CPSC 340 Tutorial
Assignment 5 part 2

Questions 3 to 5

Multi-Dimensional Scaling (MDS)

The function *example_MDS* loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

$X \in R^{n \times d}$

$$\operatorname*{argmin}_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2. \tag{1}$$

Multi-Dimensional Scaling (MDS)

The function *example_MDS* loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

$X \in R^{n \times d}$

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2. \tag{1}$$

$x_i \in R^d$

$z_i \in R^k$

Multi-Dimensional Scaling (MDS)

The function *example_MDS* loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

$$\operatorname*{argmin}_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2. \tag{1}$$

- Recall: principal component analysis (PCA) projects d-dimensional data points to a hyperplane orthogonal to the directions of maximal variance.

$X \in R^{n \times d}$

$x_i \in R^d$

$z_i \in R^k$

$SVD(X) = U, \Sigma, V^T$

$W = V^T_{(1:d,1:k)}$ Eigenvectors of (directional vectors) $\rightarrow$ $X^T X$

Covariance matrix

$Z = XW^T$

Multi-Dimensional Scaling (MDS)

The function *example_MDS* loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2. \tag{1}$$

- Recall: principal component analysis (PCA) projects d-dimensional data points to a hyperplane orthogonal to the directions of maximal variance
- PCA preserves **covariance** between the data points

Multi-Dimensional Scaling (MDS)

The function *example_MDS* loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2. \tag{1}$$

- MDS projects data points to a space where similar data points are clustered together
- MDS preserves distances between points

Multi-Dimensional Scaling (MDS)

The function *example_MDS* loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

Optimizing for ***Z***

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2.$$

Original distances but in d-dimensional space

Derives k-dimensional data points such that the original distances are preserved

- MDS preserves distances between points

Multi-Dimensional Scaling (MDS)

The function *example_MDS* loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2.$$

Original distances but in d-dimensional space

Derives k-dimensional data points such that the original distances are preserved

- We want $||x_i - x_j|| \approx ||z_i - z_j|| \quad \forall i, j$
- where, for example,

$$x_i \in R^{50} \qquad z_i \in R^2$$

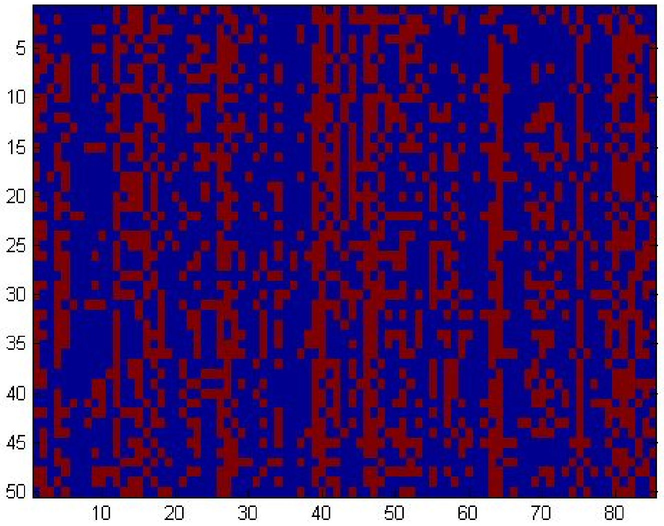
exampleMDS.m file

```
load animals.mat
[n,d] = size(X);

% Figure 1 shows raw data
figure(1);
imagesc(X);

% Figure 2 shows PCA visualization
figure(2);clf;
[U,S,V] = svd(X);
W = V(:,1:2)';
Z = X*W';
figure(2);
plot(Z(:,1),Z(:,2),'.');
hold on;
for i = 1:n
    text(Z(i,1),Z(i,2),animals(i,:));
end

% Figure 3 shows MDS visualization
z = visualizeMDS(X,2,animals);
```

```
load animals.mat
[n,d] = size(X);

% Figure 1 shows raw data
figure(1);
imagesc(X);

% Figure 2 shows PCA visualization
figure(2);clf;
[U,S,V] = svd(X);
W = V(:,1:2)';
Z = X*W';
figure(2);
plot(Z(:,1),Z(:,2),'.');
hold on;
for i = 1:n
    text(Z(i,1),Z(i,2),animals(i,:));
end

% Figure 3 shows MDS visualization
z = visualizeMDS(X,2,animals);
```

Figure 1 - Displays matrix *X*

n samples



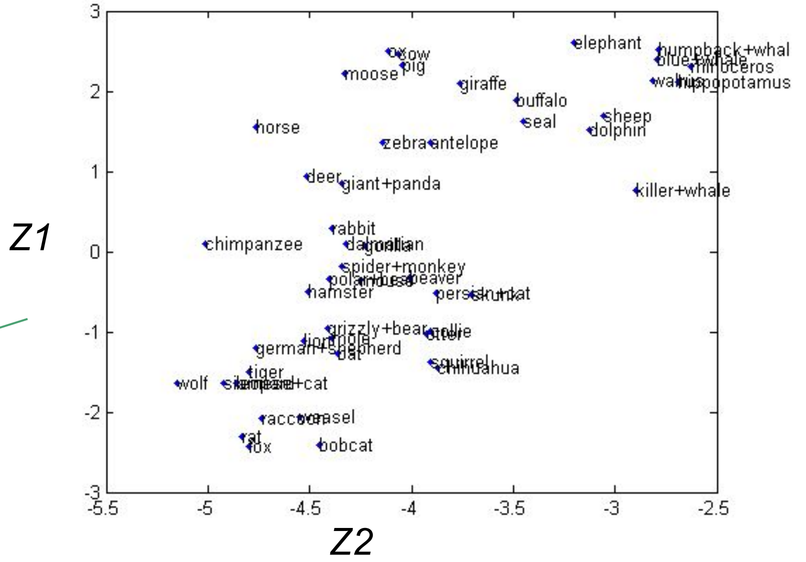
d features

X consists of values 0 and 1

blue is 0; red is 1

```matlab
load animals.mat
[n,d] = size(X);

% Figure 1 shows raw data
figure(1);
imagesc(X);

% Figure 2 shows PCA visualization
figure(2);clf;
[U,S,V] = svd(X);
W = V(:,1:2)';
Z = X*W';
figure(2);
plot(Z(:,1),Z(:,2),'.');
hold on;
for i = 1:n
    text(Z(i,1),Z(i,2),animals(i,:));
end

% Figure 3 shows MDS visualization
z = visualizeMDS(X,2,animals);
```

Figure 2 - Displays the projection of *X* onto the first two Principal components

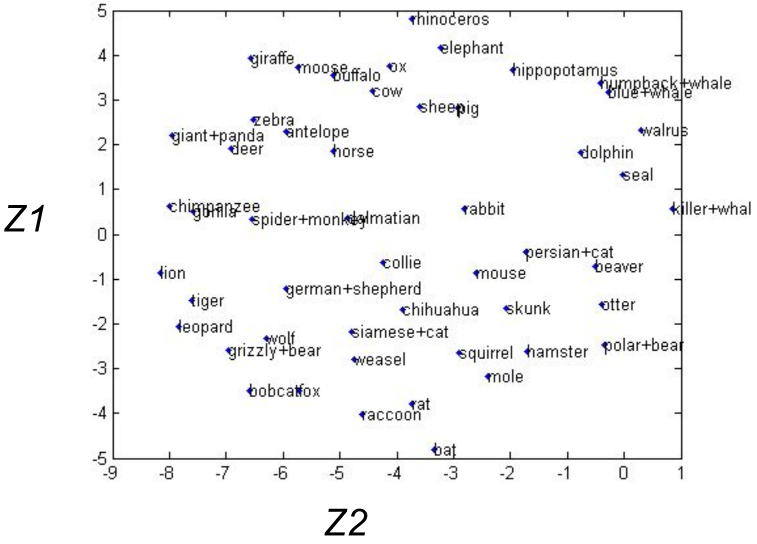


$$X(i) = [Z_{i1}, Z_{i2}]$$

```
load animals.mat
[n,d] = size(X);

% Figure 1 shows raw data
figure(1);
imagesc(X);

% Figure 2 shows PCA visualization
figure(2);clf;
[U,S,V] = svd(X);
W = V(:,1:2)';
Z = X*W';
figure(2);
plot(Z(:,1),Z(:,2),'.');
hold on;
for i = 1:n
    text(Z(i,1),Z(i,2),animals(i,:));
end

% Figure 3 shows MDS visualization
z = visualizeMDS(X,2,animals);
```

$$\underset{Z\in\mathbb{R}^{n\times k}}{\operatorname{argmin}} \frac{1}{2}\sum_{i=1}^{n}\sum_{j=i+1}^{n}(\|x_i - x_j\| - \|z_i - z_j\|)^2$$

Figure 3 - Displays the datasets in terms of the two latent features obtained from MDS



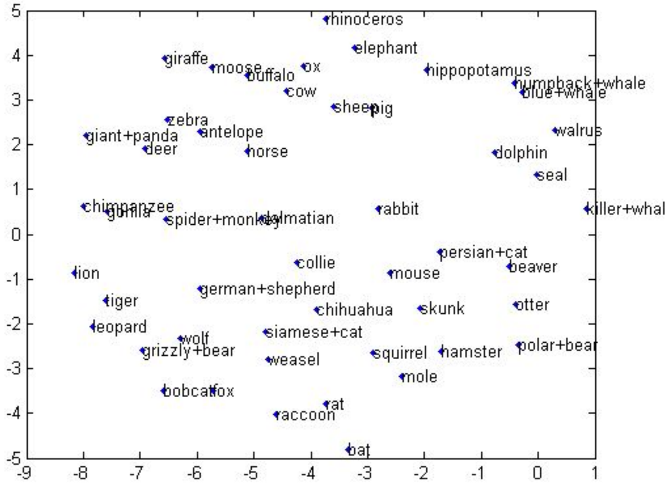
$$X(i) = [Z_{i1}, Z_{i2}]$$

```
load animals.mat
[n,d] = size(X);

% Figure 1 shows raw data
figure(1);
imagesc(X);

% Figure 2 shows PCA visualization
figure(2);clf;
[U,S,V] = svd(X);
W = V(:,1:2)';
Z = X*W';
figure(2);
plot(Z(:,1),Z(:,2),'.');
hold on;
for i = 1:n
    text(Z(i,1),Z(i,2),animals(i,:));
end

% Figure 3 shows MDS visualization
z = visualizeMDS(X,2,animals);
```

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2$$

Figure 3 - Displays the datasets in terms of the two latent features obtained from MDS

Z1



Z2

$$X(i) = [Z_{i1}, Z_{i2}]$$

```matlab
function [Z] = visualizeMDS(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
```

Computes the distances between each pair of samples $D(i,j) = ||x_i - x_j||^2$

Initializes *Z* using *PCA*

MDS

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2$$

```
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(:);
```

MDS

$$\underset{Z\in\mathbb{R}^{n\times k}}{\operatorname{argmin}} \frac{1}{2}\sum_{i=1}^{n}\sum_{j=i+1}^{n}(\|x_i - x_j\| - \|z_i - z_j\|)^2.$$

MDS function value

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

```matlab
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(:);
```

MDS

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2.$$

MDS function value

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

```
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(:);
```

MDS

$$\underset{Z\in\mathbb{R}^{n\times k}}{\operatorname{argmin}} \frac{1}{2}\sum_{i=1}^{n}\sum_{j=i+1}^{n}(\|x_i - x_j\| - \|z_i - z_j\|)^2.$$

MDS function value

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

```
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(:);
```

MDS

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2.$$

MDS function value

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

```matlab
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(:);
```

MDS function value

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

MDS gradient w.r.t z_i and z_j

$$s = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(||x_i - x_j|| - ||z_i - z_j||)$$

$$s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2}$$

$$\boxed{f(Z) = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(\frac{1}{2}s_{ij}^2)}$$

```
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(:);
```

MDS function value

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

MDS gradient w.r.t z_i and z_j

$$s = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(||x_i - x_j|| - ||z_i - z_j||)$$

$$s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2}$$

$$\boxed{f(Z) = \sum_{i=1}^{n}\sum_{j=i+1}^{n}\left(\frac{1}{2}s_{ij}^2\right)}$$

Derivative

$$\frac{\partial f_{ij}}{z_i} = \frac{\partial f_{ij}}{s_{ij}}\frac{\partial s_{ij}}{z_i}$$

```matlab
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(:);
```

MDS function value

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

MDS gradient w.r.t z_i and z_j

$$s = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(||x_i - x_j|| - ||z_i - z_j||)$$

$$s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2}$$

$$\boxed{f(Z) = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(\tfrac{1}{2}s_{ij}^2)}$$

Derivative

$$\frac{\partial f_{ij}}{z_i} = \frac{\partial f_{ij}}{s_{ij}}\frac{\partial s_{ij}}{z_i}$$

$$\frac{\partial f_{ij}}{\partial s_{ij}} = \frac{\partial(\frac{1}{2}s_{ij}^2)}{\partial s_{ij}} = ?$$

```matlab
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(:);
```

MDS function value

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

MDS gradient w.r.t z_i and z_j

$$s = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(||x_i - x_j|| - ||z_i - z_j||)$$

$$s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2}$$

$$\boxed{f(Z) = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(\tfrac{1}{2}s_{ij}^2)}$$

Derivative

$$\frac{\partial f_{ij}}{z_i} = \frac{\partial f_{ij}}{s_{ij}}\frac{\partial s_{ij}}{z_i}$$

$$\frac{\partial f_{ij}}{\partial s_{ij}} = \frac{\partial(\frac{1}{2}s_{ij}^2)}{\partial s_{ij}} = s_{ij}$$

```matlab
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(:);
```

MDS function value

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

MDS gradient w.r.t z_i and z_j

$$s = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(||x_i - x_j|| - ||z_i - z_j||)$$

$$s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2}$$

$$\boxed{f(Z) = \sum_{i=1}^{n}\sum_{j=i+1}^{n}\left(\frac{1}{2}s_{ij}^2\right)}$$

Derivative

$$\frac{\partial f_{ij}}{z_i} = \frac{\partial f_{ij}}{s_{ij}}\frac{\partial s_{ij}}{z_i}$$

$$\frac{\partial f_{ij}}{\partial s_{ij}} = \frac{\partial(\frac{1}{2}s_{ij}^2)}{\partial s_{ij}} = s_{ij}$$

$$\frac{\partial s_{ij}}{\partial z_i} = \frac{\partial(\sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2})}{\partial z_i} = ?$$

```matlab
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(:);
```

MDS function value

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

MDS gradient w.r.t z_i and z_j

$$s = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(||x_i - x_j|| - ||z_i - z_j||)$$

$$s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2}$$

$$\boxed{f(Z) = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(\tfrac{1}{2}s_{ij}^2)}$$

Derivative

$$\frac{\partial f_{ij}}{z_i} = \frac{\partial f_{ij}}{s_{ij}}\frac{\partial s_{ij}}{z_i}$$

$$\frac{\partial f_{ij}}{\partial s_{ij}} = \frac{\partial(\frac{1}{2}s_{ij}^2)}{\partial s_{ij}} = s_{ij}$$

$$\frac{\partial s_{ij}}{\partial z_i} = \frac{\partial(\sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2})}{\partial z_i} = -\frac{z_i - z_j}{\sqrt{(z_i - z_j)^2}}$$

```
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(:);
```

MDS function value

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

MDS gradient w.r.t z_i and z_j

$$s = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (||x_i - x_j|| - ||z_i - z_j||)$$

$$s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2}$$

$$\boxed{f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\frac{1}{2} s_{ij}^2)}$$

Derivative

$$\frac{\partial f_{ij}}{z_i} = \frac{\partial f_{ij}}{s_{ij}} \frac{\partial s_{ij}}{z_i}$$

$$\frac{\partial f}{z_i} = \frac{\partial f}{\partial s} \frac{\partial s}{z_i} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (s_{ij}) \left(- \frac{z_i - z_j}{\sqrt{(z_i - z_j)^2}} \right)$$

Question 3.1: *visualizeSammon*

Make new function *visualizeSammon* that implements gradient descent for MDS Sammon mapping objetive,

Sammon's mapping objective function

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(\|x_i - x_j\| - \|z_i - z_j\|)^2}{\|x_i - x_j\|}.$$

MDS objective function

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2$$

Question 3.1: *visualizeSammon*

Make new function *visualizeSammon* that implements gradient descent for MDS Sammon mapping objetive,

Sammon's mapping objective function

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(\|x_i - x_j\| - \|z_i - z_j\|)^2}{\boxed{\|x_i - x_j\|}}.$$

MDS objective function

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2$$

Question 3.1: *visualizeSammon*

Sammon's mapping objective function

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

MDS objective function

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

```
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(:);
```

Question 3.1: *visualizeSammon*

Sammon's mapping objective function

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

Make changes here

Recall: $D(i,j) = ||x_i - x_j||$

MDS objective function

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

```
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(:);
```

Question 3.1: *visualizeSammon*

Sammon's mapping objective function

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

Make changes here

Recall: $D(i,j) = ||x_i - x_j||$

$s_{ij} = ?$

MDS objective function

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

$$s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2}$$

```
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(:);
```

Question 3.1: *visualizeSammon*

Sammon's mapping gradient function

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

$$\boxed{s_{ij} = \ ?}$$

MDS gradient function

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

$$\boxed{s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2}}$$

```
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(:);
```

Question 3.1: *visualizeSammon*

Sammon's mapping gradient function

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

$s_{ij} =$?

$$\boxed{\frac{\partial f}{z_i} = \frac{\partial f}{\partial s}\frac{\partial s}{z_i} = \quad ?}$$

Make changes here

MDS gradient function

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

$$s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2}$$

$$\boxed{\frac{\partial f}{z_i} = \frac{\partial f}{\partial s}\frac{\partial s}{z_i} = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(s_{ij})\left(-\frac{z_i - z_j}{\sqrt{(z_i - z_j)^2}}\right)}$$

```
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(:);
```

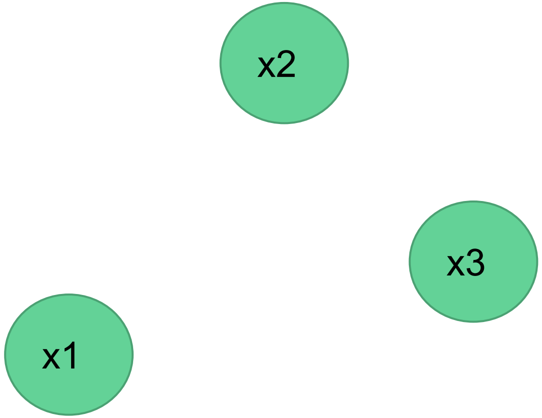
Question 3.2: ISOMAP

This suggests that ISOMAP may give a better visualization. Make a new function *visualizeISOMAP* that computes the approximate geodesic distance (shortest path through a graph where the edges are only between k-nearest neighbours, and the edge weights are the distances between these neighbours) between each pair of points, and then fits a standard MDS model (1) using gradient descent. Hand in your code and the plot of the result when using the 3-nearest neighbours.

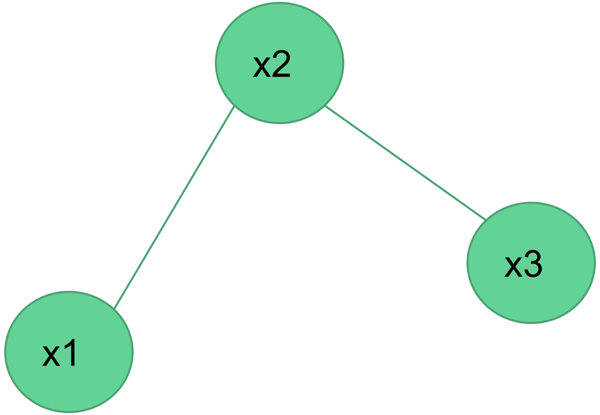
Question 3.2: ISOMAP

This suggests that ISOMAP may give a better visualization. Make a new function *visualizeISOMAP* that computes the approximate geodesic distance (shortest path through a graph where the edges are only between k-nearest neighbours, and the edge weights are the distances between these neighbours) between each pair of points, and then fits a standard MDS model (1) using gradient descent. Hand in your code and the plot of the result when using the 3-nearest neighbours.

Multi-dimensional scaling

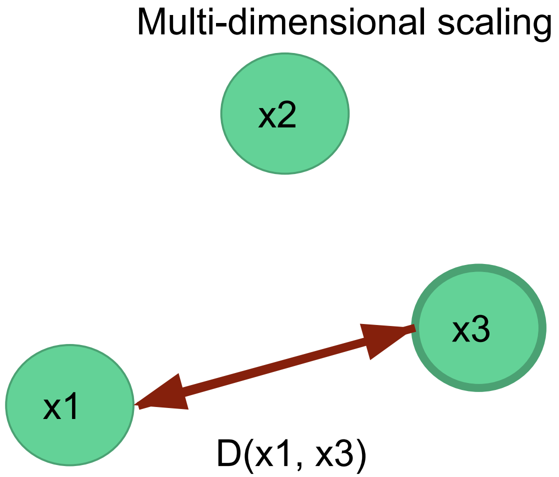


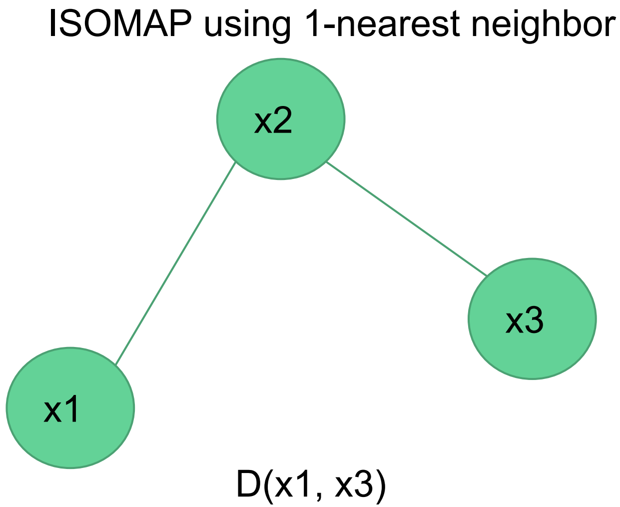
ISOMAP using 1-nearest neighbor



Question 3.2: ISOMAP

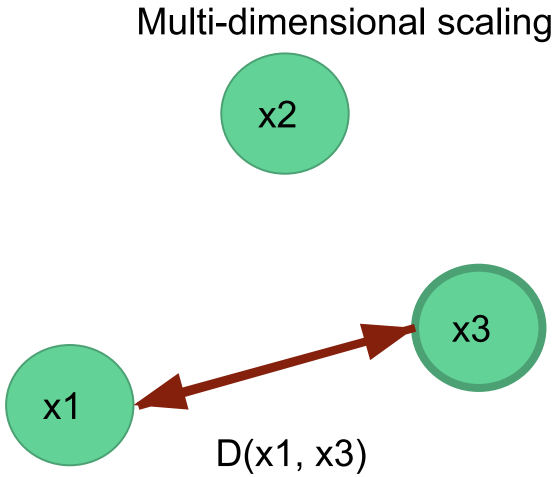
This suggests that ISOMAP may give a better visualization. Make a new function $\boxed{visualizeISOMAP}$ that computes the $\boxed{\text{approximate geodesic distance}}$ (shortest path through a graph where the edges are only between k-nearest neighbours, and the edge weights are the distances between these neighbours) between each pair of points, and then fits a standard MDS model (1) using gradient descent. Hand in your code and the plot of the result when using the 3-nearest neighbours.

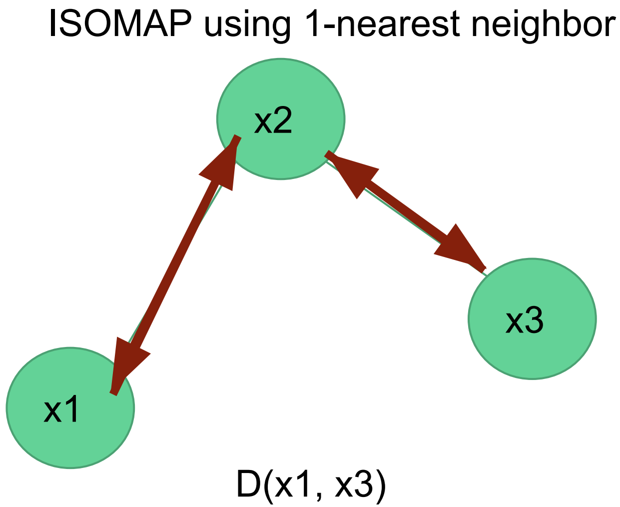




Question 3.2: ISOMAP

This suggests that ISOMAP may give a better visualization. Make a new function $\boxed{visualizeISOMAP}$ that computes the approximate geodesic distance (shortest path through a graph where the edges are only between k-nearest neighbours, and the edge weights are the distances between these neighbours) between each pair of points, and then fits a standard MDS model (1) using gradient descent. Hand in your code and the plot of the result when using the 3-nearest neighbours.





Question 3.2: ISOMAP

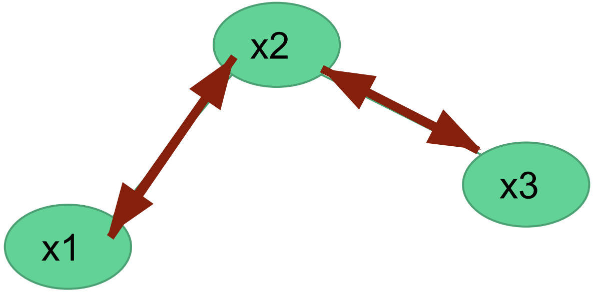
- If x is d-dimensional,
  - then ISOMAP with n-nearest neighbor is MDS
  - Otherwise the only difference is in the distance function

- If x is d-dimensional,
  - then ISOMAP with d-nearest neighbor is MDS
  - Otherwise the only difference is in the distance function

ISOMAP: Make changes here

```
function [Z] = visualizeMDS(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);
```

- If x is d-dimensional,
  - then ISOMAP with d-nearest neighbor is MDS
  - Otherwise the only difference is in the distance function

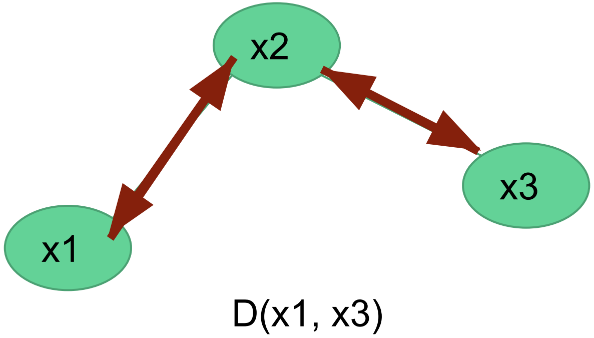


D(x1, x3)

ISOMAP:

1) Find k-nearest neighbors
   The 'k' points that are closest to each data point (see KNN)

```
function [Z] = visualizeMDS(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);
```

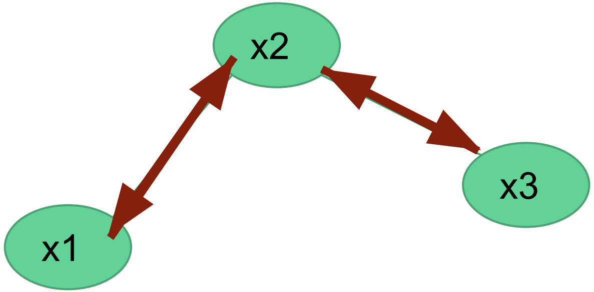



```
function [Z] = visualizeMDS(X,k,names)

[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);
```

ISOMAP:

2) Create *n x n zero* matrix *G*
(the adjacency graph)
s.t.

$$G(i,j) = \begin{cases} D(i,j) & \text{if } j \in \text{neighbors(i)} \\ 0 & \text{otherwise} \end{cases}$$



D(x1, x3)

```
function [Z] = visualizeMDS(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);
```

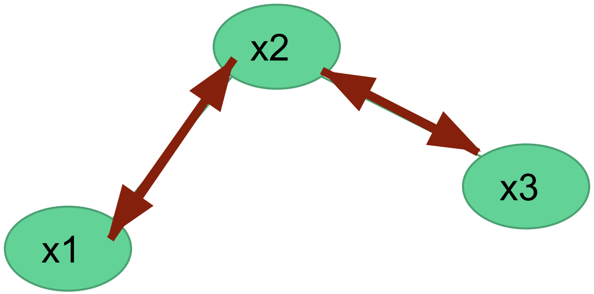
ISOMAP:

2) Create *n x n zero* matrix *G* *(the adjacency graph)* *s.t.*

$$G(i,j) = \begin{cases} D(i,j) & \text{if } j \in \text{neighbors(i)} \\ 0 & \text{otherwise} \end{cases}$$

3) Use the Djikstra function to get the shortest distance between each point *i* and *j*

```
D(i,j) = dijkstra(G,i,j)
```



D(x1, x3)

```matlab
function [Z] = visualizeMDS(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);
```

ISOMAP:

2) Create *n x n zero* matrix *G* *(the adjacency graph)* *s.t.*

$$G(i,j) = \begin{cases} D(i,j) & \text{if } j \in \text{neighbors(i)} \\ 0 & \text{otherwise} \end{cases}$$

3) Use the Djikstra function to get the shortest distance between each point *i* and *j*

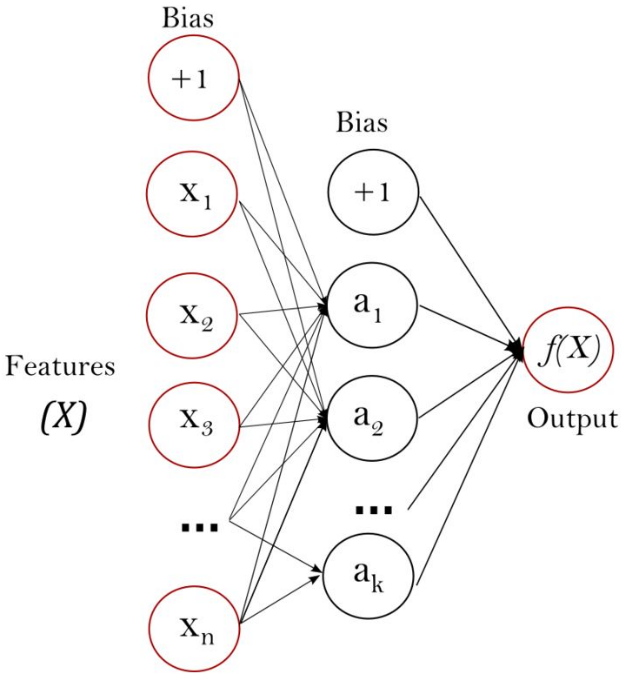
```matlab
D(i,j) = dijkstra(G,i,j)
```

Update the distance matrix

The adjacency matrix

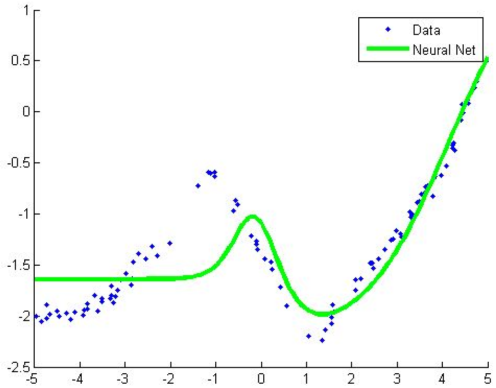
Q4: Visualizing a neural net for 1D regression

the data very well. Try to improve the performance of the method by changing the structure of the network (*nHidden* is a vector giving the number of hidden units in each layer) and the training procedure (e.g., change the sequence of step sizes, add momentum, or use *findMin* from the previous assignment). Hand in your plot after changing the code to have better performance, and list the changes you made.



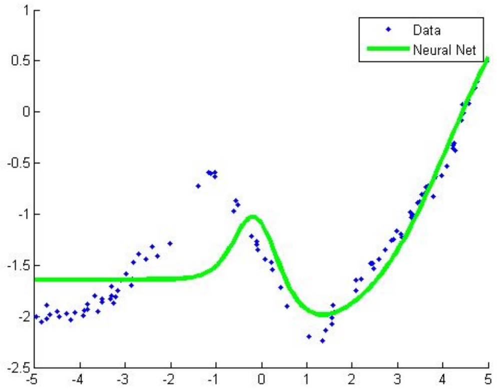
```matlab
load nnetData.mat % Loads data {X,y}
[N,d] = size(X);

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

% Choose network structure
nHidden = [3];

% 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-4;
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
```

Original performance

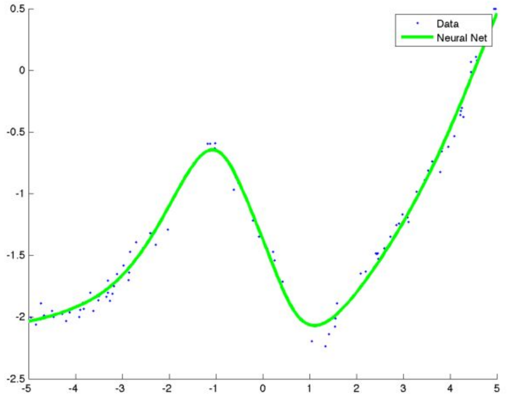


```
load nnetData.mat % Loads data {X,y}
[N,d] = size(X);

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

% Choose network structure
nHidden = [3];

% 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-4;
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
```

Original performance



Target performance



Improve the performance of your neural network

```
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 = [3];

% 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-4;
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
```

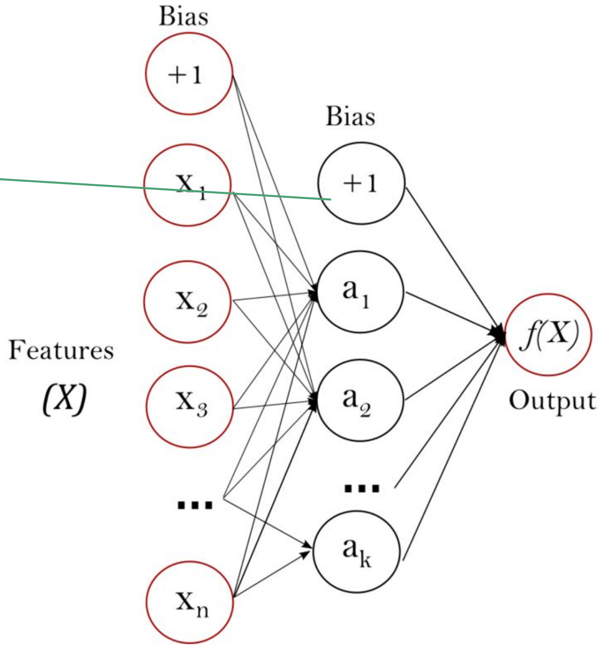
Number of hidden neurons

nParams: the total number of variables

The size of the step to take when updating 'w'

t: the epoch number

i: the index of the next sample '*x*' to be chosen
g: the gradient of that sample with respect to '*w*'



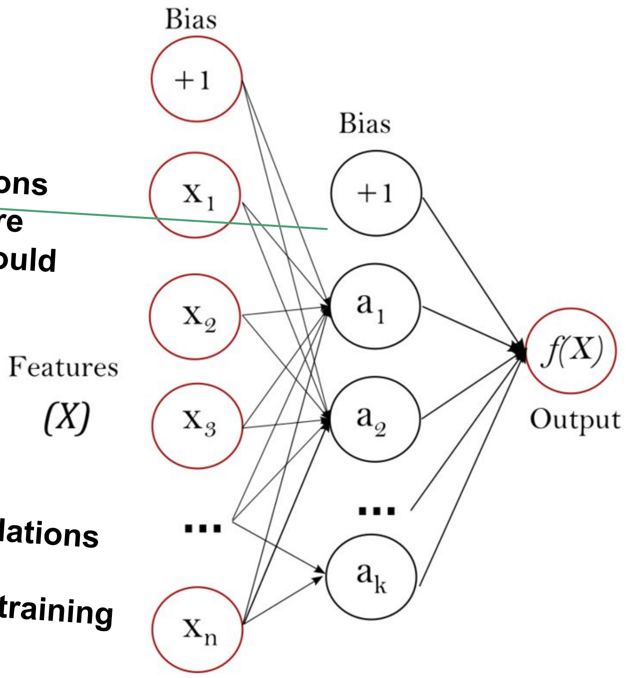
```
load nnetData.mat % Loads data {X,y}
[N,d] = size(X);

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

% Choose network structure
nHidden = [3];

% 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-4;
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
```

Improve the performance of your neural network

Adjust the number of hidden neurons (the more hidden neurons, the more powerful your model will be, but could cause overfitting)

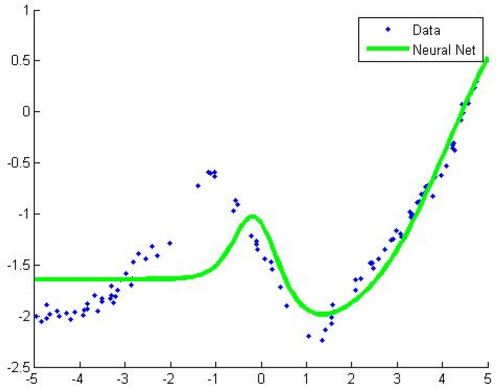
Adjust the step size

- Large step size can cause oscillations in your function value
- Small step size can cause slow training



```
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 = [3];

% 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-4;
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
```

Original performance



Target performance