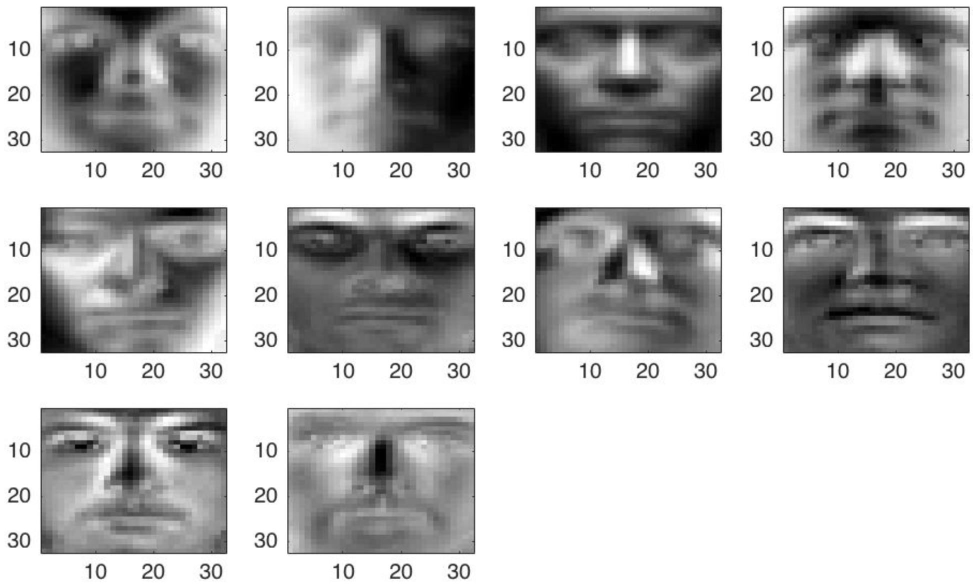
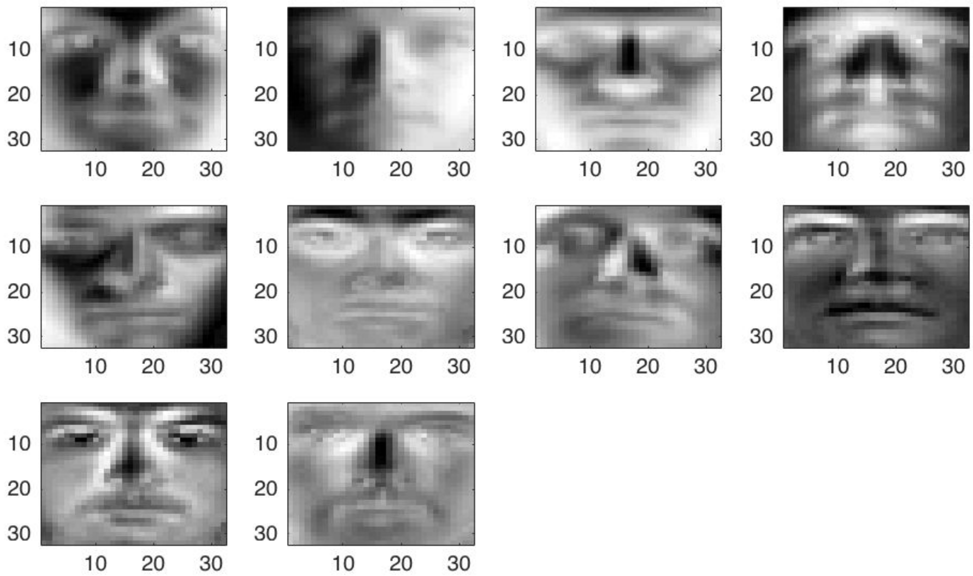
Assignment 5
Student Name: Weining Hu
Student Number: 45606134

1 Sparse Latent-Factor Models
1.1 Uniqueness of Principal Components





After running several times, I found that there are two versions of the PCA after random permutation of all the faces.The specific difference is the light. They have the same figure but different shades. (Like in the opposite way).

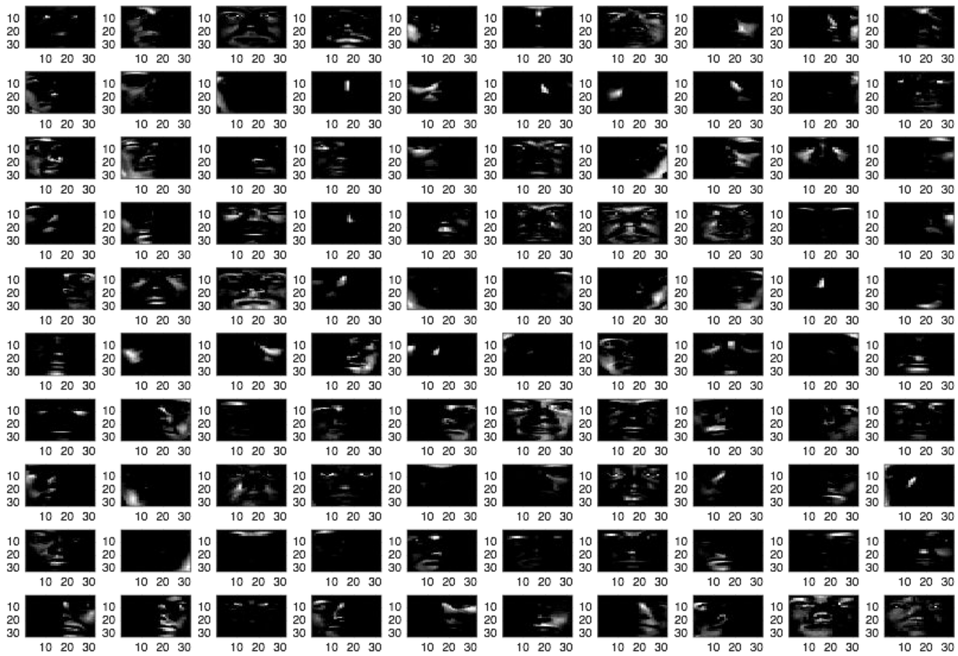
1.2 Non-Negative Matrix Factorization

code:

```
function [model] = dimRedNMF(X,k)
[n,d] = size(X);

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

% Initialize W and Z,set negative values in the matrix to 0
W = randn(k,d);
Z = randn(n,k);

W(W<0) = 0;
Z(Z<0) = 0;

f = (1/2)*sum(sum((X-Z*W).^2));
```

```
for iter = 1:50
    fOld = f;

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

    % Update W
    W(:) = findMinNN(@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.Z = Z;
model.compress = @compress;
model.expand = @expand;
end

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

X = X - repmat(mu,[t 1]);
% With W fixed we minimize Z, but with non-negative constraints
Z(:) = findMinNN(@funObjZ,Z(:),10,0,X,W);
end

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

X = Z*W + repmat(mu,[t 1]);
end

function [f,g] = funObjW(W,X,Z)
% Resize vector of parameters into matrix
d = size(X,2);
```

```
k = size(Z,2);
W = reshape(W,[k d]);

% Compute function and gradient
R = X-Z*W;
f = (1/2)*sum(sum(R.^2));
g = -Z'*R;

% Return a vector
g = g(:);
end

function [f,g] = funObjZ(Z,X,W)
% Resize vector of parameters into matrix
n = size(X,1);
k = size(W,1);
Z = reshape(Z,[n k]);

% Compute function and gradient
R = X-Z*W;
f = (1/2)*sum(sum(R.^2));
g = -(R*W');

% Return a vector
g = g(:);
end
```



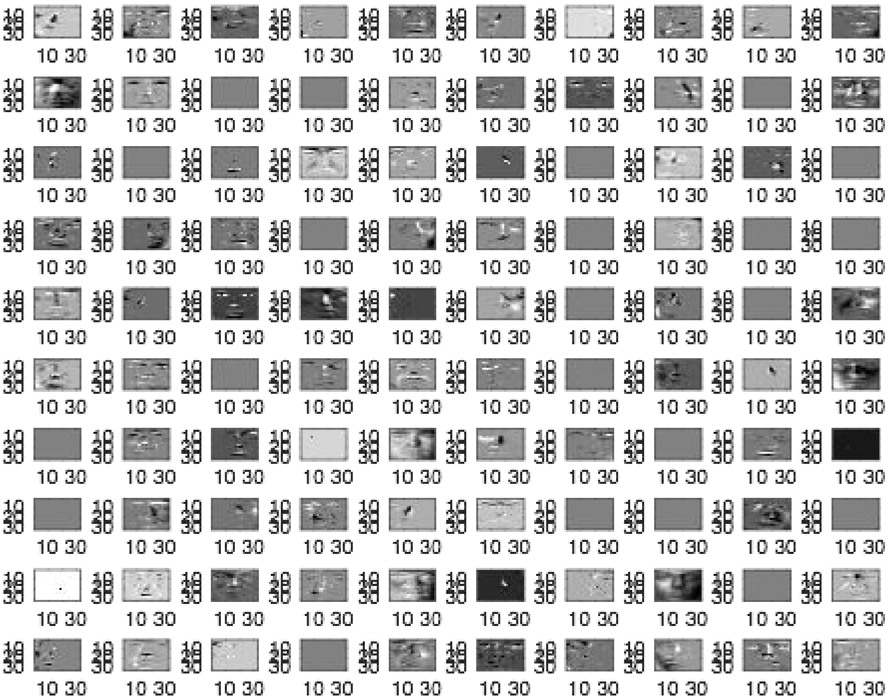
1.3 Sparse Matrix Factorization

code:

```
function [model] = dimRedSPCA(X,k,lambda)

[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(:) = findMinL1(@funObjZ,Z(:),lambda,10,0,X,W);
```

```
    % Update W
    W(:) = findMinL1(@funObjW,W(:),lambda,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;
model.lambda = lambda;
model.Z = Z;
end

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


X = X - repmat(mu,[t 1]);
% We didn't enforce that W was orthogonal so we need to solve least squares

Z(:) = findMinL1(@funObjZ,Z(:),lambda,10,0,X,W);
end

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

X = Z*W + repmat(mu,[t 1]);
end

function [f,g] = funObjW(W,X,Z)
% Resize vector of parameters into matrix
d = size(X,2);
k = size(Z,2);
W = reshape(W,[k d]);
```

```
% Compute function and gradient
R = X-Z*W;
f = (1/2)*sum(sum(R.^2));
g = -Z'*R;

% Return a vector
g = g(:);
end

function [f,g] = funObjZ(Z,X,W)
% Resize vector of parameters into matrix
n = size(X,1);
k = size(W,1);
Z = reshape(Z,[n k]);

% Compute function and gradient
R = X-Z*W;
f = (1/2)*sum(sum(R.^2));
g = -(R*W');

% Return a vector
g = g(:);
end
```



2 Recommender Systems

2.1 Latent-Factor Model(Picture)

1 2o Latent factor Model

↓

not directly measureable, existing in hidden form

Question 2.1 Latent Factor Model

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

Given the notation $r_{um} = (y_{um} - (b_u + b_m + W_m^T z_u))$

$\frac{\partial f}{\partial b_u} = \frac{1}{2} \cdot 2 \cdot (y_{um} - (b_u + b_m + W_m^T z_u)) \cdot (-1) = -r_{um}$

$\frac{\partial f}{\partial b_m} = \frac{1}{2} \cdot 2 (y_{um} - (b_u + b_m + W_m^T z_u)) \cdot (-1) = -r_{um}$

$\frac{\partial f}{\partial (W_m)_i} = \frac{1}{2} \cdot 2 \cdot (y_{um} - (b_u + b_m + W_m^T z_u)) \cdot (-1) \cdot (z_u)_i = -r_{um} \cdot (z_u)_i$

$\frac{\partial f}{\partial (z_u)_i} = \frac{1}{2} \cdot 2 \cdot (y_{um} - (b_u + b_m + W_m^T z_u)) \cdot (-1)(W_m)_i^T = -r_{um}(W_m)_i^T$

2.2 Stochastic Gradient

Code:

```
function [model] = recommendSVD_stochastic(X,y,k)

n = max(X(:,1));
d = max(X(:,2));
nRatings = size(X,1);

% Initialize parameters
% - for the biases, we'll use the user/item averages
% - for the latent factors, we'll use small random values
subModel = recommendUserItemMean(X,y);
bu = subModel.bu/2;
bm = subModel.bm/2;
W = .00001*randn(k,d);
Z = .00001*randn(n,k);

% Optimization
maxIter = 10;
alpha = 0.01;
```

```
for iter = 1:maxIter



   for j=1:nRatings

      % Compute gradient
      gu = zeros(n,1);
      gm = zeros(d,1);
      gW = zeros(k,d);
      gZ = zeros(n,k);
      % Randomly pick a index
      i = randi(nRatings,1);

      % 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) = -r;
      gm(m) = -r;
      gW(:,m) = -r*Z(u,:)';
      gZ(u,:) = -r*W(:,m)';


      % 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;
   end



   % 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

model.bu = bu;
model.bm = bm;
model.W = W;
model.Z = Z;
model.predict = @predict;
end

function [y] = predict(model,X)
t = size(X,1);
bu = model.bu;
bm = model.bm;
W = model.W;
Z = model.Z;

y = zeros(t,1);
for i = 1:t
    u = X(i,1);
    m = X(i,2);
    y(i) = bu(u) + bm(m) + W(:,m)'*Z(u,:)'; % Take the average between user and movie
ratings
end
end
```

Output for validation error:

Iter = 1, f = 4.035908e+05
Iter = 2, f = 4.010224e+05
Iter = 3, f = 4.001305e+05
Iter = 4, f = 3.997262e+05
Iter = 5, f = 3.997792e+05
Iter = 6, f = 3.995766e+05
Iter = 7, f = 3.995450e+05
Iter = 8, f = 3.993988e+05
Iter = 9, f = 3.994410e+05
Iter = 10, f = 3.993463e+05
Average absolute error by using stochastic gradient descent: 0.738913

3 Multi-Dimensional Scaling

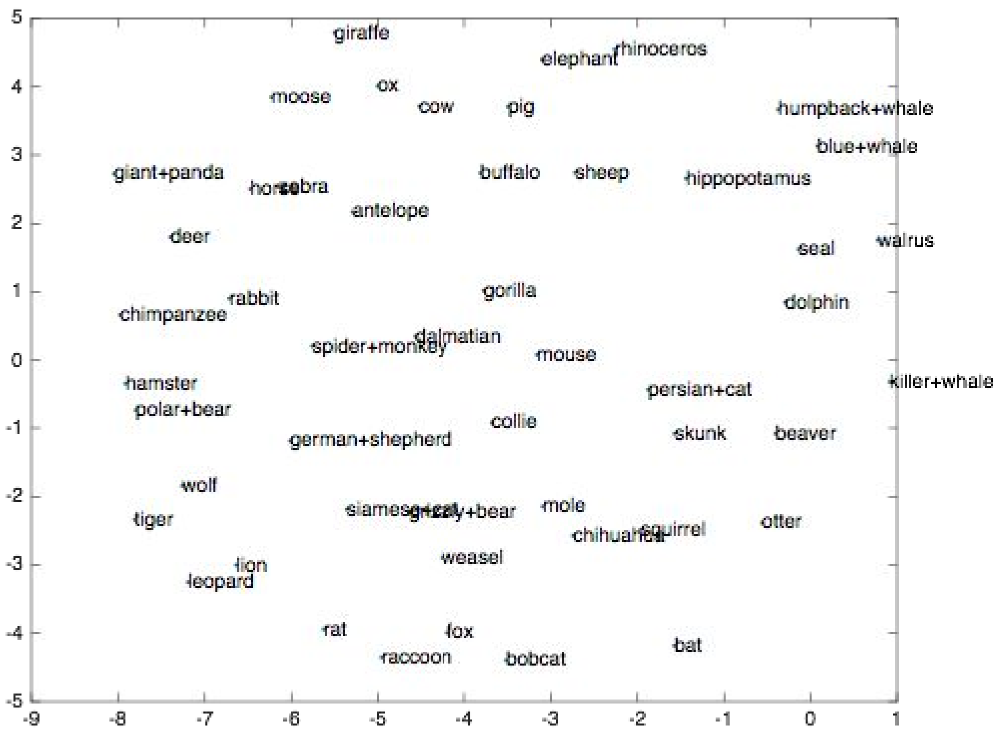
3.1 Samman Mapping

code:

```
function [Z] = visualizeSammon(X,k,names)
```

```
[n,d] = size(X);

% Compute all distances
D = X.^2*ones(d,n) + ones(n,d)*(X').^2 - 2*X*X';
D = sqrt(abs(D));

% Initialize low-dimensional representation with PCA
[U,S,V] = svd(X);
W = V(:,1:k)';
Z = X*W';

Z(:) = findMin(@stress,Z(:),500,0,D,names);

end

function [f,g] = stress(Z,D,names)

n = length(D);
k = numel(Z)/n;

Z = reshape(Z,[n k]);

f = 0;
g = zeros(n,k);
for i = 1:n
    for j = i+1:n
        % Objective Function
        Dz = norm(Z(i,:)-Z(j,:));
        s = D(i,j) - Dz;
        f = f + (1/2)*s^2/(D(i,j)^2);

        % Gradient
        df = s;
        dgi = (Z(i,:)-Z(j,:))/(Dz*D(i,j));
        dgj = (Z(j,:)-Z(i,:))/(Dz*D(i,j));
        g(i,:) = g(i,:) - df*dgi;
        g(j,:) = g(j,:) - df*dgj;
    end
end
g = g(:);

% Make plot if using 2D representation
if k == 2
    figure(3);
    clf;
    plot(Z(:,1),Z(:,2),'.');
```

```
    hold on;
    for i = 1:n
        text(Z(i,1),Z(i,2),names(i,:));
    end
    pause(.01)
end
end
```

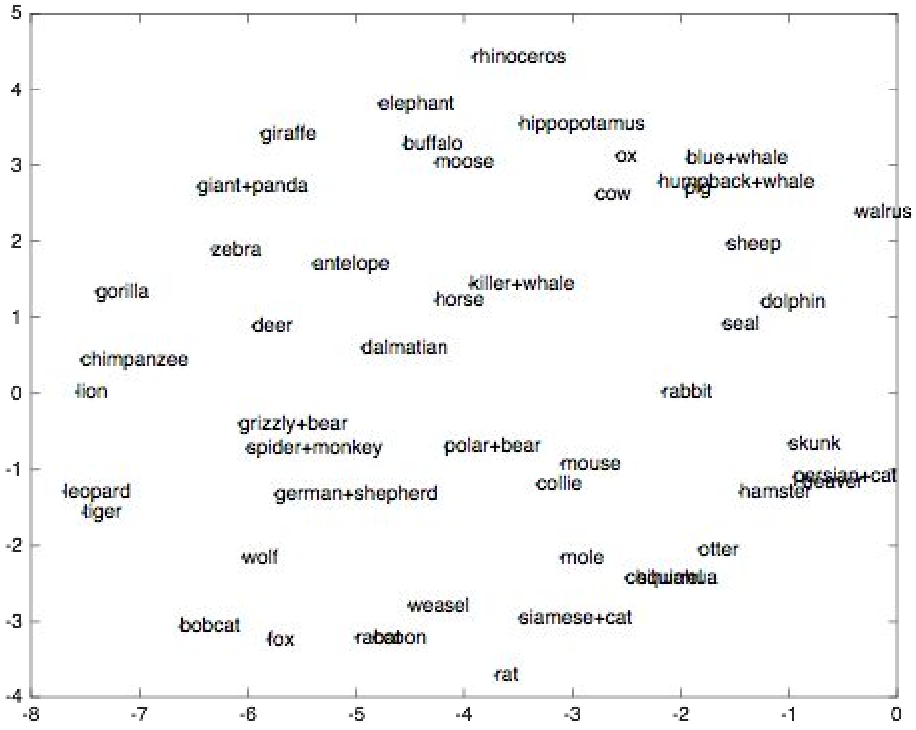


3.2 ISOMAP

```
function [Z] = visualizeISOMAP(X,k,names)

[n,d] = size(X);

% Compute all distances
D = X.^2*ones(d,n) + ones(n,d)*(X').^2 - 2*X*X';
D = sqrt(abs(D));

% Find the K nearest neighbor
% G(i,j) = D(i,j); if j belongs to neighbors(i)
G = zeros(n,n);
for i = 1:n
    test = D(:,i);
```

```
    [sortDist,sortIndex] = sort(test,'ascend');
    minIndex = sortIndex(2:k+1);
    for j = 1:k
        G(i,minIndex(j)) = D(i,minIndex(j));
    end
end



% Initialize low-dimensional representation with PCA
[U,S,V] = svd(X);
W = V(:,1:k)';
Z = X*W';

Z(:) = findMin(@stress,Z(:),500,0,D,names);


% Using Dijkstra's  algorithm
for i = 1:n
    for j = 1:n
        D(i,j) = dijkstra(G,i,j);
    end
end
end


function [f,g] = stress(Z,D,names)

n = length(D);
k = numel(Z)/n;

Z = reshape(Z,[n k]);

f = 0;
g = zeros(n,k);
for i = 1:n
    for j = i+1:n
        % Objective Function
        Dz = norm(Z(i,:)-Z(j,:));
        s = D(i,j) - Dz;
        f = f + (1/2)*s^2;

        % Gradient
        df = s;
        dgi = (Z(i,:)-Z(j,:))/Dz;
        dgj = (Z(j,:)-Z(i,:))/Dz;
```

```
        g(i,:) = g(i,:) - df*dgi;
        g(j,:) = g(j,:) - df*dgj;
    end
end
g = g(:);

% Make plot if using 2D representation
if k == 3
    figure(3);
    clf;
    plot(Z(:,1),Z(:,2),'.');
    hold on;
    for i = 1:n
        text(Z(i,1),Z(i,2),names(i,:));
    end
    pause(.01)
end
end
```

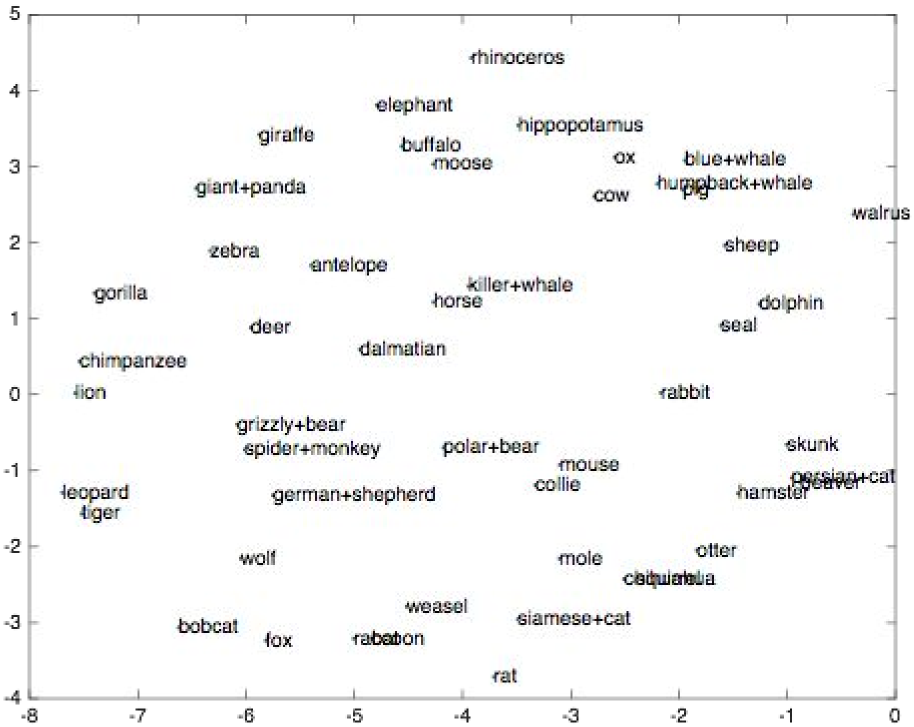


3.3 ISOMAP with Disconnected Graph

```
% Modified version

function [Z] = visualizeISOMAP(X,k,names)
```

```
[n,d] = size(X);

% Compute all distances
D = X.^2*ones(d,n) + ones(n,d)*(X').^2 - 2*X*X';
D = sqrt(abs(D));

% Find the K nearest neighbor
% G(i,j) = D(i,j); if j belongs to neighbors(i)
G = zeros(n,n);
for i = 1:n
   test = D(:,i);
   [sortDist,sortIndex] = sort(test,'ascend');
   minIndex = sortIndex(2:k+1);
   for j = 1:k
      G(i,minIndex(j)) = D(i,minIndex(j));
   end
end



% Initialize low-dimensional representation with PCA
[U,S,V] = svd(X);
W = V(:,1:k)';
Z = X*W';

Z(:) = findMin(@stress,Z(:),500,0,D,names);


% Using Dijkstra's  algorithm
for i = 1:n
   for j = 1:n
      D(i,j) = dijkstra(G,i,j);
   end
end

Dinf=find(isinf(D));
DNinf=find(~isinf(D));
D(Dinf)=max(D(DNinf));
end


function [f,g] = stress(Z,D,names)

n = length(D);
k = numel(Z)/n;
```

```
Z = reshape(Z,[n k]);

f = 0;
g = zeros(n,k);
for i = 1:n
    for j = i+1:n
        % Objective Function
        Dz = norm(Z(i,:)-Z(j,:));
        s = D(i,j) - Dz;
        f = f + (1/2)*s^2;

        % Gradient
        df = s;
        dgi = (Z(i,:)-Z(j,:))/Dz;
        dgj = (Z(j,:)-Z(i,:))/Dz;
        g(i,:) = g(i,:) - df*dgi;
        g(j,:) = g(j,:) - df*dgj;
    end
end
g = g(:);

% Make plot if using 2D representation
if k == 3
    figure(3);
    clf;
    plot(Z(:,1),Z(:,2),'.');
    hold on;
    for i = 1:n
        text(Z(i,1),Z(i,2),names(i,:));
    end
    pause(.01)
end
end
```



4 Visualizing a neural net for 1D regression

code:

```
load nnetData.mat % Loads data {X,y}
[N,d] = size(X);

% Add bias
X = [ones(N,1) X];
d = d + 1;

% Choose network structure
nHidden = [5, 5];

% Count number of parameters and initialize weights 'w'
nParams = d*nHidden(1);
for h = 2:length(nHidden)
   nParams = nParams+nHidden(h-1)*nHidden(h);
end
nParams = nParams+nHidden(end);
w = randn(nParams,1);

% Train with stochastic gradient
maxIter = 100000;
```

```
stepSize = 1e-2;
funObj = @(w,i)MLPregressionLoss(w,X(i,:),y(i),nHidden);
for t = 1:maxIter

   % Every few iterations, plot the data/model:
   if mod(t-1,round(maxIter/100)) == 0
      fprintf('Training iteration = %d\n',t-1);
      figure(1);clf;hold on
      Xhat = [-5:.05:5]';
      Xhat = [ones(size(Xhat,1),1) Xhat];
      yhat = MLPregressionPredict(w,Xhat,nHidden);
      plot(X(:,2),y,'.');
      h=plot(Xhat(:,2),yhat,'g-');
      set(h,'LineWidth',3);
      legend({'Data','Neural Net'});
      drawnow;
   end

   % The actual stochastic gradient algorithm:
   i = ceil(rand*N);
   [f,g] = funObj(w,i);
   w = w - stepSize*g;
end
```