Pattern Recognition

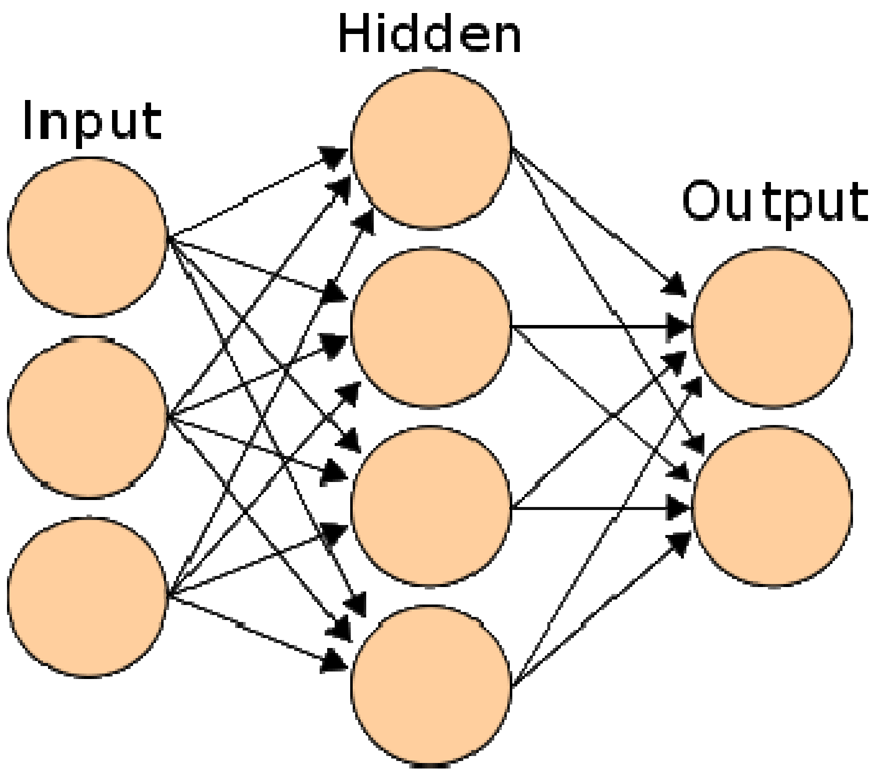
Neural Networks

Back Propagation Learning

Basic Structures of Neural Networks

- The individual neurons that make up a neural network are interconnected through their synapses.
- These connections allow the neurons to signal each other as information is processed.
- Each connection is assigned a connection weight.
- If there is no connection between two neurons, then their connection weight is zero.
- These weights are what determine the output of the neural network; therefore, the connection weights form the memory of the neural network.
- Training is the process by which these connection weights are assigned.

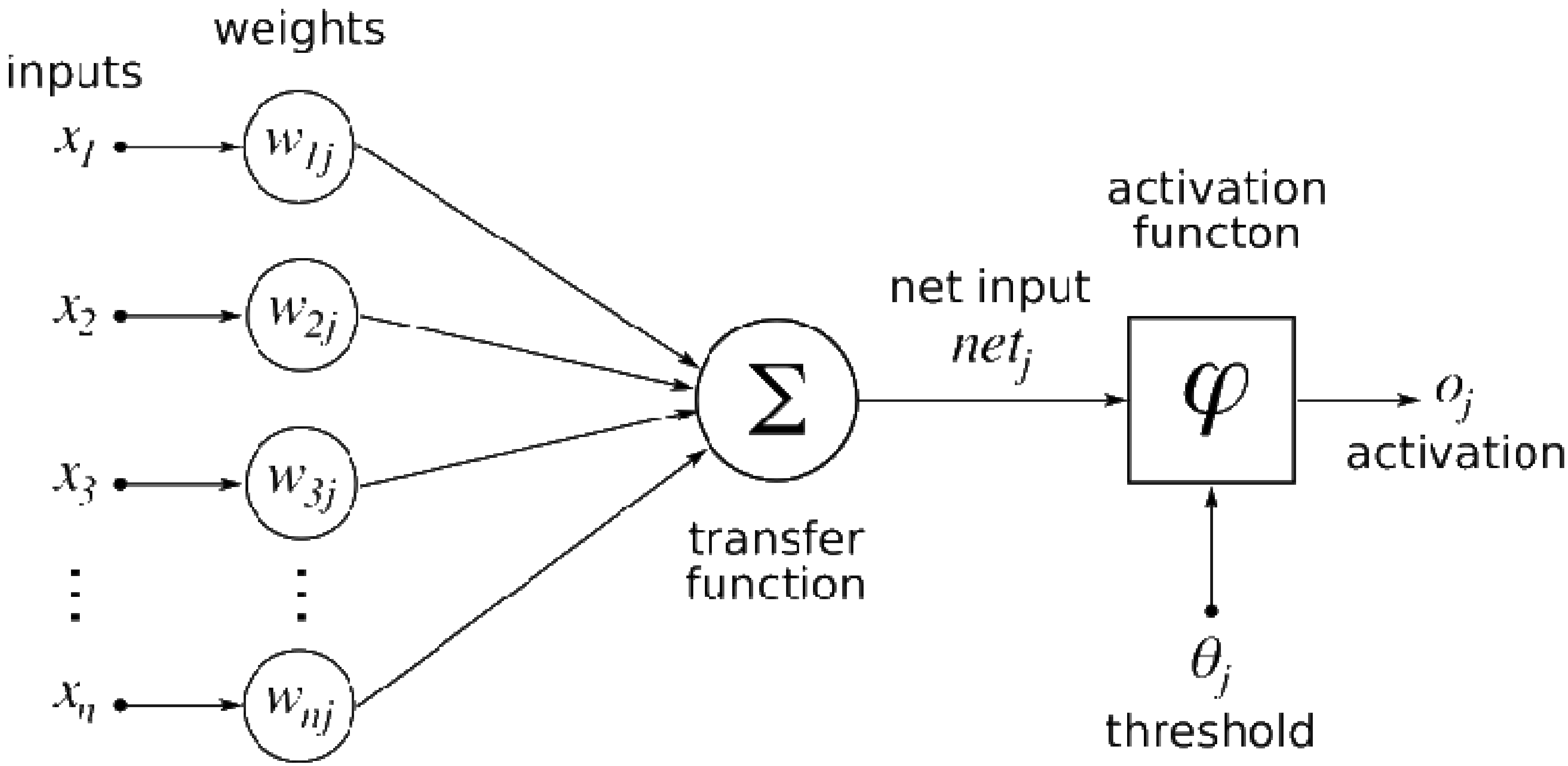
Structure of a Neural Network



Neural Network Training

- Unsupervised training:
  - In the unsupervised training, the neural network is not provided with anticipated outputs.
- Supervised Training
  - The neural network has access to the anticipated outputs

Structure of a Neuron



Feed-Forward and Back-Propagation Neural Networks

- Feed-forward describes how the neural network processes and recalls patterns.
- In a feed-forward neural network, neurons are only connected foreword.
- Back-propagation describes how the neural network is trained.
- Back-propagation is a form of supervised training.

Back-Propagation Learning

- Notation used:

$$X_j = \sum_i W_{ij} I_i$$

X_j is the total input to neuron *j*, W_{ij} is the weight of i^{th} input, I_i is the i^{th} input (output of neuron *i*).

- The output of neuron *j* is $y_j = \Phi(X_j)$

Error Term

- We sent in an input, and it generated, in the output nodes, a vector of outputs y.
  The correct answer is the vector of numbers O.
  The *error term* is:

$$E = \frac{1}{2}\sum_k (y_k - O_k)^2$$

Error Term

- For the output layer, we can write total error as a sum of the errors at each node k:

$$E = \frac{1}{2}\sum_k E_k$$

where

$$E_k = \frac{1}{2}(y_k - O_k)^2$$

The Learning Algorithm

- Variables y_k, x_k and w_{jk} each only affect the error at one particular output node k (they only affect E_k).
  So from the point of view of these 3 variables, total error:
- E = (a constant) + (error at node k) hence: (derivative of total error E with respect to any of these 3 variables) = 0 + (derivative of error at node k)
- we *can't* change y_k or x_k , but we *can* change w_{jk}

Partial Derivative (Output Layer)

$$\frac{\partial E}{\partial y_k} = \frac{1}{2} \times 2(y_k - O_k)$$

$$\frac{\partial E}{\partial x_k} = \frac{\partial E}{\partial y_k} \times \frac{\partial y_k}{\partial x_k} = \frac{\partial E}{\partial y_k} \times y_k(1 - y_k)$$

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial x_k} \times \frac{\partial x_k}{\partial w_{jk}} = \frac{\partial E}{\partial x_k} \times y_j$$

Partial Derivative (Hidden Layer)

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial x_k} \times \frac{\partial x_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial x_k} \times W_{jk}$$

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \times \frac{\partial y_j}{\partial x_j} = \frac{\partial E}{\partial y_j} \times y_j (1 - y_j)$$

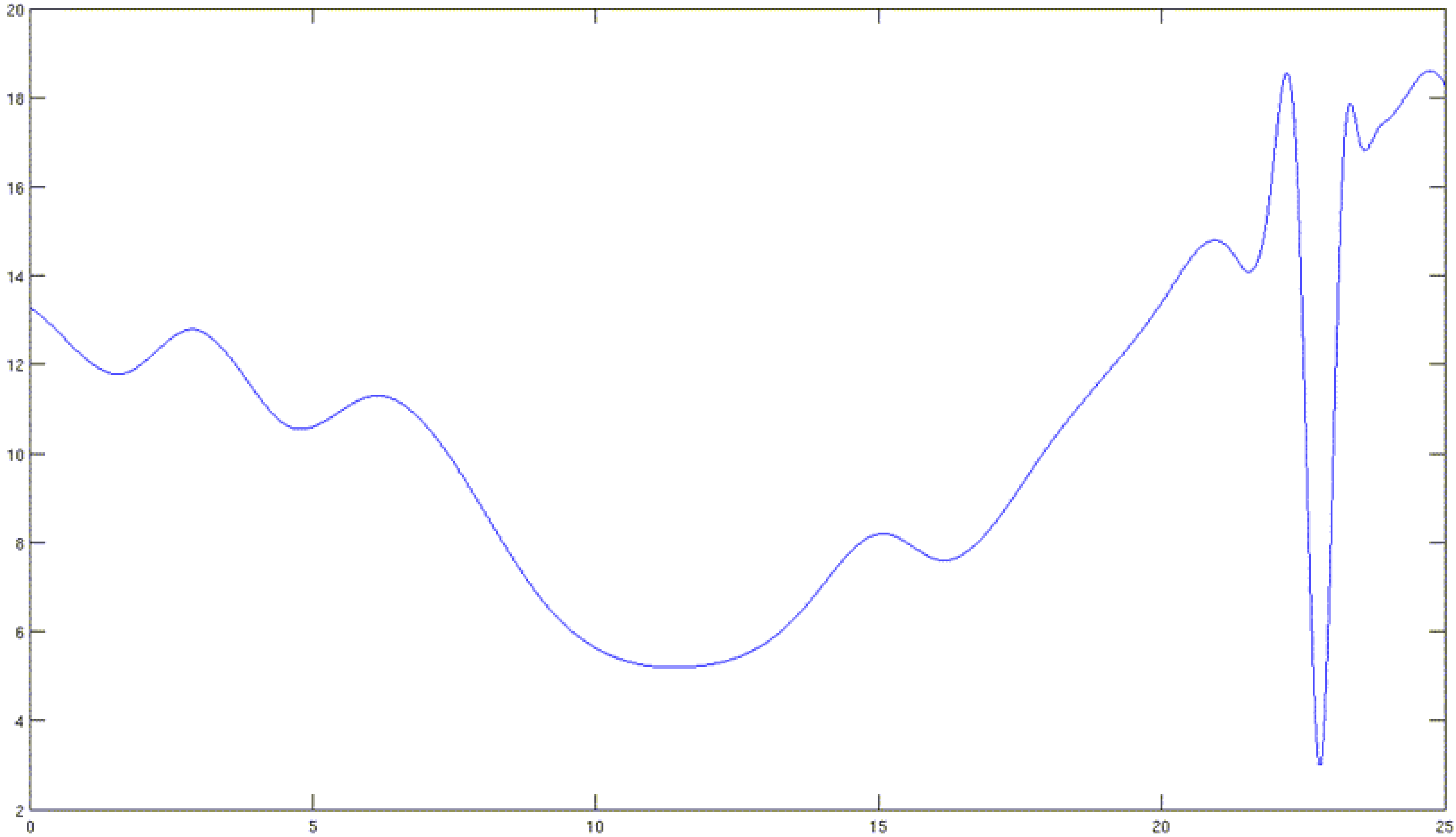
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial x_j} \times \frac{\partial x_j}{\partial w_{ij}} = \frac{\partial E}{\partial x_j} \times I_i$$

Changing the Weights to Reduce the Error

- Initialize connection weights into small random values.
- Present the k^{th} sample input vector and the corresponding output target to the network.
- For every neuron in every layer, find the output from the neuron
- Calculate error value for every neuron in every layer in backward order
- Perform weight adjustment for every connection from neuron in layer i-1 to every neuron in layer i by :

$$W_{jik} = W_{jik} + \beta \frac{\partial E}{\partial w_{jik}}$$

Hill-Climbing/Descending



Hill-Climbing/Descending

- If slope is *large* then our rule causes us to make *large* jumps in the direction that seems to be downhill.
- We do not *know* this will be downhill. We do not see the whole landscape. All we do is change the x value and hope the y value is smaller. It may turn out to have been a jump *uphill*.
- As we approach a minimum, the slope *must* approach zero and hence we make smaller jumps. As we get closer to the minimum, the slope is even closer to zero and so we make even smaller jumps. Hence the system *converges* to the minimum. Change in weights *slows down* to 0.
- If slope is zero we do not change the weight.
- As long as slope is not zero we will *keep* changing w. We will *never* stop until slope is zero.

Bias (Threshold) Values

- We need thresholds, otherwise the sigmoid function is centered at zero
- Update input to a neuron as:

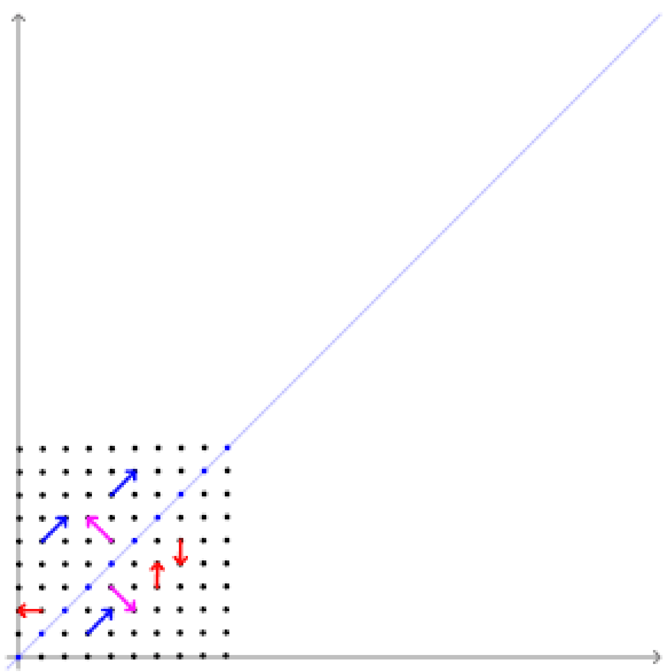
$$X_j = \sum_i W_{ij} I_i + (t_j)(-1)$$

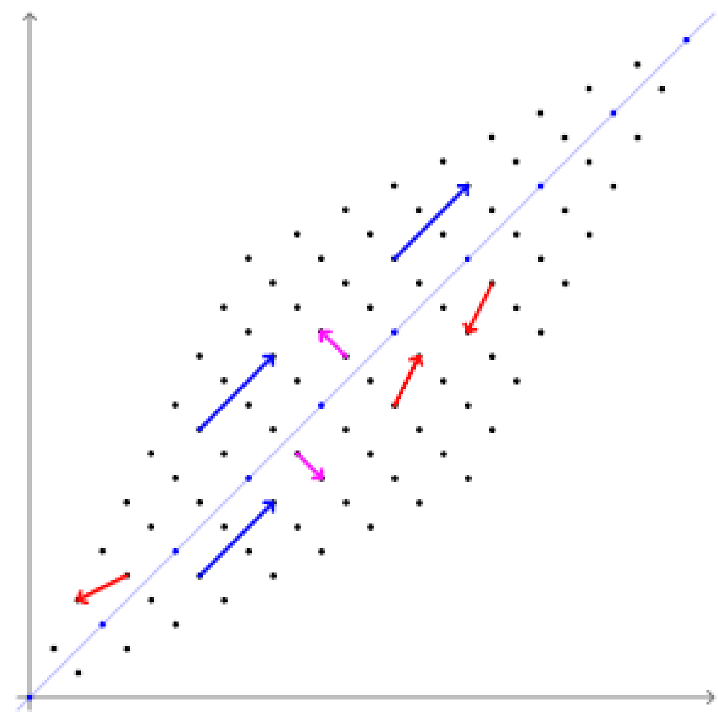
Principal Component Analysis

- Large dimension of the feature space reduces the efficiency of pattern recognition systems.
- The correlation between features does not allow a simple selection of features with smaller rate of recognition (classification) errors
- Principal Component Analysis (PCA) converts set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables

Eigenvectors and Eigenvalues

- An eigenvector of a square matrix is a non-zero vector that, when multiplied by the matrix, yields a vector that differs from the original at most by a multiplicative scalar.





Principal Component Analysis

- PCA is mathematically defined as an orthogonal linear transformation that transforms the data to a new coordinate system
- PCA is a powerful tool for analyzing data.

Applying PCA to Data

- Get data (Feature extraction)
- Subtract the mean (zero mean data)
- Calculate the covariance matrix for data
- Calculate the eigenvectors and eigenvalues of the covariance matrix
- Choose components and form a transform matrix
- Deriving the new data set

Sample Data

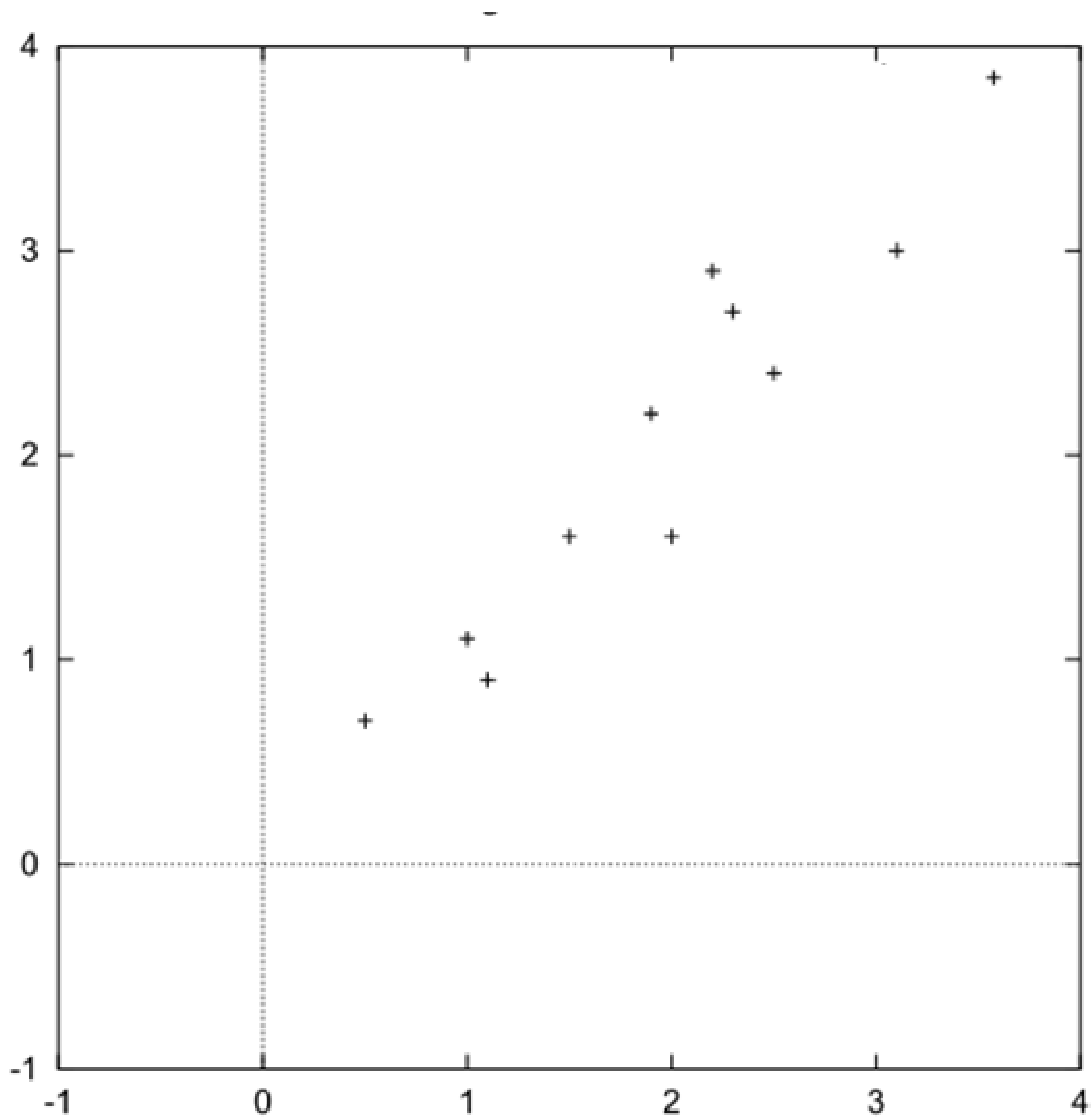
Data =

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

DataAdjust =

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

Plotting Data

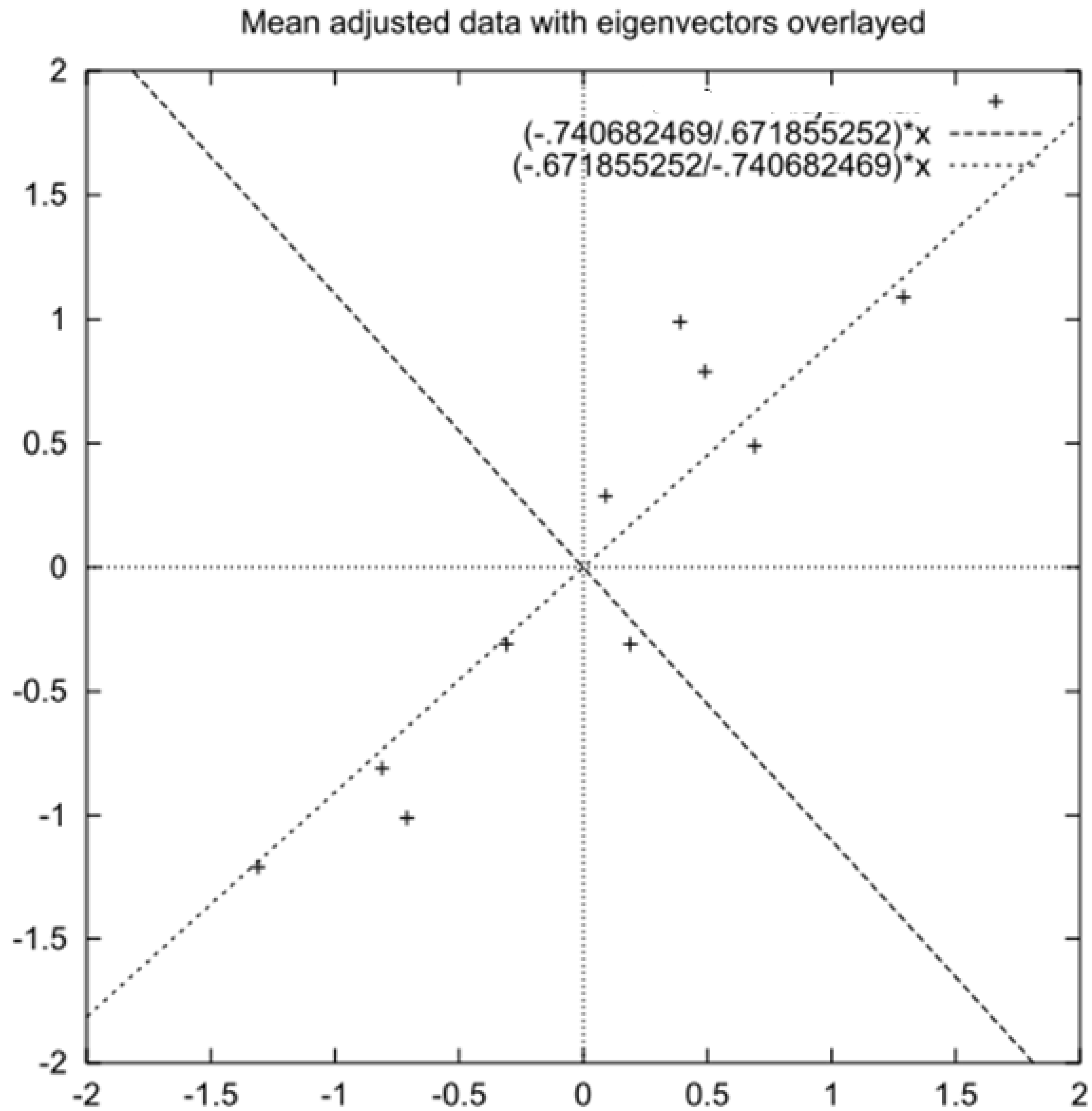


Covariance, Eigenvectors, and Eigenvalues

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$



Mean adjusted data with eigenvectors overlayed
(-.740682469/.671855252)*x
(-.671855252/-.740682469)*x
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

Forming the Transform Matrix

- Transform Matrix = (eig_1, eig_2, …, eig_n)

$$\begin{pmatrix} -.677873399 & -.735178656 \\ -.735178656 & .677873399 \end{pmatrix}$$

$$\begin{pmatrix} -.677873399 \\ -.735178656 \end{pmatrix}$$

Transform Data

- Obtain the final data by:

Transformed_Data = Transform_Matrix x Zero_Mean_Data

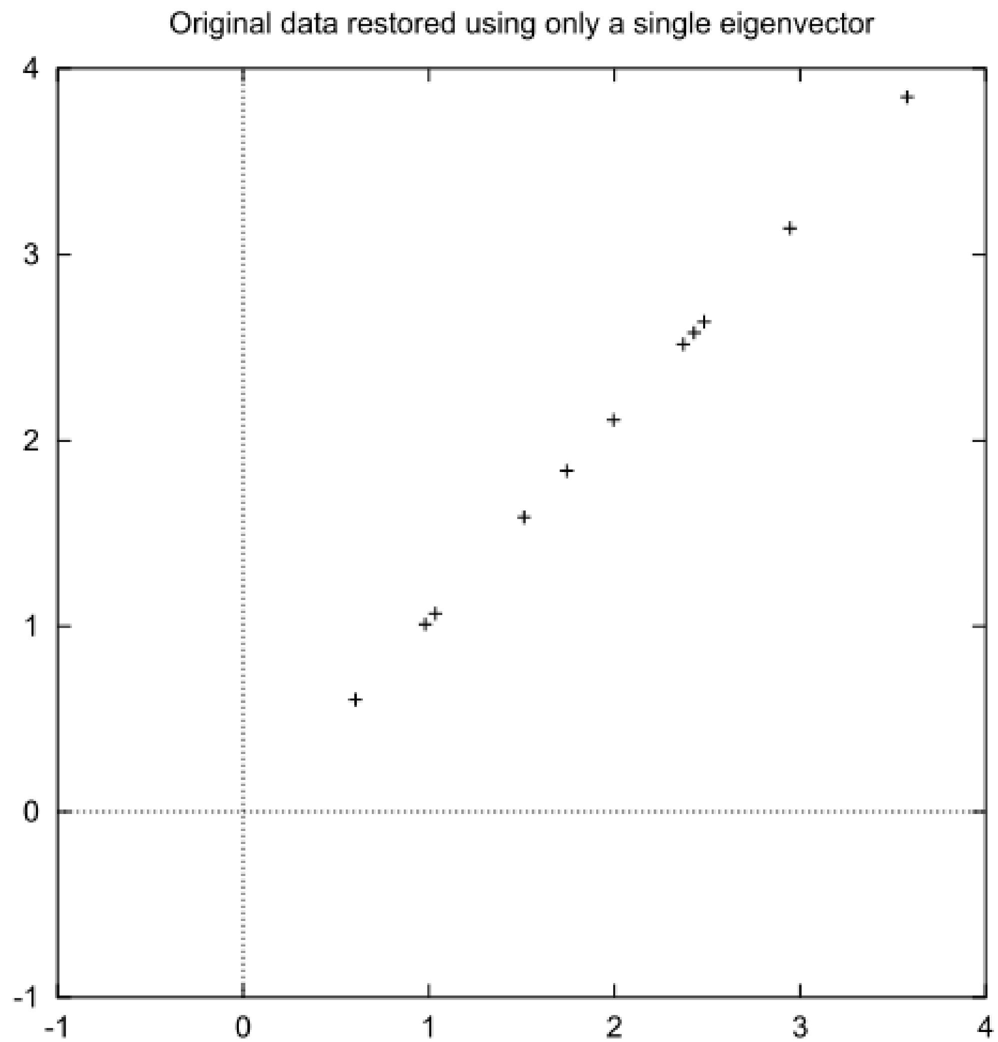
Transformed Data=

x	y
-.827970186	-.175115307
1.77758033	.142857227
-.992197494	.384374989
-.274210416	.130417207
-1.67580142	-.209498461
-.912949103	.175282444
.0991094375	-.349824698
1.14457216	.0464172582
.438046137	.0177646297
1.22382056	-.162675287

Data transformed with 2 eigenvectors

Transformed Data (Single eigenvector)

x
-.827970186
1.77758033
-.992197494
-.274210416
-1.67580142
-.912949103
.0991094375
1.14457216
.438046137
1.22382056



Questions?