u,:) - r*W(:,m)';
    end

    % Take a small step in the negative gradient directions
    bu = bu - alpha*gu;
    bm = bm - alpha*gm;
    W = W - alpha*gW;
    Z = Z - alpha*gZ;

    % Compute and output function value
    f = 0;
    for i = 1:nRatings
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';
        f = f + (1/2)*(y(i) - yhat)^2;
    end
    fprintf('Iter = %d, f = %e\n',iter,f);
end
```

Accumulates gradients for each sample

Updates variables

Question 2.2 - Gradient descent

```
% Optimization
maxIter = 10;
alpha = 1e-4;
for iter = 1:maxIter

    % Compute gradient
    gu = zeros(n,1);
    gm = zeros(d,1);
    gW = zeros(k,d);
    gZ = zeros(n,k);
    for i = 1:nRatings

        % Make prediction for this rating based on current model
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';

        % Add gradient of this prediction to overall gradient
        % (follows from chain rule)
        r = y(i)-yhat;
        gu(u) = gu(u) - r;
        gm(m) = gm(m) - r;
        gW(:,m) = gW(:,m) - r*Z(u,:)';
        gZ(u,:) = gZ(u,:) - r*W(:,m)';
    end

    % Take a small step in the negative gradient directions
    bu = bu - alpha*gu;
    bm = bm - alpha*gm;
    W = W - alpha*gW;
    Z = Z - alpha*gZ;

    % Compute and output function value
    f = 0;
    for i = 1:nRatings
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';
        f = f + (1/2)*(y(i) - yhat)^2;
    end
    fprintf('Iter = %d, f = %e\n',iter,f);
end
```

Pick one sample randomly

- Gradient Descent
  - Uses the complete dataset per iteration
  - Very costly for datasets with over million samples
- Use **Stochastic Gradient Descent** instead
  - Uses one random sample at a time to update the variables.

```
% Optimization
maxIter = 10;
alpha = 1e-4;
for iter = 1:maxIter

    % Compute gradient
    gu = zeros(n,1);
    gm = zeros(d,1);
    gW = zeros(k,d);
    gZ = zeros(n,k);
    for i = 1:nRatings

        % Make prediction for this rating based on current model
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';

        % Add gradient of this prediction to overall gradient
        % (follows from chain rule)
        r = y(i)-yhat;
        gu(u) = gu(u) - r;
        gm(m) = gm(m) - r;
        gW(:,m) = gW(:,m) - r*Z(u,:)';
        gZ(u,:) = gZ(u,:) - r*W(:,m)';
    end

    % Take a small step in the negative gradient directions
    bu = bu - alpha*gu;
    bm = bm - alpha*gm;
    W = W - alpha*gW;
    Z = Z - alpha*gZ;

    % Compute and output function value
    f = 0;
    for i = 1:nRatings
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';
        f = f + (1/2)*(y(i) - yhat)^2;
    end
    fprintf('Iter = %d, f = %e\n',iter,f);
end
```

Pick one sample randomly

Question 2.2 - Gradient descent

- Gradient Descent
  - Uses the complete dataset per iteration
  - Very costly for datasets with over million samples
- Use **Stochastic Gradient Descent** instead
  - Uses one random sample at a time to update the variables.

Accumulates gradients for each sample

Question 2.2 - Gradient descent

- Gradient Descent
  - Uses the complete dataset per iteration
  - Very costly for datasets with over million samples
- Use **Stochastic Gradient Descent** instead
  - Uses one random sample at a time to update the variables.

```
% Optimization
maxIter = 10;
alpha = 1e-4;
for iter = 1:maxIter

    % Compute gradient
    gu = zeros(n,1);
    gm = zeros(d,1);
    gW = zeros(k,d);
    gZ = zeros(n,k);
    for i = 1:nRatings

        % Make prediction for this rating based on current model
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';

        % Add gradient of this prediction to overall gradient
        % (follows from chain rule)
        r = y(i)-yhat;
        gu(u) = gu(u) - r;
        gm(m) = gm(m) - r;
        gW(:,m) = gW(:,m) - r*Z(u,:)';
        gZ(u,:) = gZ(u,:) - r*W(:,m)';
    end

    % Take a small step in the negative gradient directions
    bu = bu - alpha*gu;
    bm = bm - alpha*gm;
    W = W - alpha*gW;
    Z = Z - alpha*gZ;

    % Compute and output function value
    f = 0;
    for i = 1:nRatings
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';
        f = f + (1/2)*(y(i) - yhat)^2;
    end
    fprintf('Iter = %d, f = %e\n',iter,f);
end
```

Pick one sample randomly

~~Accumulates gradients for each sample~~

Use the gradient for the chosen sample only!

Question 2.2 - Gradient descent

```
% Optimization
maxIter = 10;
alpha = 1e-4;
for iter = 1:maxIter

    % Compute gradient
    gu = zeros(n,1);
    gm = zeros(d,1);
    gW = zeros(k,d);
    gZ = zeros(n,k);
    for i = 1:nRatings

        % Make prediction for this rating based on current model
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';

        % Add gradient of this prediction to overall gradient
        % (follows from chain rule)
        r = y(i)-yhat;
        gu(u) = gu(u) - r;
        gm(m) = gm(m) - r;
        gW(:,m) = gW(:,m) - r*Z(u,:)';
        gZ(u,:) = gZ(u,:) - r*W(:,m)';
    end

    % Take a small step in the negative gradient directions
    bu = bu - alpha*gu;
    bm = bm - alpha*gm;
    W = W - alpha*gW;
    Z = Z - alpha*gZ;

    % Compute and output function value
    f = 0;
    for i = 1:nRatings
        u = X(i,1);
        m = X(i,2);
        yhat = bu(u) + bm(m) + W(:,m)'*Z(u,:)';
        f = f + (1/2)*(y(i) - yhat)^2;
    end
    fprintf('Iter = %d, f = %e\n',iter,f);
end
```

Pick one sample randomly

- Gradient Descent
  - Uses the complete dataset per iteration
  - Very costly for datasets with over million samples
- Use **Stochastic Gradient Descent** instead
  - Uses one random sample at a time to update the variables.

~~Accumulates gradients for each sample~~

Use the gradient for the chosen sample only!

Insert the update rules inside the loop
We update every time we choose a random sample