this action is simply a dot product of the vector of w's (the *weights*) to the signal x. For the purposes of this section, we will assume no further processing.

If the signal is passed to a *cell assembly*, or group of neurons, then the overall action of the group is simply a linear mapping:

$$\mathbf{x} \to W\mathbf{x} = \mathbf{y}$$

If we have k neurons and $\mathbf{x}$ is a vector in $\mathbb{R}^n$, then W is a $k \times n$ matrix, and each row corresponds to a signal neuron's weights (that is, W_{ij} refers to the weight taking x_j to neuron i).

A further assumption will be to specify the mathematical realization of Hebb's postulate. Suppose that we present the network with a pattern, $\mathbf{x} \in \mathbb{R}^n$, and it outputs a pattern, $\mathbf{y} \in \mathbb{R}^k$. In terms of each weight, we see that

$$\boxed{W_{ij} \text{ connects the } j^{\text{th}} \text{ value of the input to the } i^{\text{th}} \text{ value of the output.}}$$

Thus we might take the following as Hebb's Rule. The change in the weight connecting the j^{th} input to the i^{th} cell is given by:

$$\Delta W_{ij} = \alpha y_i x_j$$

where α is called **the learning rate**. If both x_j and y_i match in sign, then W_{ij} becomes larger. If there is a mismatch in sign, W_{ij} gets smaller. This is the *unsupervised Hebbian rule.* Let's take a closer look at this using some linear algebra. Let us suppose we have n inputs to the network, and m outputs. Then W is $m \times n$, with $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^m$.

Note that if we compute the outer product, $\mathbf{y}\mathbf{x}^T$, we get:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} [x_1, x_2, \ldots, x_n] = \begin{bmatrix} y_1x_1 & y_1x_2 & \ldots & y_1x_n \\ y_2x_1 & y_2x_2 & \ldots & y_2x_n \\ \vdots & & & \vdots \\ y_mx_1 & y_mx_2 \ldots & y_mx_n & \end{bmatrix}$$

You should verify that in this case, we can compactly write Hebb's rule as:

$$W_{\text{new}} = W + \alpha\mathbf{y}\mathbf{x}^T$$

and that this change is valid for a single $\mathbf{x}, \mathbf{y}$ stimulus-response pair. If we define W_0 to be the initial weight matrix (we could initialize it randomly, for example), then the update rule becomes:

$$W_1 = W_0 + \alpha\mathbf{y}\mathbf{x}^T = W_0 + \alpha W_0\mathbf{x}\mathbf{x}^T = W_0\left(I_{n\times n} + \alpha\mathbf{x}\mathbf{x}^T\right)$$

EXERCISES:

1. Write W_n in terms of W_0 using the previous formula as a starting point.
2. Show that, if λ_i, $\mathbf{v}_i$ is the eigenvalue and eigenvector of a matrix A, then $(1 + \beta\lambda_i)$ and $\mathbf{v}_i$ is an eigenvalue and eigenvector of $(I + \beta A)$.

3. Let the matrix $A = \mathbf{x}\mathbf{x}^T$, where $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{x} \neq 0$. If $\mathbf{v} \in \mathbb{R}^n$, show that $A\mathbf{v}$ is a scalar multiple of x. (This shows that the dimension of the columnspace of $\mathbf{x}\mathbf{x}^T$ is 1)

4. Same matrix A as in the previous exercise. Show that one eigenvalue is $\|x\|^2$ (Hint: The eigenvector is $\mathbf{x}$).

5. We'll state the following without proof for now: If A is $n \times n$ and symmetric, and the columnspace of A has dimension 1, then there is exactly one nonzero eigenvalue. Use this, together with the previous exercises to compute the eigenvalues of $I + \alpha\mathbf{x}\mathbf{x}^T$.

6. There is a theorem that says that if the eigenvalues satisfy $|\lambda_i| \leq 1$, then the elements of A^n will converge (otherwise, the elements of A^n will diverge).

   Given our previous computation, will Hebb's rule give convergence?

Somehow, we want to take feedback into account so that we can use Hebb's rule in supervised learning problems.

Let $\mathbf{t}$ be the target (or desired) value for the input $\mathbf{x}$. In the supervised Hebbian rule, we need to update the weights based on what we want the network to do, rather than what the network is already doing:

$$\Delta W_{ij} = \alpha t_j x_i$$

There is still something unsatisfying here- When should we stop the training? It seems like the weights could diverge as training progresses, and furthermore, we're not taking the actual outputs of the network into account. Heuristically, we would like for the update to go to zero as the target values approach the network outputs. This leads us to our final modification of our basic Hebb rule, and is called the Widrow-Hoff learning rule[5]:

$$\Delta W_{ij} = \alpha(t_j - y_j)x_i$$

If we put this in matrix form, the learning rule becomes:

$$W^{\text{new}} = W^{\text{old}} + \alpha\,(\mathbf{t} - \mathbf{y})\,\mathbf{x}^T$$

where $(\mathbf{x}, \mathbf{t})$ is a desired input-output relation, and $\mathbf{y} = W\mathbf{x}$.

Additionally, sometimes it is appropriate to add a bias term so that the network has the output:

$$\mathbf{y} = W\mathbf{x} + \mathbf{b}$$

The bias update is similar to the previous update:

$$\mathbf{b}^{\text{new}} = \mathbf{b}^{\text{old}} + \alpha\,(\mathbf{t} - \mathbf{y})$$

[5]Also goes by the names Least Mean Squares rule, and the delta rule.

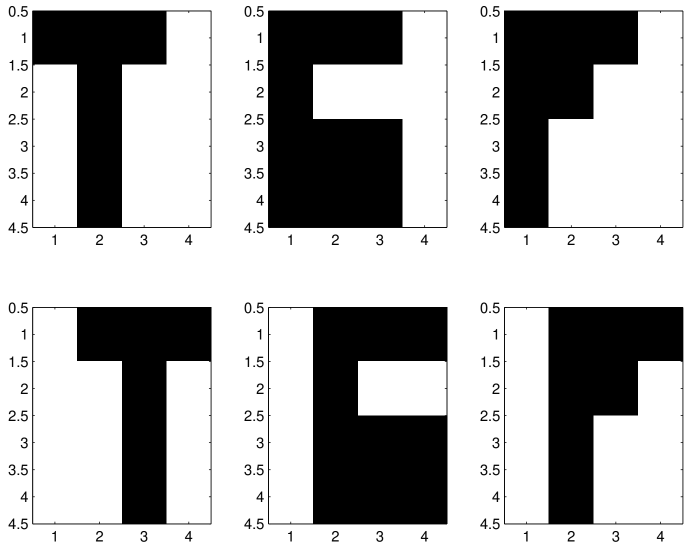

Figure 3.2: The inputs to our linear associative memory model: Three letters, T, G, H, where we have two samples of each letter, and each letter is defined by a 4×4 grid of numbers. We'll be associating T with -60, G with 0, and H with 60.

3.4.2 Example: Associative Memory

Here we will reproduce an experiment by Widrow and Hoff[6] who built an actual machine to do this (we'll do a computer simulation).

We'll have three letters as input, T, G and F. We'll associate these letters to the numbers $-60, 0, 60$ respectively. We want our network to perform the association using the Widrow-Hoff learning rule.

The letters will be defined by 4×4 arrays of numbers, where 1 corresponds to the color black, and -1 corresponds to the color white. In this example, we'll have two samples of each letter, as shown in Figure 3.2.

Implementation:

- First, we process the input data. Rather than working with 4×4 grids, we concatenate the columns to work with vectors in $\mathbb{R}^{16}$. Thus, we have 6 domain data points in $\mathbb{R}^{16}$, two samples of each letter. Construct range points so that they correspond with the letters.

- We'll use an $\alpha = 0.03$.

- We'll take several passes through all the data points.

[6] See "Adaptive Switching Circuits" by B. Widrow and M.E. Hoff, in 1960 IRE WESCON Convention Record, New York: IRE, Part 4, p. 96-104. You might find reprints also on the internet.

- To measure the error, after each pass through the data, we'll put each letter through the function to get an output value. We'll take the square of the difference between that and the desired value. We'll take the sum of the errors squared for those six samples to be the measure of the error for that pass.

Here is the code we used for this example. Again, be sure to read and understand what the code is doing. A lot of the initial part of the code is just there to get the data read in and plotted.

```
T1=[1 1 1 -1
    -1 1 -1 -1
    -1 1 -1 -1
    -1 1 -1 -1];
T2=[-1 1 1 1
    -1 -1 1 -1
    -1 -1 1 -1
    -1 -1 1 -1];
G1=[1 1 1 -1
    1 -1 -1 -1
    1 1 1 -1
    1 1 1 -1];
G2=[-1 1 1 1
    -1 1 -1 -1
    -1 1 1 1
    -1 1 1 1];
F1=[1 1 1 -1
    1 1 -1 -1
    1 -1 -1 -1
    1 -1 -1 -1];
F2=[-1 1 1 1
    -1 1 1 -1
    -1 1 -1 -1
    -1 1 -1 -1];

gg=colormap(gray);
gg=gg(end:-1:1,:);

subplot(2,3,1)
imagesc(T1)
colormap(gg)
subplot(2,3,2)
imagesc(G1)
subplot(2,3,3)
imagesc(F1)

subplot(2,3,4)
```

```
imagesc(T2)
subplot(2,3,5)
imagesc(G2)
subplot(2,3,6)
imagesc(F2)

%*************************************
%The main code starts here!
%*************************************

X=[T1(:) T2(:) G1(:) G2(:) F1(:) F2(:)];
T=[60 60 0 0 -60 -60];
alpha=0.03;
W=randn(1,16);
b=0;
TotalErr=0;
NumPoints=6;

for k=1:60
    idx=randperm(6);

    for j=1:NumPoints
        ThisOut=W*X(:,idx(j))+b;
        ThisErr=T(idx(j))-ThisOut;
    %Update the weights and biases
        W=W+alpha*ThisErr*X(:,idx(j))';
        b=b+alpha*ThisErr;
    end
    EpochErr(k)=norm((W*X+b*ones(1,6))-T);
end
figure(2)
plot(EpochErr);
```

The plot of the error is shown in Figure 3.3. The horizontal axis counts the number of passes through the data, and the vertical axis gives the sum of the squared errors. Note that after 60 passes, we get very good classification of the letters!

You should put this code into Matlab and reproduce the figures.

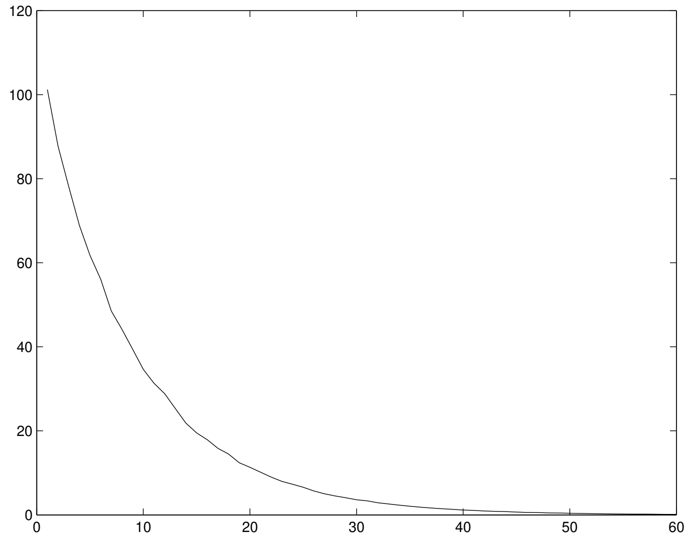

Figure 3.3: The errors for the Widrow-Hoff rule applied to letter recognition (or associative memory. After 60 passes through the data, the associations are very good.

3.4.3 Exercise

Let's put all of this together to solve a pattern classification problem. Suppose we are given the following associations:

Point	Class
$(1,1)$	1
$(1,2)$	1
$(2,-1)$	2
$(2,0)$	2
$(-1,2)$	3
$(-2,1)$	3
$(-1,-1)$	4
$(-2,-2)$	4

Graphically, we can see the classes in the plane in Figure 3.4. There are several ways of performing the desired mapping- for example, the outputs could be $1, 2, 3, 4$. But this may have unintended consequences. In this case, the metric between outputs would imply that Class 4 is much farther away from Class 1 than Class 3. A better method may be to take Class 1 to be the vector $[-1,-1]^T$, Class 2 is the vector $[-1,1]^T$, Class 3 is $[1,-1]^T$, and Class 4 is $[1,1]^T$. Now the 4 classes are on the vertices of a square.

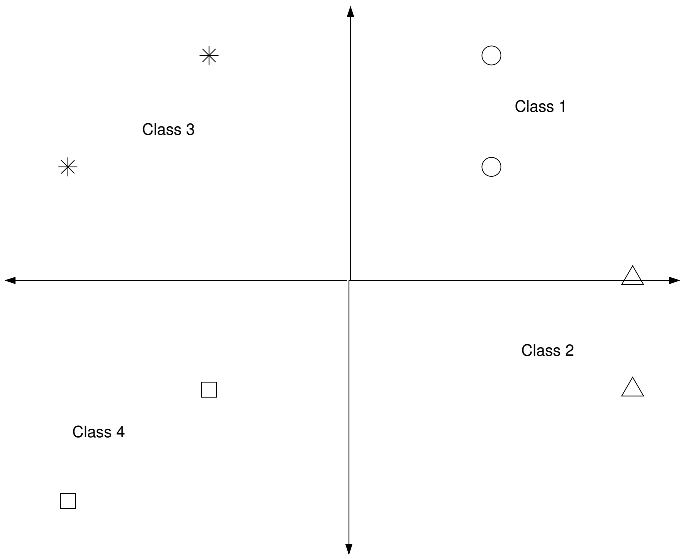

Figure 3.4: Pattern Classification Problem. Each point is a sample of one of the four classes.

Now for the details of the program. First write the inputs as an 2×8 matrix, with a corresponding output matrix that is also 2×8. Parameters that can be placed first will be the maximum number of times through the data N and the learning rate, a, which we will set to 0.04. We can also set an error bound so that we might stop early. Set the initial weights to the 2×2 identity, and the bias vector b to $[1, 1]^T$.

Check your algorithm using the following data: After using the first data point, W and b should have the values:

$$W = \begin{bmatrix} 0.88 & -0.12 \\ -0.12 & 0.88 \end{bmatrix}, \quad b = \begin{bmatrix} 0.88 \\ 0.88 \end{bmatrix}$$

Let the program run until you think it has converged (this is your choice). Then, plot the points $\mathbf{x} = [x, y]^T$ so that $W\mathbf{x} + \mathbf{b} = \mathbf{0}$, which will be the two lines:

$$W_{11}x + W_{12}y + b_1 = 0, \quad W_{21}x + W_{22}y + b_2 = 0$$

These lines form what is called the *decision boundary.*

Chapter 4

Statistics

In this course, we will be examining large data sets coming from different sources. There are two basic types of data:

- Deterministic Data: Data coming from a specific function- each point has a unique image. If we know the function, we can predict exactly the data terms that we see.

- Probabilistic Data: This data cannot be predicted exactly; with similar inputs, we might find very different outputs. The data is *random* in some sense. In this case, we can only hope to model some characteristics of the data, and not the exact data points themselves. We saw this earlier in the $N-$armed bandit problem, where we saw, with each pull of the arm, we would get different outputs. Thus, we were interested in extracting general information about the processes- the expected values, or mean payoff of each machine.

In data analysis, most data falls in between the two classes- that is, data from most sources involves a deterministic part, and a random part (perhaps from measurement errors).

We will not do a comprehensive listing of statistical procedures here. Instead, we introduce this material so that we have some statistical language to work with later on.

We enter this section with some words of caution: Our notation may be different than the notation you'll see in statistics books. For example, we should use different notation for the *sample* mean and variance versus the *population* mean and variance. However, we'll use the measures sparingly and we won't be getting too far into the statistics; therefore, we'll feel free to be informal with our notation.

4.1 Functions that Define Data

The basic way of defining non-deterministic data is through the use of a *probability density function*, or p.d.f. Rather than defining a function in terms of inputs and outputs, a p.d.f. defines the probability of certain events occurring.

To be more specific, a function $f(x)$ is said to be a probability density function if it satisfies the following conditions:

1. $f(x)$ is always non-negative.
2. $\int_{-\infty}^{\infty} f(x)\, dx = 1$
3. The probability of an event between values $x = a$ and $x = b$ is given by:

$$\Pr(a \le X \le b) = \int_a^b f(x)\, dx$$

From the definition, we see that the probability of any specific number is zero. Furthermore, in practice we won't be dealing with continuous functions (although we might try modeling them); rather, we will be looking at discrete intervals.

Therefore, suppose that our function f is non-zero on a finite interval (another way to say this is that f has bounded support). Then we can break the interval (or intervals) up into subintervals, and consider the probability of data occurring in each of these. In this case, the p.d.f. is discrete, and our definition changes somewhat:

A discrete p.d.f. will be a finite set of numbers, $\{P_1, P_2, \ldots, P_k\}$, so that:

1. P_i is non-negative, for all i.
2. $\sum_{i=1}^{k} P_k = 1$
3. The probability of a data value occurring in subinterval i (or bin i) is P_i.

Note that in this case, each of our "events" (data in subinterval i) are disjoint, as the probability of landing on an endpoint is zero. In the $N-$armed bandit problem, we had some experience with these- in that case, $P_i = \pi(i)$, which was the probability of choosing machine i.

In Matlab, we visualize a PDF via the `hist` command, which produces a histogram of the input. Let's take a look at some template probability distributions:

1. **Example 1: The Uniform Distribution**

   - The Continuous Version:

$$f(x) = \begin{cases} \frac{1}{b-a}, & \text{if } a \le x \le b \\ 0, & \text{otherwise} \end{cases}$$

- The Discrete Version (using N bins over the same interval):

$$\Pr\left(a+(i-1)\frac{b-a}{N} \le x \le a+i\frac{b-a}{N}\right) = \frac{1}{N} = P_i, i=1,2,\ldots,N.$$

- In Matlab, to obtain a value from a uniform distribution over $[0,1]$, we type $x =$ `rand`

2. **Example 2: The Normal (or Gaussian) Distribution**

   - The Continuous Version: The Normal distribution with mean μ and variance σ^2 (to be defined shortly) is defined as:

   $$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma}\exp\left(\frac{-(x-\mu)^2}{\sigma^2}\right)$$

   This is the common "bell-shaped curve"; the constant in the front is needed to make the integral evaluate to 1. Note that the normal distribution with zero mean and variance 1 simplifies to:

   $$f(x) = \mathcal{N}(0,1) = \frac{1}{\sqrt{2\pi}}\,\mathrm{e}^{-x^2}$$

   - In Matlab, we can obtain values from a normal distribution with zero mean and unit variance by $x =$ `randn`.

3. Our last example we define only in the continuous case. It is the p.d.f. commonly used to model the human voice, and is called the double Laplacian distribution:

$$f(x) = \begin{cases} K\mathrm{e}^{-|x|}, & -1 \le x \le 1 \\ 0, & \text{otherwise} \end{cases}$$

In the exercises, you'll be asked to determine the value of K, and we'll also see how the shape of the Laplacian compares to a normal distribution.

4.1.1 The probability distribution function

The probability distribution function (a.k.a. distribution function, cumulative distribution function) is defined via the probability density function:

$$F(X) = \Pr(-\infty < X < x) = \int_{-\infty}^{x} f(t)\,dt$$

We note that:

- By the Fundamental Theorem of Calculus, part I, $F(x)$ is the antiderivative of $f(x)$.
- $F(x)$ is strictly increasing, going from a minimum of zero to a maximum of 1.

- We saw the discrete version of this when we were working with the $N-$armed bandit; we used the Matlab function `cumsum` to create the cumulative distribution.
- To minimize confusion between terms, we'll refer to $F(x)$ as the distribution function, or cumulative distribution function.

4.2 The Mean, Median, and Mode

The most basic way to characterize a data set is through one number- the mean (or median or mode). Let's define these terms, using the data $\{x_1, x_2, \ldots, x_m\}$:

- The *Sample Mean* is:

  $$\mu = \frac{1}{m}\sum_{k=1}^{m} x_k$$

  We're all familiar with this definition- it's just the average of the data points. We will call this the mean, even though there are some issues here- The definition of the *population mean* requires knowledge of the underlying p.d.f., which we seldom have. In that case, the definition (a.k.a. the Expected value) is given by:

  $$\mu = E(X) = \sum_{\text{all x}} x f(x)$$

  You might remember this formula by seeing that this is really a weighted average, with those events most probable getting a higher weight than events less probable.

  Also, if your data is being drawn independently from a fixed p.d.f., then the sample mean will converge to the population mean, as the number of samples gets very large.

  Suppose we have m vectors in $\mathbb{R}^n$. we can similarly define the mean, just replace the scalar x_k with the k^{th} vector:

  $$\boldsymbol{\mu} = \frac{1}{m}\sum_{k=1}^{m} \boldsymbol{x}^{(k)}$$

  In Matlab, the mean is a built-in function. The command is

  ```
  m=mean(X);
  ```

  The output m depends on what X is:

  - If X is a vector, m is the scalar representing the mean.
  - If X is a matrix, m is a row vector, so that $m(i)$ is the mean of column i.

It will be very common for us to consider p data points in $\mathbb{R}^n$ as a $p \times n$ matrix, and we will want to mean subtract each of those data vectors. This is accomplished by two lines of Matlab code, which is going to be something you'll want to remember!

```
%X is a pxn matrix of data, p points in n dimensions

m=mean(X);  %m is the mean in n dimensions
Xm=X-repmat(m,p,1);  %Use the repmat command!

%Xm is the mean-subtracted matrix of data
```

- The *Median* is a number so that exactly half the data is above that number, and half the data is below that number. Although the median does not have to be unique, we follow the definitions below if we are given a finite sample:

  If there are an odd number of data points, the median is the middle point. If there is an even number of data points, then there are two numbers in the middle- the median is the average of these.

  Although not used terribly often, Matlab will perform the median as well as the mean:

  ```
  m=median(X);
  ```

  where the output is a scalar if X is a vector, or a row vector if X is a matrix.

- The *Mode* is the value taken the most number of times. In the case of ties, the data is multi-modal.

We'll compare these definitions in the Exercises.

4.3 The Variance and Standard Deviation

The number that is used to describe the spread of the data about its mean is the *variance*:

Let $\{x_1, \ldots, x_m\}$ be as defined above. Then the *Sample Variance* is:

$$\sigma^2 = \frac{1}{m}\sum_{k=1}^{m}(x_k - \mu)^2$$

where μ is the mean of the data. If we think of the data as having zero mean, and placing each data point in a vector of length m, then this formula becomes:

$$\sigma^2 = \frac{1}{m}\|\boldsymbol{x}\|^2$$

NOTE: Some writers define the variance by using $\frac{1}{m-1}$ rather than $\frac{1}{m}$ (for example, see the discussion in [?]). We use the definition above, as it will be consistent with our other uses of the variance.

While we're defining the terms, we should distinguish between the sample variance and the actual population variance (as we did in the mean). The population variance is defined as the expected value of $(x-\mu)^2$,

$$E((x-\mu)^2) = \sum_{\text{all x}} (x-\mu)^2 f(x)$$

However, we seldom know the population variance in practice, so we will only use the sample variance.

The *Standard Deviation* is the square root of the variance, so the standard deviation is σ.

Let's take some template data to look at what the variance (and standard deviation) measure: Consider the data:

$$-\frac{2}{n}, -\frac{1}{n}, 0, \frac{1}{n}, \frac{2}{n}$$

If n is large, our data is tightly packed together about the mean, 0. If n is small, the data are spread out. The variance of this sample is:

$$\sigma^2 = \frac{1}{5}\left(\frac{4+1+0+1+4}{n^2}\right) = \frac{2}{n^2}$$

so that the standard deviation is:

$$\sigma = \frac{\sqrt{2}}{n}$$

and this is in agreement with our heuristic: If n is large, our data is tightly packed about the mean, and the standard deviation is small. If n is small, our data is loosely distributed about the mean, and the standard deviation is large. Another way to look at the standard deviation is in linear algebra terms: If the data is put into a vector of length m (call it $\boldsymbol{x}$), then the standard deviation is:

$$\sigma = \frac{\|\boldsymbol{x}\|}{\sqrt{m}}$$

4.3.1 Covariance and Correlation Coefficients

If we have two data sets, sometimes we would like to compare them to see how they relate to each other.

Definition: Let $X = \{x_1, \ldots, x_m\}, Y = \{y_1, \ldots, y_m\}$ be two data sets with means μ_x, μ_y respectively. Then the *covariance* of the data sets is given by:

$$\mathrm{Cov}(X,Y) = \sigma_{xy}^2 = \frac{1}{m}\sum_{k=1}^{m}(x_k-\mu_x)(y_k-\mu_y)$$

There are exercises at the end of the chapter that will reinforce the notation and give you some methods for manipulating the covariance. In the meantime, it is easy to remember this formula if you think of the following:

If X and Y have mean zero, and we think of X and Y as vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, then the covariance is just the dot product between the vectors, divided by m:

$$\operatorname{Cov}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{m}\boldsymbol{x}^T\boldsymbol{y}$$

We can then interpret what it means for X, Y to have a covariance of zero: $\boldsymbol{x}$ is orthogonal to $\boldsymbol{y}$. Continuing with this analogy, if we normalized by the size of $\boldsymbol{x}$ and the size of $\boldsymbol{y}$, we'd get the cosine of the angle between them. This is the definition of the correlation coefficient, and gives the relationship between the covariance and correlation coefficient:

Definition: The *correlation coefficient* between x and y is given by:

$$\rho_{xy} = \frac{\sigma_{xy}^2}{\sigma_x \sigma_y} = \frac{\sum_{k=1}^m (x_k - \mu_x)(y_k - \mu_y)}{\sqrt{\sum_{k=1}^m (x_k - \mu_x)^2 \cdot \sum_{k=1}^m (y_k - \mu_y)^2}}$$

Again, thinking of X, Y as having zero mean and placing the data in vectors $\boldsymbol{x}, \boldsymbol{y}$, then this formula becomes:

$$\rho_{xy} = \frac{\boldsymbol{x}^T\boldsymbol{y}}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|} = \cos(\theta)$$

This works out so nicely because we have a $\frac{1}{m}$ in both the numerator and denominator, so they cancel each other out.

We also see immediately that ρ_{xy} can only take on the real numbers between -1 and 1. Some interesting values of ρ_{xy}:

If ρ_{xy} is:	Then the data is:
1	Perfectly correlated ($\theta = 0$)
0	Uncorrelated ($\theta = \frac{\pi}{2}$)
-1	Perfectly (negatively) correlated ($\theta = \pi$)

One last comment before we leave this section: The Covariance and Correlation Coefficient only look for *linear* relationships between data sets!

For example, we know that $\sin(x)$ and $\cos(x)$ (as functions, or as data points sampled at equally spaced intervals) will be uncorrelated, but, because $\sin^2(x) + \cos^2(x) = 1$, we see that $\sin^2(x)$ and $\cos^2(x)$ are perfectly correlated.

This difference is the difference between the words "correlated" and "statistically independent". Statistical independence (not defined here) and correlations are not the same thing! We will look at this difference closely in a later section.

4.4 The Covariance Matrix

If we have p data points in $\mathbb{R}^n$, we can think of the data as a $p \times n$ matrix. Let X denote the *mean-subtracted* data matrix (as we defined previously). A natural question to ask is then how the i^{th} and j^{th} dimensions (columns) covary- so we'll compute the covariance between the i, j columns to define:

$$\sigma_{ij}^2 = \frac{1}{p} \sum_{k=1}^{p} X(k,i) \cdot X(k,j)$$

Computing this for all i, j will result in an $n \times n$ symmetric matrix, C, for which:

$$C_{ij} = \sigma_{ij}^2$$

In the exercises, we have you show that we can conclude that C can be computed using the definition below:

Definition: Let X denote a matrix of data, so that, if X is $p \times n$, then we have p data points in $\mathbb{R}^n$. Furthermore, we assume that the data in X has been mean subtracted. Then the *covariance matrix* associated with X is given by:

$$C = \frac{1}{p} X^T X$$

In Matlab, it is easy to compute the covariance matrix. For your convenience, we repeat the mean-subtraction routine here:

```
%X is a pxn matrix of data:
[p,n]=size(X);
m = mean(X);
Xm = X-repmat(m,p,1);
C=(1/p)*X'*X;
```

Matlab also has a built-in covariance function. It will automatically do the mean-subtraction (which is a lot of extra work if you've already done it!).

```
C=cov(X);
```

If you forget which sizes Matlab uses, you might want to just compute the covariance yourself. It assumes, as we did, that the matrix is $p \times n$, and returns an $n \times n$ covariance- HOWEVER, it will divide by $p-1$ rather than by p. This is not a big issue for the applications we will be considering- you may use either method, but be aware that there are differences in the actual algorithms.

4.5 Exercises

1. By hand, compute the mean and variance of the following set of data:

$$1, 2, 9, 6, 3, 4, 3, 8, 4, 2$$

2. Obtain a sampling of 1000 points using the uniform distribution: and 1000 points using the normal distribution:

```
x=rand(1000,1);
y=randn(1000,1);
```

Compare the distributions using Matlab's *hist* command: `hist([x y],100)` and print the results. You'll note that the histograms have not been scaled so that the areas sum to 1, but we do get an indication of the nature of the data.

3. Compute the value of K in the double Laplacian function so that f is a p.d.f.

4. Next, load a sample of human voice: `load laughter` If you type `whos`, you'll see that you have a vector y with the sound data. The computers in the lab do have sound cards, but they don't work very well with Matlab, so we won't listen to the sample. Before continuing, you might be curious about what the data in y looks like, so feel free to plot it. We want to look at the distribution of the data in the vector y, and compare it to the normal distribution. The mean of y is already approximately zero, but to get a good comparison, we'll take a normal distribution with the same variance:

```
clear
load laughter
whos
sound(y,Fy);  %This only works if there's a good sound card
s=std(y);
x=s*randn(size(y));
hist([x y],100);  %Blue is "normal", Red is Voice
```

Print the result. Note that the normal distribution is much flatter than the distribution of the voice signal.

5. Compute the covariance between the following data sets:

x	-1.0	-0.7	-0.4	-0.1	0.2	0.5	0.8
y	-1.3	-0.7	-0.1	0.5	1.1	1.7	2.3

(4.1)

6. Let $\mathbf{x}$ be a vector of data with mean μ, and let a, b be scalars. What is the mean of $a\mathbf{x}$? What is the mean of $\mathbf{x}+b$? What is the mean of $a\mathbf{x}+b$?

7. Let $\mathbf{x}$ be a vector of data with variance σ^2, and let a, b be scalars. What is the variance of $a\mathbf{x}$? What is the variance of $\mathbf{x}+b$? What is the variance of $a\mathbf{x}+b$?

8. Show that, for data in vectors $\mathbf{x}$, $\mathbf{y}$ and a real scalar a,
$$\mathrm{Cov}(ax, y) = a\mathrm{Cov}(x, y) \quad \mathrm{Cov}(x, by) = b\mathrm{Cov}(x, y)$$

9. Show that, for data in $\mathbf{x}$ and a vector consisting only of the scalar a,
$$\mathrm{Cov}(x, a) = 0$$

10. Show that, for a and b fixed scalars, and data in vectors $\mathbf{x}, \mathbf{y}$,
$$\mathrm{Cov}(x + a, y + b) = \mathrm{Cov}(x, y)$$

11. If the data sets X and Y are the same, what is the covariance? What is the correlation coefficient? What if $Y = mX$? What if $Y = mX + b$?

12. Let X be a $p \times n$ matrix of data, where we n columns of p data points (you may assume each column has zero mean). Show that the $(i, j)^{\text{th}}$ entry of $\dfrac{1}{p}X^TX$ is the covariance between the i^{th} and j^{th} columns of X. HINT: It might be convenient to write X in terms of its columns,
$$X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$$
Also show that $\frac{1}{p}X^TX$ is a symmetric matrix.

13. This exercise shows us that our geometric insight might not extend to high dimensional space. We examine how points are distributed in high dimensional hypercubes and unit balls. Before we begin, let us agree that a hypercube of dimension n has the edges:
$$(\pm 1, \pm 1, \pm 1, \ldots, \pm 1)^T$$
so, for example, a 2-d hypercube (a square) has edges:
$$(1, 1)^T, (-1, 1)^T, (1, -1)^T, (-1, -1)^T$$
   (a) Show that the distance (standard Euclidean) from the origin to a corner of a hypercube of dimension d is $\sqrt{d}$. What does this imply about the shape of the "cube", as $d \to \infty$?
   (b) The volume of a $d-$dimensional hypersphere of radius a can be written as:
$$V_d = \frac{S_d a^d}{d}$$
   where S_d is the $d-$dimensional surface area of the unit sphere.
   First, compute the volume between hyperspheres of radius a and radius $a - \epsilon$.
   Next, show that the ratio of this volume to the full volume is given by:
$$1 - \left(1 - \frac{\epsilon}{a}\right)^d$$
   What happens as $d \to \infty$?
   If we have 100,000 data points "uniformly distributed" in a hypersphere of dimension 10,000, where are "most" of the points?

4.6 Linear Regression

In this section, we examine the simplest case of fitting data to a function. We are given two sets of data-

$$X = \{x_1, x_2, \ldots, x_n\} \quad Y = \{y_1, y_2, \ldots, y_n\}$$

We wish to find the best linear relationship between X and Y. But what is "best"? It depends on how you look at the data, as described in the next three exercises.

1. **Exercise:** Let y be a function of x. Then we are trying to find m and b so that

$$y = mx + b$$

best describes the data. If the data were perfectly linear, then this would mean that:

$$\begin{aligned} y_1 &= mx_1 + b \\ y_2 &= mx_2 + b \\ &\vdots \\ y_n &= mx_n + b \end{aligned}$$

However, most of the time the data is not actually, *exactly* linear, so that the values of y don't match the line: $mx + b$. Thus we have an error:

$$E_1 = \sum_{k=1}^{n} |y_k - (mx_k + b)|$$

(a) Show graphically what this error would represent for one of the data points.

(b) E_1 is a function of two variables, m and b. What is the difficulty in determining the minimum[1] error using this error function?

(c) Define the squared error as:

$$E_{\mathrm{se}} = \sum_{k=1}^{n} (y_k - (mx_k + b))^2$$

Why is it appropriate to use this error instead of the other error?

(d) E_{se} is a function of m and b, so the minimum value occurs where

$$\frac{\partial E_{\mathrm{se}}}{\partial m} = 0 \quad \frac{\partial E_{\mathrm{se}}}{\partial b} = 0$$

[1] We could solve this problem by using Linear Programming, but that is outside the scope of this text.

Show that this leads to the system of equations: (the summation index is 1 to n)

$$\begin{aligned} m\sum x_k^2 + b\sum x_k &= \sum x_k y_k \\ m\sum x_k + bn &= \sum y_k \end{aligned}$$

(e) **Exercise:** Write a Matlab routine that will take a $2 \times n$ matrix of data, and output the values of m and b found above. The first line of code should be:

```
function [m,b]=Line1(X)
```

and save as `Line1.m`.

2. **Exercise:** If we treat x as a function of y (i.e., $x = f(y)$), how does that change the equations above? Draw a picture of what the error would represent in this case. Write a new Matlab routine `[m,b]=Line2(X)` to reflect these changes.

3. **Exercise:** The last case is where we treat x and y independently, so that we don't assume that one is a function of the other.

   (a) Show that, if $ax + by + c = 0$ is the equation of the line, then the distance from (x_1, y_1) to the line is

   $$\frac{|ax_1 + by_1 + c|}{\sqrt{a^2 + b^2}}$$

   which is the size of the orthogonal projection of the point to the line. This is actually problem 53, section 11.3 of Stewart's Calculus text, if you'd like more information.
   (HINT: The vector $[a, b]^T$ is orthogonal to the line $ax + by + c = 0$. Take an arbitrary point P on the line, and project an appropriate vector to $[a, b]^T$.)
   Conclude that the error function is:

   $$E = \sum_{k=1}^{n} \frac{(ax_k + by_k + c)^2}{a^2 + b^2}$$

   (b) Draw a picture of the error in this case, and compare it graphically to the error in the previous 2 exercises.

   (c) The optimimum value of E occurs where $\frac{\partial E}{\partial c} = 0$. Show that if we mean subtract X and Y, then we can take $c = 0$. This leaves only two variables.

   (d) Now our error function is:

   $$E = \sum_{k=1}^{n} \frac{(ax_k + by_k)^2}{a^2 + b^2}$$

Show that we can transform this function (with appropriate assumptions) to:

$$E = \sum_{k=1}^{n} \frac{(x_k + \mu y_k)^2}{1 + \mu^2}$$

(for some μ), and conclude that E is a function of one variable.

(e) Now the minimum occurs where $\frac{dE}{d\mu} = 0$. Compute this quantity to get:

$$\mu^2 A + \mu B + C = 0$$

where A, B, C are expressions in $\sum x_k y_k$, $\sum x_k^2$, $\sum y_k^2$. This is a quadratic expression in μ, which we can solve. Why are there (possibly) 2 real solutions?

(f) Write a Matlab routine `[a,b,c]=Line3(X)` that will input a $2 \times n$ matrix, and output the right values of a, b, and c.

4. **Exercise:** Try the 3 different approaches on the following data set, which represents heights (in inches) and weight (in lbs.) of 10 teenage boys. (Available in `HgtWgt.mat`)

X	69	65	71	73	68	63	70	67	69	70
Y	138	127	178	185	141	122	158	135	145	162

Plot the data with the 3 lines. What do the 3 approaches predict for the weight of someone that is 72 inches tall?

5. **Exercise:** Do the same as the last exercise, but now add the data point $(62, 250)$. Compare the new lines with the old. Did things change much?

6. **Matlab Note:** Consider using the command `subplot` to plot multiple graphs on the same figure. For example, try the following sequence of commands:

```
x=linspace(-8,8);
y1=sin(x);
y2=sin(2*x);
y3=sin(x.*x);
y4=sin(exp(-x));
subplot(2,2,1);
plot(x,y1);
subplot(2,2,2);
plot(x,y2);
subplot(2,2,3);
plot(x,y3);
subplot(2,2,4);
plot(x,y4);
```

4.7 The Median-Median Line:

The median of data is sometimes preferable to the mean, especially if there exists a few data points that are far different than "most" data.

1. **Definition:** The *median* of a data set is the value so that exactly half of the data is above that point, and half is below. If you have an odd number of points, the median is the "middle" point. If you have an even number, the median is the average of the two "middle" points. Matlab uses the `median` command.

2. **Exercise:** Compute (by hand, then check with Matlab) the medians of the following data:

   - $$1, 3, 5, 7, 9$$

   - $$1, 2, 4, 9, 8, 1$$

   The motivation for the median-median line is to have a procedure for line fitting that is not as sensitive to "outliers" as the 3 methods in the previous section.

   ### Median-Median Line Algorithm

   - Separate the data into 3 equal groups (or as equal as possible). Use the $x-$axis to sort the data.
   - Compute the median of each group (first, middle, last).
   - Compute the equation of the line through the first and last median points.
   - Find the vertical distance between the middle median point and the line.
   - Slide the line 1/3 of the distance to the middle median point.

3. **Exercise:** Understand the commands in the program `mmline`. There are several new matlab functions used. Below are some hints as to how we'll divide the data into three groups.

   - If we divide a number by three, we have three possible remainders: $0, 1, 2$. What is the most natural way of seperating data in these three cases (i.e., if we had 27, 28 or 29 data points)?
   - Look at the Matlab command `rem`. Notice that:

     `rem(27,3)=0   rem(28,3)=1   rem(29,3)=2`

- Look at the Matlab command `sort`. The full command looks like: `[s,index]=sort(x)` The output vector s will be x sorted. The vector `index` will contain which order the original indices were in. That is,

  `x(index)=s`

- We can therefore sort x first, then sort y according to the index for x.

4. **Exercise:** Try this algorithm on the last data set, then add the new data point. Did your lines change as much?

5. **Exercise:** Consider the following data set [**?**] which relates the index of exposure to radioactive contamination from Hanford to the number of cancer deaths per 100,000 residents. We would like to get a relationship between these data. Use the four techniques above, and compare your answers. Compute the actual errors for the first three types of fits and compare the numbers.

County/City	Index	Deaths
Umatilla	2.5	147
Morrow	2.6	130
Gilliam	3.4	130
Sherman	1.3	114
Wasco	1.6	138
Hood River	3.8	162
Portland	11.6	208
Columbia	6.4	178
Clatsop	8.3	210