Laboratory 2: Neuronal Pattern Recognition

Goals

1. To learn how a neuron can act as a pattern recognition and decision-making device;
2. To understand the meaning and execution of linear discrimination, i.e., how input weights and threshold interact to determine the set of patterns recognized by a neuron;
3. To explain in what sense a McCulloch-Pitts neuron generalizes;
4. To demonstrate some limitations of linear discrimination as performed by neurons.

In this lab, you will create three networks and assess their performance on pattern recognition problems. In the first network, a single postsynaptic neuron receives connections from three different presynaptic inputs. The second network consists of two postsynaptic neurons, which are able to recognize two different patterns. These networks recognize a pattern in terms of its membership in a particular category. The third network has three neurons that recognize three different patterns. In the last two cases, the input is two-dimensional. Finally, you will work with partially overlapping sets of input patterns. This set will be used to explore learn about the two types of errors that a neuron might make. As you will also see, different input environments lead to different levels of performance by the simulated network models.

Students who are new to linear algebra will need to refer to appendix 2.1 and 2.3 as necessary.

Introduction

Beginning with this chapter, this laboratory manual delineates programming experiences that lay a strong, intuitive foundation for understanding the neurobiological bases of cognition.

Summary of Preceding Lecture:

All the Neuroscience You Need to Know *(at least for the next three weeks in this course)*

In the neocortex there are two kinds of cells in the brain, nerve cells and glial cells. For this class, we will assume that neurons, i.e. nerve cells, are the only ones that process information.

There are two functionally distinguishable classes of neurons, inhibitory neurons and excitatory neurons.

The simplest generic neuron has three functional parts plus its synapses: a dendrite where it receives almost all of its inputs, a cell body (also called a soma) which, along with the dendrite, "adds" up the excitation created by active inputs, and an axon which is the output portion of the neuron. A dendrite can receive many inputs. An axon can send information to many neurons.

An axon is a long-distance, binary communication channel. It carries brief, discrete electrical events called action potentials. An action potential is initially generated where the axon arises from the cell body. Once initiated, action potential conduction will be assumed perfect and non-failing. Because a conducted action potential (i.e., network update cycles) is an all or none

phenomenon, and because our simplified neuron has no memory across time, we can think of the axon as being instantaneously valued by its action potential status, i.e., zero or one, {0,1}, where zero means no action potential during the current computational interval and one means there is an action potential at this time. From our computational perspective it takes one timestep for a neuron to add up its inputs and transmit an action potential to another neuron.

Information is transmitted from one neuron to another at a synapse. Moreover, each synapse is made up of just two neurons, where one neuron is termed presynaptic (part of an axon) and the other part is termed postsynaptic (part of a dendrite). For the purpose of this course, assume that all such axo-dendritic synapses are excitatory.

The primary information processors in the cerebral cortex are the excitatory neurons which outnumber inhibitory neurons seven to one. A neuron is designated as excitatory or inhibitory as a function of its output synapses. That is, we assume all output synapses of a single neuron are of one type, either excitatory or inhibitory and this defines the neuron's class.

A single nerve cell receives many synaptic excitations and inhibitions during any given functional time-interval, and these inputs are combined into a single, scalar value. In general, we speak of neurons as mediating temporal and spatial 'integration' to produce its net level of excitation. By this we mean that a neuron adds up its inputs over some small time period (its time constant) and over the space of its dendrite. If the net excitation is great enough, the cell body (which we picture as the final site of neuronal integration) may reach or exceed a voltage called threshold. When a neuron's somatic (cell body) voltage is raised up to or above this threshold, the neuron emits an action potential that travels down its axon. Voltages below threshold cause a zero as output and nothing is sent by the axon.

In sum, information, as a zero or one, flows from a cell body and down an axon, which is the presynaptic partner to perhaps many synapses. Information flows across each active synapse from the presynaptic side to the postsynaptic side. This entire process introduces a temporal lag that is misleadingly called the synaptic delay (in fact, at 37° C the axonal conduction delay is many times longer than true synaptic delays).

From a quantitative perspective, not all excitatory synapses are the same. That is, an active synapse exerts an effect that is a function of its synaptic strength, and different synapses have different strengths. A synonym for synaptic strength is synaptic weight. This variety of synaptic strengths is fundamental to the information processing abilities of any neural network. In fact, memories can be implanted in a neural network by altering synaptic weights.

Synapses are modified in a way that depends on their activity history. Most importantly, modification is a function of the correlated history of activity of the two cells that form each synapse. Thus, synaptic modification is ultimately a localized chemical reaction.

Finally, inhibition can be viewed either as a subtractive or a divisive process. In either case, inhibition opposes excitation; the more inhibition a neuron receives, the greater is the required excitation needed to fire that neuron.

A mini lecture: The fundamental model and our modeling philosophy

The Promissory Note:

The ultimate issues we engage are cognitive in nature. Such issues include:

1. Decision-making
2. Discrimination
3. Abstraction
4. Generalization
5. Learning and memory
6. Representation
7. Prediction and forecasting

As equivalent, or overlapping topics, these issues can also be described as (7) categorization, (8) clustering, and (9) concept formation. Such topics are in the domain of statistical pattern recognition at least as much belonging to cognitive science.

Overall, our strategy is to create an intuitive foundation that explains how biology can produce these aspects of cognition. Moreover, beyond intuition, these connections can be made quantitative. Thus, by virtue of being quantifiable, there is the possibility of interpreting the evolved neurobiology in a scientifically meaningful way, much the way physics became meaningful in the hands of Galileo and then Newton.

In order to take such a long-term goal seriously, we ask the student to submit to one assumption.

Credo: If a microscopic neurobiological process can perform an abstracted version of a defined cognitive process, then we will presume that this abstract function will be preserved even as the microscopic neurobiology is embedded into ever more complex neurobiological systems.

We also ask the student to tolerate a multiplicity of models (e.g., two). A basic idea in any science, and often in teaching too, is economy of explanation. Sometimes such economy leads to multiple models. Here we will examine models with binary outputs and continuous or binary inputs. So we ask the student to consider each perspective. The two perspectives are ultimately connected. The ultimate connection is that a sequence of binary inputs will be translated by a synapse or a neuron into continuously valued signals. The binary output neuron with continuously valued inputs will become evident in sections 2.5-2.8 and in homework 2.5.1, but is also invoked in the next subsection.

As in all successful scientific disciplines simplicity with appropriate explanatory power is our goal. The simplest neuron with an ability to made decisions, discrimination patterns, and to demonstrate a form of generalization is the *McCulloch-Pitts neuron*. This neuron model is able to recognize patterns in its environment based on the features of a given stimulus. Although

introduced well before any biophysical details of neuronal function existed, such neurons are still considered to embody the most fundamental properties of biologically-based information processing; specifically, a neuronal computation is a weighted summation of excitatory inputs that is converted to a binary, {0,1}, signal. The binary states represent the neuron firing, a one, or doing nothing, a zero.

Networks of such neurons can produce fairly sophisticated forms of cognition and can be used to model neural networks. Moreover, because of the simplicity of these neuron models, they give us a better understanding of how the brain can produce cognition.
Cognitive processes that are modeled using these neurons are *pattern recognition* and two, generic binary *decision-making*.

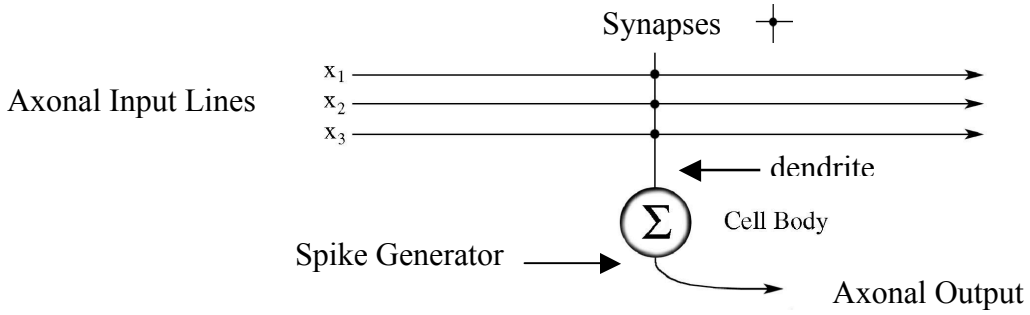

Figure 2.1. A single McCulloch-Pitts neuron with inputs. The input vector x(t) has three elements: $x_1(t)$, $x_2(t)$, and $x_3(t)$ that can be different on each timestep. Each synapse, a connection formed between a presynaptic axon (input line) and the single postsynaptic dendrite, is indicated by the darkened circular symbol •. The Σ indicates the summation of internal excitation, which is analogous to the altered polarization of neuronal cell body. The output of the postsynaptic neuron is another axon that transmits any action potentials of the postsynaptic neuron.

Decisions Based on Similarity Underlie Pattern Recognition by Neurons

There is a simple heuristic for predicting the future:

Similar causes, in similar contexts, produce similar effects.

This heuristic is not limited in its applicability—all brain regions implement this heuristic by using neurons that can measure and communicate similarity. In this laboratory, you'll learn the basic principles underlying similarity decisions by neurons. The principle is a qualitative description of the generic concept of generalization.

Similarity Judgment as a Problem in Geometry

Humans are willing to judge the similarity of two items where similarity is a continuous parameter. A simple way to access similarity between two patterns is to picture a Euclidean geometric space—two dimensions are good enough. In this geometric space, individual points, the simplest geometric structures, are specified at particular locations. For example, we can

specify a point V with location $\begin{bmatrix}1.6\\0.7\end{bmatrix}$, where each scalar element of this vector locates point V's projection onto each orthogonal axis of our two-dimensional space.

In the absence of any other knowledge, we shall say that two points are similar if they are close together in a Euclidean space. Likewise, two points are different if they are far apart. That is, the distance between two points is inversely related to similarity.

Such a geometric space is a quantitative way to represent features and patterns. As part of the whole, a feature can be a quantitative aspect of the total pattern. For example, if the pattern is a particular person, then quantitative features might include a person's weight, height, hair length, bank account, or grade point average. A feature can also be qualitative, valued zero or one, for the absence or presence of a feature (e.g., 'has a driver's license'). Here, because we want the simplest model that illustrates the most general principles, we will use non-negative values to quantify each feature; this value can correspond to each of a neuron's input, i.e., spike (feature present) or no spike (feature absent), or inverse interpulse interval if we want a rational positive value over some range such as [0,1).

An ordered set of such features is called a *feature vector, input vector,* or just *vector,* and such a vector is just like the vectors that we have already learned to create with MatLab in Laboratory 1. Thus, what we call a feature, a mathematician calls an element of a vector. Moreover, this same mathematician will be happy to represent a vector made up of two elements as a geometric point in a two-dimensional space, a vector with three elements as a point in three-dimensional space, and in general, an n-dimensional vector as a point in n-dimensional space.

Thus a vector is, at the same time, a point in a geometric space and an ordered set of scalar values. Because these values are ordered, a vector can be called a pattern. Indeed, anything that you would call a pattern can be expressed as a vector.

A single neuron, and even more so, an organized system of neurons called a *neural network*, operate on patterns in very high-dimensional spaces—e.g., one neuron in the neocortex might receive 5,000-20,000 excitatory inputs and thus process patterns of dimensions at 5,000-20,000. Moreover, each neuron is part of a network that processes information in an even higher dimensional space. For organized, functionally delimited brain region that can be called a neural network, the dimension arguably ranges from 500,000 to 5,000,000 or more. These are big numbers but, regardless of the size of these numbers, we still view a particular pattern with its quantified features as a vector—i.e., as a point in the appropriately dimensioned geometric space. Likewise, regardless of the dimensionality, the similarity of any two such n-dimensional points can be evaluated in terms of their proximity to one another in this n-dimensional space. Thus what you learn here about comparing two-dimensional vectors will generalize to more biologically appropriate dimensions.

Pattern Recognition and Decision-making

Pattern recognition and decision-making are ubiquitous cognitive processes. In this lab, you will demonstrate that individual neurons can make decisions. That is, a neuron can recognize a set of

features as a pattern. Importantly, you will begin to understand how the synapses made between individual presynaptic inputs and a postsynaptic neuron transform that input. You will also learn about the important role a neuron's *threshold* for action potential generation plays in determining a neuron's decision.

Assume for now that each neuron is dedicated to recognizing a family of closely related (i.e., similar) patterns, for example, a particular person viewed from a variety of angles and on different days are closely related patterns. For each such pattern, each active input to a postsynaptic neuron contributes to the excitation of that postsynaptic neuron. This synaptic contribution to the total excitation is in proportion to the synaptic strength or *weight*, which connects that presynaptic input to the postsynaptic cell. We will restrict our work to positive weights because these dominate in forebrain cortical systems. Furthermore, we will restrict all input values to the two values zero or one, {0,1}. Then if we look at a single postsynaptic neuron, neuronal decision-making is metaphorically a voting system in which each presynaptic input neuron casts its vote for the firing of the postsynaptic neuron, either by firing its own action potential or by not firing. The postsynaptic neuron tallies the votes over the set of inputs—a 'no' vote is a zero and a 'yes' vote has some positive value. If the tally is large enough, the postsynaptic neuron declares 'yes' and it too fires—if the tally is not large enough, the postsynaptic neuron decides 'no' and does not fire. However, because not all votes are equal, such a neuronal vote is not a true democracy. Instead, some 'yes' votes count more than others do. For example, an affirmative decision by a postsynaptic neuron may require only a relatively small number of 'yes' votes from its heavily weighted input features.

Why should some votes count for more? One reason is that some features are more reliably part of a particular family of patterns than other features.

Section 2.1. The computational elements of a McCulloch-Pitts neuron

In this lab, you'll use McCulloch-Pitts neurons to study pattern recognition by very small, feed-forward neural networks. Such a neuron functions in discrete time intervals and has no memory of its past excitation. In other words, here we will assume that successive input sets, and the postsynaptic decisions they drive, occur in discrete, sequential time intervals (an input set contains one or more input patterns). In this chapter, a postsynaptic neuron makes a decision at each time interval based solely on (*i*) its current input weights, (*ii*) its active inputs, and (*iii*) its threshold.

Such a neuron, including its inputs, is a parallel computational device. This is true for both McCulloch-Pitts neurons (see Fig. 2.1 for a schematic view of one such neuron) and more complicated neuron models. The net synaptic excitation of a McCulloch-Pitts neuron is a linear function of its inputs. That is, each synapse scales its axon's value, and then the postsynaptic neuron adds up these scaled values. Each postsynaptic neuron has its own distinct scaling for its summation process. Thus, this scaled summation converts the current set of active synapses into a scalar value termed the *internal excitation* or *net input excitation* (or just *excitation* for short). However, the output of such a McCulloch-Pitts neuron is either zero or one. To implement the crude binary encoding of the internal excitation, the neuron fires (output of one) when its net

input excitation exceeds a certain value called *threshold*. This threshold is the minimum value of the sum of the weighted active inputs needed for the postsynaptic neuron to fire. (Although we will often study neurons with binary, {0, 1}, outputs, the inputs to these neurons will often be continuously valued, [0, 1].)

It takes one computational cycle of a network simulation for the weighted summation of active inputs and the thresholding to occur. On the next computational cycle of the simulation, this process is repeated. Remember that a postsynaptic McCullough-Pitts neuron does not remember what happened on the last cycle. (We use the phrase *computational cycle* to describe computations by the simulated biological model, not elementary operations by your computer's CPU).

A parameterized McCulloch-Pitts neuron must include the following specifications:

- the list of its input axons
- the synaptic weight associated with each input axon
- a threshold value for output spike generation

Functionally, a McCulloch-Pitts neuron:

- scales the value of each active input ($x_i = 1$) via the connection weight at that synapse, then
- these scaled inputs are summed, and finally,
- a spike generator crudely encodes, as a zero or as a one, the value of this summation; often a spike generator encodes as zero or one (a prediction or retrodiction of category membership based on the input, but the computational operation is no different).

This process is repeated on each cycle, with each new input, and there is no memory of the excitations or firings of the previous cycles. Thus, because the McCulloch-Pitts neuron has no memory, we can concentrate our attention to the operations performed in a single timestep.

Section 2.2. Mathematizing and programming the McCulloch-Pitts neuron

We need to reduce such a McCulloch-Pitts neuron to a purely mathematical entity. This entails creating a variable corresponding to each of the functional parts just described.

A neuron *j* with **n** inputs *i* performs **n** multiplications and **n-1** additions at each discrete timestep. The multiplications are performed in parallel—each synapse multiplies the incoming excitation from each input by its weight. In each McCulloch-Pitts neuron *j*, these products are summed instantaneously producing the somatic value y_j at time *t*,

$$y(j,t) = \sum_i w(i,j)\,x(i,t).$$

At each soma there is a spike generator. This bridge between the soma and its axon then compares this summated excitation with its threshold value, and this comparison determines

whether or not the neuron fires. More exactly for *j*'s output activity at the end of the t[th] computational cycle,

$$z(j,t) = \begin{cases} 1 \text{ if } y(j,t) \geq \text{ threshold, but} \\ 0 \text{ otherwise.} \end{cases}$$

On the next timestep, the whole process begins again. Remember that a McCulloch-Pitts neuron forgets its summated internal excitation from one timestep to the next.

Note the agreement between the z and y indices in this description, particularly the agreement of time *t*. In this particular reduction of biology to computation, which in the hippocampus works at temporal resolutions more coarse than 10 milliseconds, the delays that reflect this increment of time are placed in axonal conduction times between neurons. This placement is only for the purposes of simulation. We could interpose the delay between the dendritic summation and the spike generator, but the results would be the same for our purposes.

Table 2.1. Notations

Typical	MatLab	English
$x_i(t)$	$x(i,t)$	the state of axon *i* at *t*
w_{ij}	$w(i,j)$	a fixed synaptic weight from *i* to *j*
$y_j(t)$	$y(j,t)$	dendrosomatic excitation of *j* at *t*
$z_j(t)$	$z(j,t)$	the output state of *j* at *t*

The typical indicial notation differs from the MatLab form. Instead of indicating the value of the i[th] input at time *t* as $x_i(t)$, here we will use MatLab's notation $x(i,t)$. That is, the inputs are a matrix with a row indicating a particular input axon and a particular column indicates a particular time point.

In a network of neurons, it also takes two indices to indicate the weight value of a particular synapse. Again, the convenient mathematical notation is w_{ij} but MatLab's notation $w(i,j)$ is just as good. Please note that our convention of *i* for a presynaptic axon and *j* for a postsynaptic dendrite is not universally accepted, so be careful about the notation when you look at published research articles.

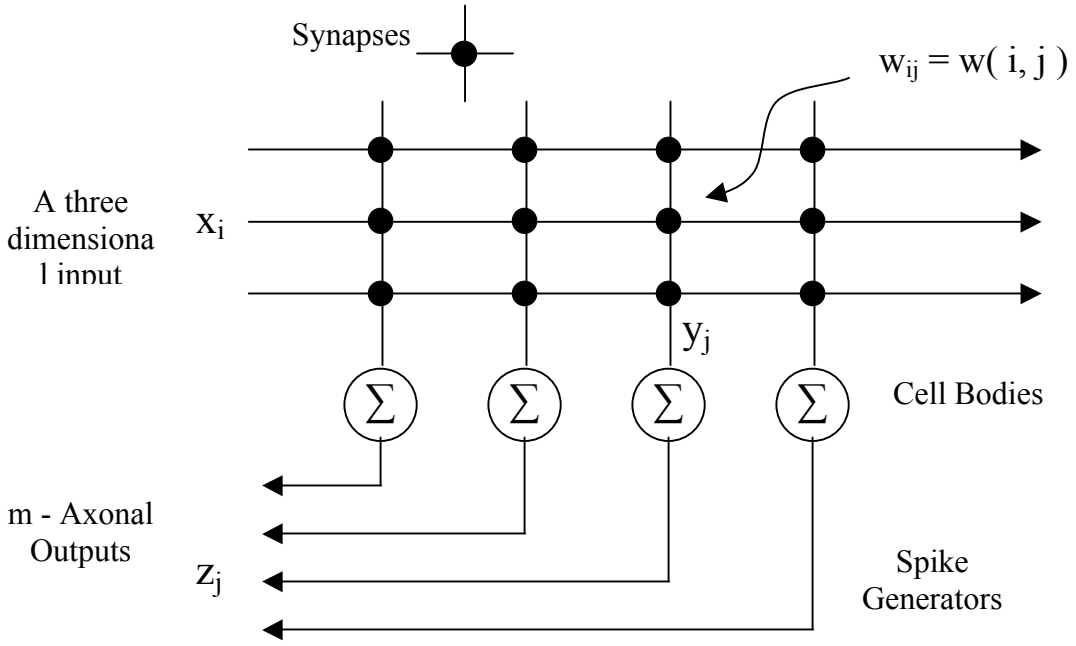

Figure 2.2. A 3x4 feedforward network of McCulloch-Pitts neurons. A three dimensional input vector excites a four dimensional set of postsynaptic neurons. For any one active axon, each pair-wise connection is scaled by the corresponding synaptic weight.

Section 2.3. Internal excitation—a linear computation

Let's look at a parameterized example of such a neuron.

Suppose there is a postsynaptic neuron *j* with three inputs, then we must specify a synaptic weight for each input, say $w = \begin{bmatrix} 0.2 \\ 0.3 \\ 0.4 \end{bmatrix}$, and we must specify a threshold, say 0.7. Suppose further that at some particular time point *t*, the inputs take on values $x(t) = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$.

Mathematically, for neuron *j*, we can express the sum of the inputs scaled by their respective synaptic weights at time *t*,

$$y(j,t) = \sum_{i=1}^{3} x_i(t) w(i,j) = 1 \cdot 0.2 + 0 \cdot 0.3 + 1 \cdot 0.4 = 0.2 + 0 + 0.4 = 0.6.$$

Then, we compare this somatodendritic excitation *y* to threshold (specifically, we note the inequality $0.6 < 0.7$) and conclude that the neuron does not fire, i.e., as output we write,

$z(j,t)$=0. Now suppose the next input, which by definition occurs at time $t+1$, is $x(t+1)=\begin{bmatrix}0\\1\\1\end{bmatrix}$. Then $y(j,t+1)=0.7$, and since threshold is 0.7, the neuron has just the minimum amount of excitation to fire; i.e., $z(j,t+1)=1$.

A similar example assumes inputs valued at $\begin{bmatrix}.9\\.1\\.85\end{bmatrix}$ which produces excitation

$(.9\cdot.2)+(.1\cdot.3)+(.85\cdot.4)=$

Section 2.4. Programming a single McCulloch-Pitts neuron

MatLab gives us at least two options for calculating the dendritic excitation. You can program your own inner product or you can use MatLab's. Eventually, speed of execution will be an issue, and MatLab is faster if you use its specialized linear algebra commands. However, some students are new to programming, so let's look at how to write your own.

Because reuse is implicit when we write a function, the function cannot be too specific. Consider a function, called `postexcite( )` that calculates the dendritic excitation of a McCulloch-Pitts neuron.

```
function [post_excitation] =postexcite(inputvector,weightvector)
            %input vector and weight vector must use column
            %vectors of the same dimension

      post_excitation = 0;
      for axon_number = 1: length(inputvector)
      post_excitation = post_excitation + inputvector(axon_number) * …
      weightvector (axon_number);
end
```

Please access

```
>>help length
```

Notice how the command `length( )` is used to generalize the function to a neuron with an arbitrary number of inputs.

Whenever we write a function or a program, we always stop and ask, what can go wrong? If the wrong type of data is provided to this function, it will fail to execute. In a large program with many functions, it is wise to embed error-checking procedures. This is very hard for the novice, much harder than writing the original functions or program itself, because the novice tends to ignore, or fails to understand, the great breadth of possibilities when it comes to mistakes.

In the previous example, the most obvious possibility is that the input vector and the weight vector are not the same dimension.

Here is an error checking routine that might be inserted into the function `postexcite( )` prior to execution of the for-loop or inner product.

```
If length(inputvector) ~= length(weightvector)
   disp('error in my Function postexcite')
   disp('dimensions of inputs and weight are different')
end
```

However, you might want to make this a function so you can reuse it. In that case, you might write:

```
function  checkvecs(vec1,vec2,func_name)
```

where `func_name` is a string variable that identifies where `checkvecs( )` is being used. Of course, you need to alter the print command appropriately.

To understand the novel commands, please access

```
>>help ~=
>>help if
>>help disp
```

Note the use of the "not equal" comparison to control the outcome of the conditional if-then statement.

Table 2.2. The inner product

Returning to the calculation of postsynaptic excitation, there is a simpler notation, and faster execution, when we take full advantage of MatLab. This notation captures the parallel, instantaneous nature of the calculation of postsynaptic excitation. The operation is called, somewhat mysteriously for us, the *inner product* of two vectors or, more sensibly, the *scalar product* of two vectors, which means exactly the same thing.

To examine the inner product operation, we first need to parameterize two vectors. Here is an example. Note that `input1` and `weightsj` are specific, three-dimensional vectors whose elements are each arranged as a column. Enter each element of the input column vector `input1` in succession.

```
>>input1=[1;0;0.5]
>>input1
            %just to check, print the entire vector to the screen
>>input1'
            %the prime indicates the transpose operation which
            %turns a column vector into a row vector
```

Now enter each element of the synaptic weights vector `weightsj` in succession:

```
>>weightsj=[0.1;0.3;0.2]
         %enter each element of the synaptic weight vector
         %weightsj in succession; and again, we've created a
         %single three dimensional column vector

>>weightsj
         %print the entire synaptic weight vector to the screen
```

Now comes the operation of interest—the inner (scalar) product computation: this operation multiples, element-by-element, a row vector by a column vector and adds all of these pairwise multiplications to produce a single scalar value. If you calculate this answer by hand, what do you get? Now ask MatLab for the answer and compare it to your calculation.

```
>>input1'*weightsj
         %calculate the inner product excitation produced by the
         %transpose of the single vector input, input1, and the
         %postsynaptic weights, weightsj
```

Thus, when an input vector arrives at a postsynaptic neuron, the neuron calculates the inner product for each presynaptic input and the associated synaptic weight and then sums these products to obtain its postsynaptic excitation.

Using new values and variable names, compare the result of the MatLab code to the summation calculation shown in the text just above. The results from these two calculations should be the same.

*(Note: the order of multiplication and the use of the apostrophe are critical. The row times column operational form is just the notational definition of the inner product—it doesn't have any meaning, but it is a convention everyone uses to keep track of linear operations. Therefore you too must use this same convention. If you have trouble remembering the rule—**row vector times column vector yields a scalar**—just test yourself using MatLab to multiply two vectors).*

For the next example let us assume that you have opted to calculate excitation with MatLab's inner product command (see Table 2.2). Here is such a function that calculates the same dendrosomatic excitation.

```
function  [post_excitation] = postexcite(inputvector,weightvector)

    checkvecs(inputvector, weightvector, 'postexcite')
         post_excitation = inputvector' * weightvector;
```

Obviously a very short and convenient function, but what could go wrong?

Here are three possible errors that seem to occur much more often than might be believed: (i) the column vector convention is violated with one of the vectors being a row vector; a nonconformable matrix is used; or even more subtle, (iii) an unintended conformable matrix. It is extremely difficult to catch all possible errors, but here is a routine that will catch a lot of them.

We will start with the same checking routine as before, but we need to be careful because the `length( )` command returns the same value for a column vector or the row vector formed by the column vector's transform. Worse still, `length( )` returns the larger dimension of a matrix. Try some examples.

On the other hand, the `size( )` command gives more information than `length( )` and can be used to avoid the mistakes `length( )`. Lets step through some demonstrations.

Execute from the command line,

```
>>mymatrix = rand(3,2)

>>[numrows,numcols] = size(mymatrix)

>>numrows

>>numcols
```

Now compare the `size( )` and `length( )` commands with the two vectors

```
>>myvector1 = rand(3,1)

>>myvector2 = rand(1,3)

>>length(myvector1)

>>length(myvector2)

>>size(myvector1)

>>size(myvector2)
```

Table 2.3.

Using the `disp()` command to print messages to the screen

```
>>disp('use disp() to print a string and use brackets, [ ], to concatenate
strings')
use disp() to print a string and use brackets, [ ], to concatenate strings'
```

Here is an optional technique for displaying a message that includes characters and numbers using `num2str`.

```
>>number = 9
>>disp(['use num2str to print a numerical variable, such as ',
num2str(number),'.'])
```

Now using the `size` command, we write an error-trapping function and appropriate messages that check for two things: (i) the two inputs are column vectors and (ii) they both have the same dimensions. You should add this or a similar error-detecting function to your excitation function, whether your excitation function calculates each neuron's somatodendritic excitation via a for-loop or does that calculation via an inner-product command. However, in either case, be sure to include the error-trapping function in your McCulloch-Pitts neuron excitation function.

```
function [truth]=checkfor_col_vector(input,desired_dim,… func_name)

            %function returns the binary integer as the variable truth
            %truth is 1 if true or zero if false; desired_dim must
            %be a positive scalar

errorfound =['Function checkfor_col_vector has detected an error in'
,func_name];

truth = 1;
[numb_rows,numb_cols] = size(input);

if numb_cols ~= 1
    beep
    truth = 0;
    disp(errorfound)
    disp('Not a column vector')
elseif numb_rows~=desired_dim
            % must be a column vector to get here
           beep
           truth = 0;
           disp('errorfound')
           disp('Column vector is not desired length')
       end
end
```

Section 2.5. Setting threshold

A threshold is a simple logical operation that compares two values. A neuron must decide if its excitation is greater than or equal to some threshold value. If this is true, then that neuron fires an action potential; if it is false, the neuron does not fire. You can also think of a threshold as a computational parameter of a neuron that defines the minimum excitation needed for that neuron to fire.

Therefore, let's complete a single McCulloch-Pitts implementation by adding a spike-generating threshold function that depends on the somatodendritic excitation. There are several ways we could implement a threshold with the available MatLab commands, but the if-then-else function is the most explicit in terms of what is happening.

```
function [neuron_output] = spikegen(threshold, excitation)
            %threshold and excitation must be scalars

           %error detection
            if length(excitation) > 1
               disp('function spikegen takes a scalar not a vector or matrix')
            end

            %threshold function
            if excitation < threshold
               neuron_output = 0;
            else
               neuron_output = 1;
            end
```

For the Lab 2 homework, keep the two functions separate. However, because these two functions are so simple, in the future, you can combine them if you wish.

With these two functions available, you can easily create a simple, one-neuron network simulation. The vector inputs, which might change over time, will be successively fed into a parameterized McCulloch-Pitts neuron. Thus, to examine how threshold controls firing you need some input values, and you need to initialize the simulation parameters.

Let there be a single postsynaptic neuron with three distinct inputs whose weights are 0.4, 0.5, and 0.6 respectively. Use the random number generator to generate a three-dimensional nonnegative input vector and then normalize each such three-dimensional input vector. Create one thousand such random inputs. The assignment is to set threshold so that the neuron fires close to half the time, approximately, 490-510 times from the fixed set of 1000 random inputs. The next section will help you conceptualize and visualize the roles played by thresholds and weights in pattern recognition.

Homework 2.5.1.

1. To illustrate the threshold setting that gives the desired firing rate, use a relative frequency plot (as devised in Lab 1) of the neuronal dendrosomatic excitation. We would like to find threshold with these data. However, a better visualization, and one that will make your task easier, is a cumulative relative frequency histogram. Make a cumulative relative frequency histogram using the same data (see appendix 1.4). A cumulative relative histogram adds the successive bins of your relative frequency histogram. Necessarily the last bin is height one. Comment on the location of threshold versus the probability distribution. (Make sure your cumulative relative frequency histogram is a function you can reuse.) Show where threshold is on the x-axis; for example, use MatLab's visual interface to draw a vertical line or an arrow at the relevant place on the x-axis of your histograms.

Don't forget that you'll need to add your own random seed command to your homework.

2. In what sense is this McCulloch-Pitts neuron a parallel computational device? Give a specific example.
3. In what sense is this neuron a decision-making device? Describe the decision.

Section 2.6. Visualizing a neuronal decision

Synaptic weights and understanding threshold

To quantify an input vector in a manner that is suitable for an unambiguous binary decision requires the transformation of each vector input to a scalar. For each input vector or equivalently, for each input pattern, a neuron transforms this vector in a way that allows it to answer the question, "How close is this input to my 'best' or strongest exciting pattern?"

We can visualize a neuron with a two dimensional input by plotting points in a two dimensional space where each point is an input. To make comparisons even easier, we will require all inputs to be normalized to the unit circle. (This is a Euclidean normalization, making the root mean square of the elements of a vector equal to one. Thus, an n-dimensional input can be normalized to the surface of a n-dimensional hypersphere.) We will also normalize the synaptic weights of a neuron to length 1, see Fig. 2.3.

This favorite pattern can be thought of in terms of synaptic weights as shown in Fig. 2.7.2. Suppose a neuron receives two inputs and has a synaptic weight vector $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$. Now consider the excitatory effect of two different inputs, $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ and $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$. To illustrate this situation (that is, we project *v* onto *w* and project *u* onto *w*) we have normalized the weights and the two inputs to the unit circle. If we then drop a perpendicular from each of the two inputs to the weight vector *w*, we can use trigonometry to determine how much each input "excites" the neuron.

Let's say that the angle between the weight vector and the first input vector (v) is called α. Then, we know that cos(α) = adjacent/hypotenuse, or ($v^T w$) / ($\|w\|$ $\|v\|$), which is simply equal to $v^T w$, since $\|w\|$ and $\|v\|$ are both 1 (i.e., both are normalized to the unit circle) (see Appendix 2.2). We can see that the closer the input is to the synaptic weights, the larger the cosine of the angle between the two vectors, and the longer the length of *v* projected onto *w*. We can use this information to help us determine an appropriate threshold for our neuron.

Table 2.4.

1. Threshold defines the minimum excitation to fire a neuron.
2. A neuron's input synaptic weight vector defines a point in a geometric space.
3. An input vector defines a point in the same geometric space.
4. Given the normalization constraint, the greater the distance between these two points, the less the somatodendritic excitation.
5. Therefore, threshold partitions the space of possible inputs into a set of points that are near versus far from a central point defined by a neuron's input weights.

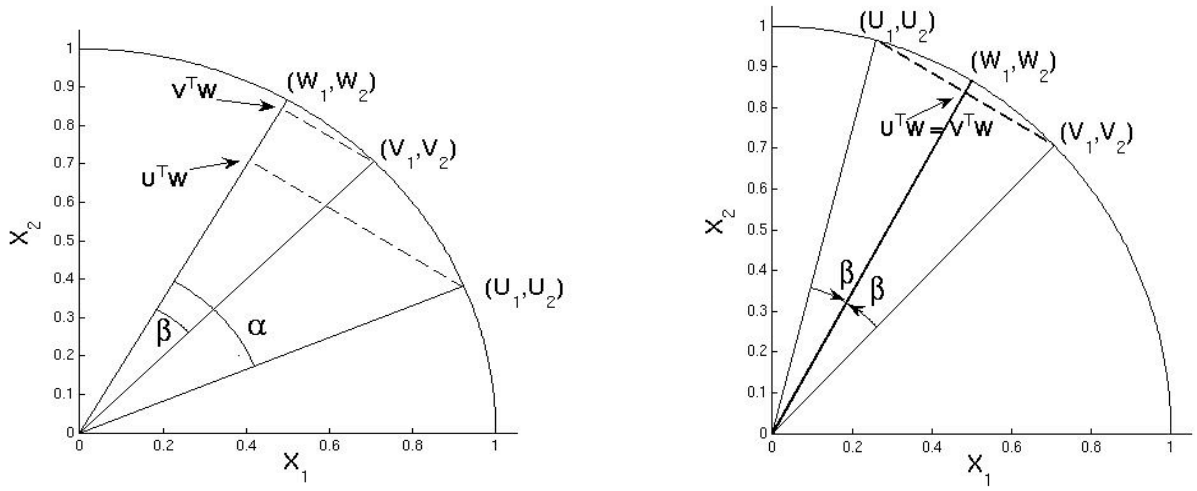

Fig. 2.3. (**A**) Input excitation is greater when an input vector is closer to a neuron's weight vector. This is most easily seen by projecting each normalized input vector onto the weight vector. Recalling the cosine relationship between two normalized vectors is the same as their inner product, we can see input $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ produces greater excitation than input $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ because v more closely aligns with $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ than u aligns with w. That is, angle β is less than angle α which corresponds to the indicated inner products $v^T w$ for cos(β) and $u^T w$ for cosine (α). (**B**) Somatodendritic excitation does not distinguish two input vectors that are symmetric about the neuron's weight vector. Threshold produces a continuous region that is symmetric around a neuron's weight vector. If a neuron has normalized weight vector $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$, then the normalized input vectors that can reach or exceed threshold form a continuous set around the neuron's weight vector. That is, if $u^T w$ is exactly at threshold, then so is $v^T w$ and if we include these two vectors and all the points of the arc between them, we have pictured the complete set of inputs that fire the neuron.

Suppose we want a neuron to recognize a set of five different input patterns (see Fig. 2.4A) where all these patterns lie on the unit circle and are near each other. (Remember, an input normalized to the unit circle is a point at distance one from the origin.) A neuron makes a successful pattern recognition decision if it fires (output=1) to an input pattern that is a member of this set of input patterns. How can we parameterize the synaptic weights and the threshold of a neuron to recognize patterns successfully?

First, consider this problem if all the synaptic weights are positive and the threshold is zero. In this case, the neuron fires in response to all input patterns. This problem is trivial and uninteresting because the neuron can only make one decision—namely, it recognizes everything (i.e., every point in the positive quadrant of the unit circle). In essence this neuron transmits no information, it just says the same thing each time. On the other hand, our neuron is doing nothing

wrong for a world like Fig. 2.4A. However, if the world is like Fig. 2.4B, then the neuron's strategy is no good.

As illustrated in Fig. 2.4B, the decision problem becomes more interesting if we add another set of input patterns. Now there are some patterns to discriminate between. Fig. 2.4B depicts the situation with a set of three 'wrong' patterns, i.e., a set of three patterns that lie outside the set we want the neuron to recognize. In this case, our initial strategy of all positive synaptic weights and a threshold of zero fails because the neuron does not discriminate between the two pattern sets. To parameterize our neuron successfully, we must do a little thinking.

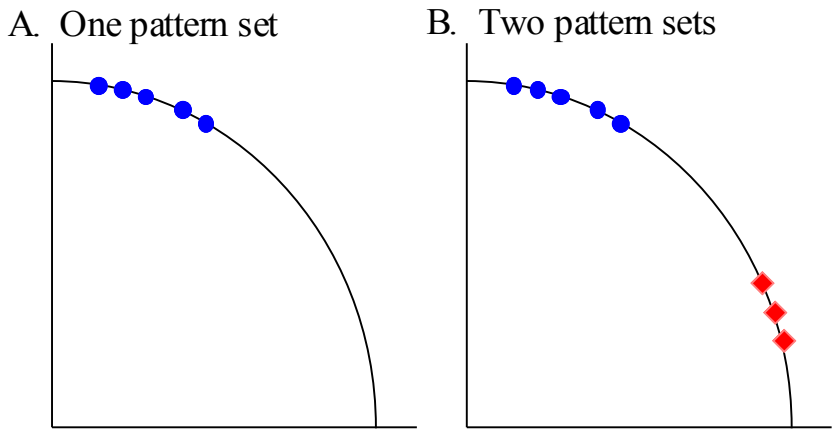

Fig. 2.4. Geometric representation of pattern sets on the unit circle. **A**, example of one set of input patterns. **B**, example of two sets of non-overlapping input patterns, one represented by blue circles and the other by red diamonds.

For the problem depicted in Fig. 2.4B, there are only two sets of input patterns and, relative to their positions on the unit circle, these two sets do not overlap. As a result, we can draw a line from the origin to the unit circle such that all patterns of the first set are on one side of this line and all patterns of the second set are on the other side of this line (Fig. 2.5A). Because only a single line through the origin is needed to separate all the members of the two sets, we refer to these two sets of points as *linearly separable*. Of course, pattern sets are not always separable in this simple manner (e.g., see Fig. 2.5B).

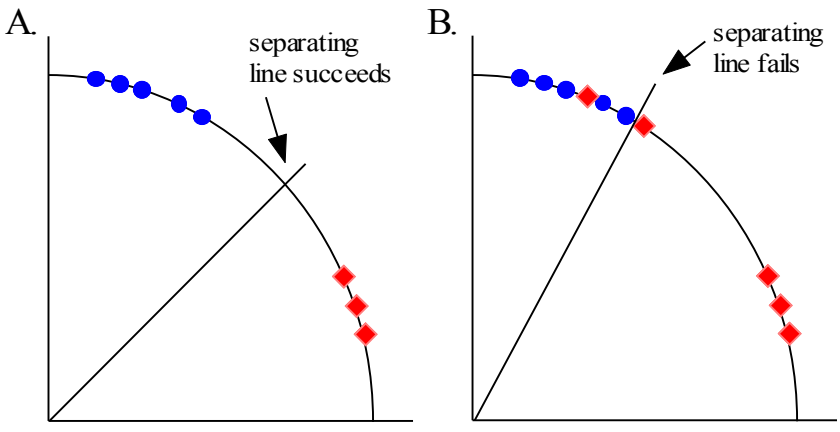

Fig. 2.5. Linear separation of pattern sets: one success and one failure. **A**, successful linear separation of two pattern sets; **B**, there is no error-free separating line (although the diamond next to the separating line is no particular problem, note the problematic red diamond sandwiched among the blue circles).

Linear separability here implies that we can construct a neural network with two postsynaptic neurons in which the network makes errorless decisions about the data points relative to the two

pattern sets. This means that one postsynaptic neuron can be parameterized to fire an action potential if, and only if, a member of the first pattern set is an input while the second postsynaptic neuron can be parameterized to fire selectively only to a member of the second pattern set.

Although many different parameterizations can work, here (e.g., Fig. 2.4B or, equivalently, Fig. 2.5A) is a simple one: Set the synaptic weights of neuron one as $\begin{bmatrix} w_{11} \\ w_{21} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and set those of neuron two as $\begin{bmatrix} w_{12} \\ w_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Then, draw a line from the origin to the unit circle that separates the two sets of sample points. “Separate” means that all sample points belonging to one pattern set lie on one side of this line while, on the other side of the line, lie all the sample points of the second pattern set. This separating line tells us where we can place the thresholds—that is, the threshold establishes the boundary for set inclusion/exclusion. Just project this separating line onto each axis. The length of each projection is the threshold for the corresponding neuron.

A better way to set threshold

For two linearly separable sets with elements on the unit circle, start by creating an appropriate weight vector for each set. For each particular set, this weight vector should be placed in the 'center' of its set where, just now, center is defined as bisecting the two extreme points of the set. Next, to find the largest threshold that will fire the neuron for all members of this set, project a perpendicular from an extreme point onto this bisecting line. (Note both extreme points should produce the same intersection point; why?). Declare threshold to be the length of the bisecting line from origin to intersection point. Note in this case, there are two angles to find but of course they can be calculated.

In either procedure, we have now fully parameterized this two-neuron network by identifying the thresholds of both neurons so that the network is capable of perfect performance.

Why thresholding can be useful

Threshold-based decision-making can be used to describe cognitive performance. As a function of neuronal weights and threshold, there is a relationship between the set of input patterns at the neuronal level that reproduces two cognitive concepts- generalization and discrimination, which are summarized in Table 2.5.

Questions to by answered: In what sense(s) is a neuron and its weight vector like an axis of a graph? Consider a two-dimensional space with axes NOT at right angles. Can two-dimensional data points be uniquely represented with such axes?

Table 2.5.

1. Generalization Principle: Similar patterns should be treated similarly. That is, patterns near to each other should be treated similarly. 2. Discrimination Principle: The greater the distance between two patterns, the more they should be encoded distinctively and treated differently. (How might you generalize this last to two overlapping sets?)

The plot points in Fig. 2.4B illustrate these two principles in action. The points that are crowded together are presumed to be of the same kind. That is, the circular bluish-purple points are close to each other and the red diamond points are close to each other. Remembering that each point is a pattern, we see the generalization principle in action when we group the similarly colored points into the same set. Likewise, we see the discrimination principle in action when we decide that there are two distinct sets.

Nonlinear Discrimination

A combination of linear separators can sometimes be used to discriminate examples such as Fig. 2.5B. Four linear neurons could be used followed by appropriate combining at a later stage of processing the output of these four neurons. There are at least two difficulties with generalizing such a solution. First in terms of number of neurons used, it will be expensive if there is an arbitrary intermixing of neighbors derived from differing classes. Second, such discrimination requires some kind of teaching because many of the discriminations do not follow from the neighborly principles (see Table 2.5) just mentioned.

Section 2.7. Two Class Exercises in Preparation for the Homework

Here we extend the idea of parallel computation by programming and simulating a network in which two postsynaptic neurons operate in parallel each receiving two input connections. This network is schematically depicted in Fig. 2.6. The input environment is divided into two categories called A and B. Here we'll create two sets of input patterns placed on the unit circle. Then we will create two McCulloch-Pitts neurons to recognize the two pattern sets (with each neuron correctly recognizing only members of one input pattern set). That is, we must set their synaptic weights and threshold so that each neuron recognizes only one set of input patterns. (Use the same threshold for both neurons.)

For the two sets of input patterns, generate random numbers on the unit circle by rescaling the random number generator to go over a range of $\pi/6$ radians. Generate two input pattern sets with the same range ($\pi/6$ radians) but be sure to offset one set so that they do not overlap with the other set (see section 1.5 for insights). In each set, generate 25 points. Suppose the input pattern sets, called `XinputPat1` and `XinputPat2`, are two matrices that are 2 rows by 25 columns.

One neuron should fire only to the first set of 25 patterns, and the second neuron should fire only to the second set of 25 patterns. Extend your single McCulloch-Pitts neuron to multiple neurons. You have the option of using the function you already have or writing a more general function.

In a few weeks, you will need a more general function that can use an arbitrary number of postsynaptic neurons.

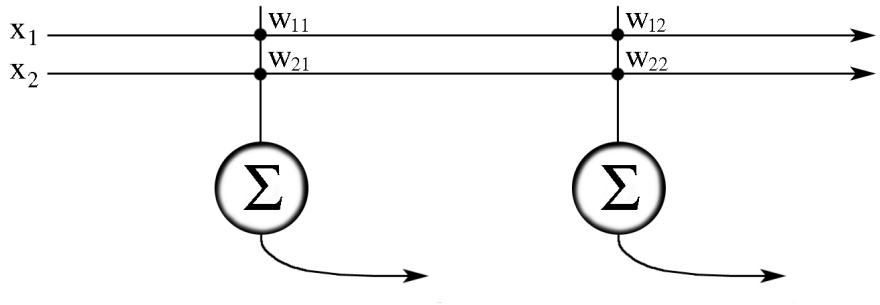

Fig 2.6. Two McCulloch-Pitts postsynaptic neurons each receiving the same input, $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$.

There are four synapses formed whose weights are indicated by w_{ij}. Each neuron adds up its weighted inputs. If the sum equals or exceeds threshold, the spike is generated.

A weight matrix defines a feedforward network. Call such a weight matrix *W*. Then a single vector input, x, excites all of the post synaptic neurons producing the vector of excitations, y. In terms of linear algebra, $x^T \cdot W = y$, where y is a row vector. (If a column vector of excitation is desired, the calculation is $W^T \cdot X = y^T$, where ^{T} turns the columns into rows and vice versa.).

We can also conceptualize many vector inputs as a matrix. Let *X* be m column vectors representing n different inputs to a single neuron. Then there will be m excitations generated by this input matrix. In terms of linear algebra, $X^T w = y$, where w is a list of *n* weights and *y* is a column vector of *m* excitations. Putting this together with the previous paragraph, linear algebra provides a compact form for writing the excitation of k neurons by m input vectors, $X^T W = Y$, where *Y* is a *k* by *m* matrix. That is, each of the *k* different rows represent the excitation of the network by one vector input.

Remark: Using two neurons to code membership in one of two sets seems very inefficient as one neuron is enough, just allow the "1" state to represent membership in one category and the "0" state to represent membership in the other category. However, we would like to use codes in the all-positive orthant and such inefficiency is barely noticeable when there are many categories.

Homework 2.7.1.

1. Explain why there can be more than two thresholds that produce zero error for the patterns of Fig. 2.4B. Compare this to the nearest actual pattern samples. (*Hint:* consider the excitation generated in each neuron by the point you used to determine the threshold).

2. Explain why increasing the threshold while holding the synaptic weights constant allows the neuron to be a more selective decision-maker. What does selective mean? Be sure to study and understand Fig. 2.3 and its legend when you answer this question.

3. What is the largest threshold you can use and still have a neuron recognize all of the patterns in its designated set of input patterns (use Fig. 2.4B)? Explain this in terms of the geometry we just used to construct the threshold. (*Hint*: A single neuron has only one threshold, but two neurons can have different thresholds.)

4. Using the same sample points as in Fig. 2.3B, suppose we insist that both neurons have the same threshold and that recognition must be perfect. How do you select this single, shared threshold value? (*Hint:* you might find it useful to consider and discuss arc lengths of the unit circle relative to each input pattern set).

5. Given an input set of three vectors on the unit circle, namely, $\left\{\frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}\right\}$ and weights [0.5, 0.5], can you identify a threshold such that a neuron fires to input vectors $\pi/6$ and $\pi/4$ but not to input vector $\pi/3$? If so, comment on whether higher or lower threshold values work and why. If not, explain why not. (*Hint*: See Fig. 2.3B.)

6. Using the same input set as in exercise 5, create a new synaptic weight, vector, and threshold such that the neuron fires to input vectors $\pi/3$ and $\pi/4$ but not to $\pi/6$.

7. Again using the same input set, list the weights and thresholds of three neurons with the requirement that each neuron fire selectively to a single, but different, input pattern (i.e., one fires only to input vector $\pi/6$, and then one only to input vector $\pi/3$, and the third fires only to input vector $\pi/4$). Explain how you got your answers and include a diagram if it helps.

Homework 2.7.2. (Read completely before starting)

This exercise is divided into two parts. In both cases, you'll run simulations that discriminate among input pattern sets. You will need two postsynaptic neurons in the first environment and three in the second one.

Environment 1: (This part is not to be handed in) Create two non-overlapping data sets on the unit circle. Each set should have a central pattern that is perturbed by the addition of small random numbers to create the elements of the set. Make sure there is enough variability between input patterns so that you can distinguish the different exemplars of each category in your figures. See for example Fig. 2.4B. Save these input sets for use in Lab 4.

Environment 2: Now add a third non-overlapping set to environment 1.Take the two input sets of environment 1 and insert a third set between them (e.g., make the range of each set equal to $\pi/6$ radians, such that the three sets fill the positive quadrant of the unit circle). Construct a feedforward categorization network with three postsynaptic neurons that discriminates between the input pattern sets of this environment. You should feel free to use different thresholds for the different neurons. Save these input sets for use in Lab 4.

Questions to answer:

What, if any, advantage is there in setting the synaptic weight vector to the center of each category? Your answer should explain the role threshold plays in concert with weight vectors.

Suppose there are lots of specific inputs uniformly distributed on the unit circle and that there are two neurons with identical thresholds which would be used to discriminate inputs. Devise a scheme for discriminating as many different positions on the unit circle. Be sure to specify the two weight vectors and the threshold. (Assume all thresholds are the same.) Explain your answer. Note that there are four different, i.e. distinct, outputs for a two neuron system - {{0,0}, {1,0}, {0,1}, {1,1}} be sure to use all of these output states. Name each of the four states. Identify an equal arc length for each state (if you haven't already done this). It is OK to have two noncontiguous arcs correspond to the same state. (Hint: $\frac{\pi}{2} \cdot \frac{1}{4} = \frac{\pi}{8}$; have each output state cover the same arc length.) Is there more than one answer that produces the same (i.e., best possible) result? Why/Why not?

Homework 2.7.3. (Read completely before starting)

Assume input normalization, and suppose there are two postsynaptic neurons. How can an input (i.e., a point in the input space) be localized more precisely in the geometric space with these two neurons versus either one alone?

Suppose input points are not uniformly spread throughout the input space; rather there is some region of the space where inputs are more likely. What weight vectors and thresholds would you choose to localize inputs as accurately as possible?

Try this problem for one neuron. Inputs are normalized and nonnegative. Suppose 90% of the points are uniformly spread between $\frac{\pi}{4}-\frac{\pi}{16}$ and $\frac{\pi}{4}+\frac{\pi}{16}$ and 10% of the points are uniformly spread outside of this region. Let the weight vector be normalized. For a fixed threshold, show that a weight vector centered on this dense region minimizes the error.

Create two neurons to make your comparison. Give them the same threshold, which means that each neuron will recognize the same arc-length of inputs. Make the neurons different by virtue of the weight vector associated with each one by putting one of these vectors in the center of the dense part of the pattern set and the other outside of this subset.

What happens to the average error if threshold is changed? Create examples. What do you conjecture about weight vectors when input points are not uniformly concentrated? Consider the physical world you encounter and argue that input possibilities are concentrated.

Section 2.8. An exploration of errors

Just as in statistics, we define the errors made by a McCulloch-Pitts neuron as one of two types, either *type I* or *type II*. A type I error occurs if a hypothesis is rejected when it is in fact true. A type II error occurs if a hypothesis is accepted when it is actually false. We will define a type I error of a neuron as a *failure to fire* when a pattern *is* a member of the input set it should recognize. Conversely, we will define a type II error as *firing* to a pattern that is *not* a member of the set of input patterns that the neuron should recognize.

Homework 2.8.1.

You need to create an input set. Format this input set as a matrix with a row for each input axon and a column for each input vector.

Program a function that can create your input data. The user will specify the input dimensions and number of inputs. The random numbers might be binary with 1 at probability p or continuously valued, [0, 1].

1. For a given weight vector, what is the mean and variance of the postsynaptic excitation as a function of the input data matrix? Write a program that performs the calculation.
2. If you have not already done so, use the linear algebra functions to write its program. Write the mean and variance as a linear algebra function given that the input matrix is called InputMatrix.

In two dimensions, create an input set in which the two input pattern sets overlap. Here is a simple example you can use:

(a) Place all patterns on the unit circle.

(b) Both input sets should have a range of $\pi/3$ radians:

(i) the first set has no offset,

(ii) the second set has an offset of $\pi/6$ radians from the abscissa.

With a large enough number of patterns per input set, it is now impossible for a single linear neuron to recognize selectively just one of these input sets. Why? Construct and use a picture to illustrate your explanation.

Using a large number of patterns in each of two input sets just described (or a similar pair of input sets of your own specification), construct a 2-input, 2-neuron network of McCulloch-Pitts neurons. Set the synaptic weights to whatever values you wish but make them mirror images of each other for the sake of symmetry (i.e., $w_{12}=w_{21}$ and $w_{11}=w_{22}$).

Homework 2.8.2

Learn how the threshold parameter of your network affects performance by doing the following exercises (refer to Homework 2.8.1):

1. Keeping the synaptic weights fixed and using the same threshold for both neurons, vary this threshold systematically.
2. Keep track of the type I and type II errors made by each of the two neurons as a function of threshold. How does threshold affect the number of type I and type II errors made by the network?
3. On a single plot, separately graph, as a function of threshold, the total number of type I errors and the total number of type II errors (see the MatLab command, **hold**, for insight into plotting two curves on one plot). At what point do the plots intersect? What is the most appropriate threshold to use in order to minimize error for both neurons in the network?
4. On a second graph, plot the total number of errors as a function of threshold. Describe the relationship you observe between threshold and the commission of type I and type II errors.

Homework 2.8.3.

Discuss in your laboratory which is worse—a type I or type II error? Suppose the two error types are equally undesirable. Use your plots to identify an optimal threshold that minimizes total error. In general, how would you suggest parameterizing neurons to achieve this minimal total error?

For students interested in going further: If you are allowed three neurons, do you think you could create a network with better performance than a network with only two neurons? If so, what would you do and how much better would performance be? If not, why not? Could you improve performance with even more neurons? Why or why not?

Homework 2.8.4. Your lab write-up

The following elements should be part of your lab write-up. Your instructor will look for these items when grading the lab.

- a diagram of each network
- a definition of categorization
- a definition and explanation of the inner product
- a definition of McCulloch-Pitts neurons related to the variables of internal excitation and output activation
- a definition of Type I and Type II errors and a depiction of the trade-off between these errors
- a description of how the input sets were generated and the characteristics used to inspect them (e.g. whether and how the points were distributed across the appropriate region)
- a description of the interaction between weights and thresholds that determines a neuron's discrimination performance relative to an input pattern set
- a description of the sense in which a neuron implicitly draws separating line(s)
- an explanation of how MatLab's linear algebra commands can capture the essence of parallel processing, at least at the level of the way the code is written.

Section 2.9. Summary

A quantitative description of a McCulloch-Pitts neuron requires:

1. the set of all input axons and the synaptic weight associated with each axon (this synaptic weight scales the potency of each active, presynaptic input),
2. the threshold for firing this neuron.

Moreover, a McCulloch-Pitts neuron

3. has no memory from one timestep to the next;
4. within a timestep, it linearly sums its inputs as scaled by their respective synaptic weights;
5. it fires (i.e., its output equals 1) if the value of its internal excitation exceeds a preset threshold value of excitation, otherwise it does not fire (output equals 0).
6. Threshold can be viewed in relationship to a neuron's probability distribution of dendrosomatic excitation. In this case, threshold helps determine the fraction of time a neuron fires.
7. A neuron's input weights are a point in multidimensional geometric space as are the inputs to a neuron. With this geometric perspective, threshold can be viewed as specifying a partitioning of the space in terms of "closeness" to the weight vector. All normalized inputs closer to the weight vector than the end of the arc specified by threshold are recognized by the neuron.
8. Because of this geometric perspective and the projection created by the inner product, a neuron's input weight vector specifies an axis; several neurons specify several axes which

happen to be nonorthogonal. Such nonorthogonal axes can be used to localize points of the input space.

9. As a binary pattern recognition device, there are two types of errors a neuron can make.

Section 2.10. Possible quiz questions

1. Explain how a neuron can act as a decision-making device and how it can recognize a pattern.
2. How is the decision whether a pattern is recognized by a neuron translated or coded? That is, how does the postsynaptic neuron tell us whether it recognizes a pattern or not?
3. What is linear separability?
4. What are some of the limitations of linear discriminations?
5. Based on your results in this laboratory, briefly summarize how input weights and threshold interact to determine which patterns are recognized by a postsynaptic neuron.
6. In what sense can a pattern recognition neuron generalize from its best input pattern?
7. What is the common characteristic of each normalized weight vector for three neurons where the first neuron gets two inputs, the second neuron gets three inputs, and the third neuron gets n inputs?
8. For a neuron that gets three normalized inputs and has a normalized weight vector, describe the geometric region that includes all patterns that will fire the neuron. (Suppose threshold is .95.)

Section 2.11. Advanced topics

Not quite normalized inputs

Question: Suppose that input lengths are allowed to vary around one as plus or minus a small constant. Suppose also that input points are uniformly distributed in two dimensions (picture a square lattice of points in the all-positive quadrant). Are more input points located in the annular region above or below the radius longer or shorter than length one? Can you extend this argument to 3-dimension or higher? Why?

Hint: Compare the areas of two annular regions of width δ, a small positive number. Calling the radius r, Area + δ = ($\pi (r + \delta)^2 - \pi r^2$) versus Area - δ =($\pi r^2 - \pi (r - \delta)^2$). Expand and compare. Do the same for the spherical shells, $\frac{4}{3}\pi(r+\delta)^3 - \frac{4}{3}\pi r^3$ versus $\frac{4}{3}\pi r^3 - \frac{4}{3}\pi(r-\delta)^3$. Can you spot a trend as the power (dimension) increases?

Variable Threshold

An upper class neuroscience text is bound to point out that threshold is a dubious concept. By this, we mean that threshold is not an absolute value but depends on excitation over time. Let's consider how this neurobiological observation could be put to use. That is, now consider a neuron with a very particular variable threshold, one that depends on the previous firing, can be exploited by Nature.

Because of its simplicity and power, we have concentrated on neurons with binary outputs. However, it is easy to imagine, and there are calculations supporting the idea, that a neuron can transmit an additional amount of information by combining successive transmissions. Here is a simplified version of this possibility from the encoding perspective.

Suppose the input to a neuron changes slowly; specifically, let's say that one level of excitation stays on for two time steps. Then, a variable threshold becomes useful.

Typically, the threshold of a neuron increases after it fires and it decreases if it has not fired for a while. Note how this allows a more precise localization of the separating plane: indeed, there are now multiple planes.

Homework 2.11.1.

A. Create a MatLab function that allows you to call a neuron with an arbitrary threshold. Use this function for the following problems. Suppose each input point appears for two time intervals. For the first timestep suppose that the neuron's threshold is set so that the neuron fires to an input on the unit circle with an arc of $\pi/6$ radians centered on the weight vector. Then on the second timestep, the threshold changes in a manner dependent on the output on the last timestep. If the neuron fires, then increase threshold so that the neuron's arc of recognition shrinks to $\pi/12$. But if the neuron does not fire on the first time step, its output is zero and we let the threshold requirement be less demanding (you choose a new value). For example, let the arc increase to $\pi/4$. (Be sure your neuron only fires to inputs in the all-positive quadrant.) In what sense have we enhanced the resolution of a neuron's response to any one input pattern? (*Hint*: there are four states that can be discriminated.)

B. (For the advanced student) Using such neurons, reconsider the second homework problem in 2.7.2.

Where to place weights

A better output code can transmit more information. An interspike interval code (called an interpulse interval code in electrical engineering) can transmit a variety of values with a single pulse. In particular, the time between any two spikes signals the intensity of a neuron's excitation: shorter intervals mean stronger excitation. With this output code we are motivated to move the neuron's weight vector to the weighted center of the region defined by its input set of recognized patterns. In such a case the output will reflect how close the input is to a neuron's best pattern. We will return to this issue in Lab 4.

Glossary Terms

distance	how far apart two points lie from each other. The Euclidean distance measure (see below) is the most typical of such measures, but it is far from the only one
Euclidean distance	for points (i.e., column vectors) X and Y, $\sqrt{((X - Y)^T (X - Y))}$
excitation	the scalar value that is the result of the postsynaptic summation process over a set of active synapses (see inner product)
feedforward network	the simplest network with one class of inputs originating outside of the primary neurons and whose states are not influenced by the states of the primary neurons.
inner product	(also known as the scalar product) the calculation of a postsynaptic excitation that is the multiplication of a row vector by a column vector to produce a single scalar value, or the excitation
McCulloch-Pitts neuron	a **model** neuron that is able to recognize patterns in its environment based on the features of a given stimulus and has the following characteristics: (1) synaptic weights that scale excitatory inputs, (2) a summation process that adds up the scaled inputs, (3) a spike generator that crudely encodes the value of this summation
neural network	an organized system of neurons that operate on patterns in very high-dimensional spaces, where an *n*-dimensional space means there are *n* inputs.
synaptic weight	the strength of a synapse that connects a presynaptic input to a postsynaptic cell
threshold	the specific value of a neuron at which an excitation causes that neuron to generate an action potential generation; the characteristic of a neuron that enables decision-making
type I error	failure to fire when a pattern is a member of the input set it should recognize
type II error	neuronal firing to a pattern that is not a member of the set of input patterns that the neuron should recognize
vector	a point in space*

* Although in this course we define a vector as a point, it is standard usage to refer to the length of a vector and the direction of a vector. This should not be confusing because the *length of a vector x* is the Euclidean distance from the origin to point vector x, and the *direction of a vector x* is the direction of the line drawn *from* the origin *to* a point vector. Of course, the origin is the vector of all zero elements with the same dimension as x.

Appendix 2.1. The Matrix Transpose

We have already seen in Lab 1 how the transpose operates on vectors; i.e., it can change a row to a column or a column to a row. The transpose also operates on a matrix in a similar way; i.e., the rows become the columns or, equivalently, vice versa. Just as with vectors, the transpose of a transposed matrix returns the original matrix. One more trick is needed. The transpose of a product reverses the order of multiplication. This is just what you saw for the vector inner product.

Here we introduce some properties of the matrix transpose and then use MatLab to illustrate some of these properties.

Consider a matrix `A` that is *m x n*. The transpose of matrix `A`, denoted A^T, is an *n x m* matrix such that

```
Atranspose = A'
```

In all cases, `A(i,j) - Atranspose(j,i)` equals zero.

In other words, row *i* of A^T consists of the entries of column *i* of `A`, and column *j* of A^T consists of the entries of row *j* of matrix `A`.

Here are three examples of the relationship between a matrix and its transpose.

$$\text{If } A = \begin{bmatrix} 2 & 1 \\ 3 & -1 \\ 0 & 4 \end{bmatrix}, \text{ then } A^T = \begin{bmatrix} 2 & 3 & 0 \\ 1 & -1 & 4 \end{bmatrix}.$$

$$\text{If } B = \begin{bmatrix} 1 & 7 & -8 \end{bmatrix}, \text{ then } B^T = \begin{bmatrix} 1 \\ 7 \\ -8 \end{bmatrix}$$

and, rather importantly for certain advanced calculations,

$$\text{If } I = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix}, \text{ then } I^T = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix}.$$

In the last example, `I = I`^{T}. Moreover, any matrix **A** with the property, $A^T = A$, is called a *symmetric matrix.* Thus matrix `A = [`a_{ij}`]` is symmetric if $a_{ij}=a_{ji}$ for every value of *i* and *j*.

Clearly, only a square matrix, i.e. a matrix with an equal number of rows and columns, can be symmetric.

The transpose has other important properties which are depicted in the next three equations.

$$(A^T)^T = A \quad \textbf{(1)}$$

In other words, the transpose of a transpose equals the original matrix.

$$(A + B)^T = A^T + B^T \quad \textbf{(2)}$$

This equation shows us that the transpose of the sum of two matrices is equal to the sum of the individual transposes. And, finally,

$$(AB)^T = B^T A^T. \quad \textbf{(3)}$$

Equation (3) demonstrates the equality between the transpose of the product of two matrices. In a matrix product, the order of the matrices is important. As shown in Equation (3), the order of the matrices is reversed when the transpose is distributed to the product.

From these properties, you can deduce the following: whenever a square matrix is added to its transpose or whenever any matrix is multiplied by its transpose, a symmetric matrix is always obtained. Therefore, it follows that:

$$(A + A^T)^T = A^T + (A^T)^T = A^T + A = A + A^T \quad \textbf{(4)}$$

and

$$(AA^T)^T = (A^T)^T A^T = AA^T \quad \textbf{(5)}$$

Appendix 2.2. Inner Products and Cosine Comparisons

Comparing representations, code words, or network states is fundamental to the quantitative understanding of neural computation.

A very popular comparison function is the cosine of an angle (recall that the cosine of an angle is defined, for a right triangle, as the side adjacent to the angle of interest divided by the hypotenuse). This comparison function is useful because a) it can compare two vectors in a state space regardless of the dimension of that space, and b) this comparison is normalized so that the dimension of the space (i.e., the number of neurons used for encoding) or the activity level of the network (which often fluctuates) does not affect this comparison.

The reason this comparison can be made regardless of dimension is that a vector is a point, and three points (e.g., the two vectors being compared and the origin) always define an angle in a two-dimensional plane. Thus, there is no need to wrestle with solid angles subtending hypersurfaces on a hypersphere.

Note that if the angle between two points is zero degrees, cos(0) = 1; if the angle between two points is ninety degrees ($\pi/2$ radians), then cos ($\pi/2$) = 0.

Here we show that the cosine between two column vectors, x and w, is easily calculated through linear algebra as

$$\cos(x,w) = \frac{x^T w}{\left(x^T x\right)^{\frac{1}{2}}\left(w^T w\right)^{\frac{1}{2}}}. \qquad (1)$$

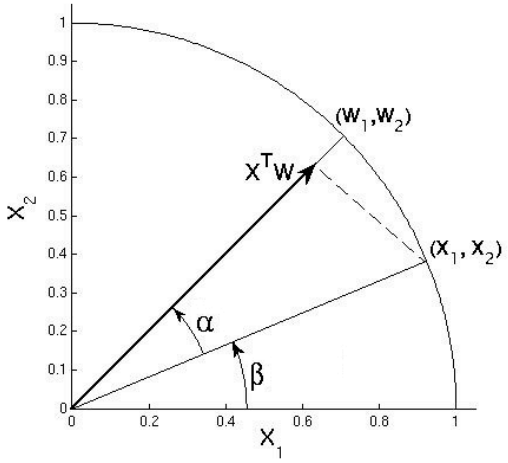

Fig. 2.8. *The projection of $x^T w$ of the unit vector x onto the unit vector w*
The angle between x and w is α. The angle between x and the abscissa is β.

Consider the vector $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ making angle β with the abscissa and another vector $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ with an angle α between it and vector x. We use the angle α or cos (α) to describe how vector x differs from vector w.

From the definitions of cosine and sine,

(2) $$\cos(\beta) = \frac{x_1}{(x^T x)^{\frac{1}{2}}}$$

(3) $$\sin(\beta) = \frac{x_2}{(x^T x)^{\frac{1}{2}}}$$

because the hypotenuse, i.e., the denominator $\left(x^T x\right)\frac{1}{2}$, is the Euclidean length of *x*. Likewise for the point $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$,

(4) $$\cos(\alpha + \beta) = \frac{w_1}{(w^T w)^{\frac{1}{2}}}$$

(5) $$\sin(\alpha + \beta) = \frac{w_2}{(w^T w)^{\frac{1}{2}}}$$

Using (2)-(5) the mathematical notation of distance $\left(x^T x\right)^{\frac{1}{2}} = \|x\|$ and (2)-(5) likewise for *w* we rewrite the angular positions of *x* and *w*.

Thus $$\frac{x}{\|x\|} = \frac{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}{\|x\|} = \begin{bmatrix} \cos(\beta) \\ \sin(\beta) \end{bmatrix}$$ and

$$\frac{w}{\|w\|} = \frac{\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}}{\|w\|} = \begin{bmatrix} \cos(\alpha+\beta) \\ \sin(\alpha+\beta) \end{bmatrix}.$$

Recall (6) $\sin(\alpha+\beta) = \sin(\beta)\cos(\alpha) + \cos(\beta)\sin(\alpha)$, and
(7) $\cos(\alpha+\beta) = \cos(\beta)\cos(\alpha) - \sin(\beta)\sin(\alpha)$.

Now we will reduce the right hand side of eq. (1) to a cosine.

$$\frac{x^T w}{\left(x^T x\right)^{\frac{1}{2}}\left(w^T w\right)^{\frac{1}{2}}} = \frac{x^T}{\|x\|} \cdot \frac{w}{\|w\|} = \left[\cos(\beta), \sin(\beta)\right]\begin{bmatrix} \cos(\alpha+\beta) \\ \sin(\alpha+\beta) \end{bmatrix}$$

$$= \cos(\beta)\cos(\alpha+\beta) + \sin(\beta)\sin(\alpha+\beta)$$

$$= \cos(\beta)^2\ \cos(\alpha) + \sin(\beta)^2\ \cos(\alpha)$$

$$= \cos(\alpha)\left(\sin(\beta)^2 + \cos(\beta)^2\right)$$

$$= \cos(\alpha)$$

Finally, we designate the angle α by the points that define it with regard to the origin. That is, $\cos(\alpha) \equiv \cos(x, w)$.

It is worth noting that cosine is inherently normalized for length in the sense that it is the creation of an inner product between two normalized vectors, i.e., $\dfrac{x^T w}{\left(x^T x\right)^{\frac{1}{2}}\left(w^T w\right)^{\frac{1}{2}}} = \dfrac{x^T}{\left(x^T x\right)^{\frac{1}{2}}} \cdot \dfrac{w}{\left(w^T w\right)^{\frac{1}{2}}}$.

However if *x* and *w* are normalized to length one, then $\cos\ (\alpha) = \dfrac{x^T w}{1 \cdot 1} = x^T w$.

Of course, the comparison is symmetric because the transpose of a scalar is that very scalar, and thus $x^T w = (x^T w)^T = w^T x$, or from a geometric viewpoint, a cosine is symmetric about zero. Thus, the inner product, just like the cosine, function does not discriminate directions of above or below the nominal axis.

Appendix 2.3. Matrix Notation and Creating a Network of Two or More McCulloch-Pitts Neurons

We digress for a moment to point out the notation for a synapse, w_{ij}, used just above. Because a synapse is the connection between two neurons, it takes two integers to specify a synapse. The subscript *i* indicates a specific input neuron, and the subscript *j* indicates a specific output neuron. In the present example, there are three possible synapses:

w_{11}, the synapse connecting input neuron 1 with output neuron 1;
w_{21}, the synapse between input neuron 2 and output neuron 1; and
w_{31}, the synapse between input neuron 3 and output neuron 1.

In the general case then, a synapse has the notation $w_{input,output}$ or w_{ij}. This notation allows us to talk precisely about a specific synapse out of a group of synapses.

Now we have the basic knowledge needed to code the computation performed by a neuron. Suppose a neuron has two input lines. You might think of these two inputs as distinct quantitative features, e.g., hair color and height. To distinguish these two features, they each must have a different name. Rather than hair color and height, call these inputs 1 and 2, respectively. Call the value of an input x_i where the subscript $i \in \{1,2\}$ and the variable $x_i \in [0,1)$[1]. Suppose there are also two output neurons. If y_j is the internal excitation of an output neuron, and $j \in \{1,2\}$, then $y_j = \sum_{i=1}^{2} x_i w_{ij}$ is the internal excitation of postsynaptic neuron *j* and the output of this neuron is $z_j = \begin{Bmatrix} 1 \text{ if } y_j \geq \text{threshold, otherwise} \\ 0 \end{Bmatrix}$. Note that this description defines a McCulloch-Pitts neuron: 1) linear excitation that changes with each input, 2) no memory for the past, and 3) a binary, {0,1}, output.

[1] We use the symbol $\in$ to mean "is an element of the set" when braces are used to define a set {}; that is, $x \in \{0,1\}$ means x can take on either zero or one. When a variable is continuous, brackets and/or parentheses are used, and $\in$ can be read as "lies in" or "lies within." For example, $x \in [0,1]$ means $0 \leq x \leq 1$ and $x \in [0,1)$ means $0 \leq x < 1$.

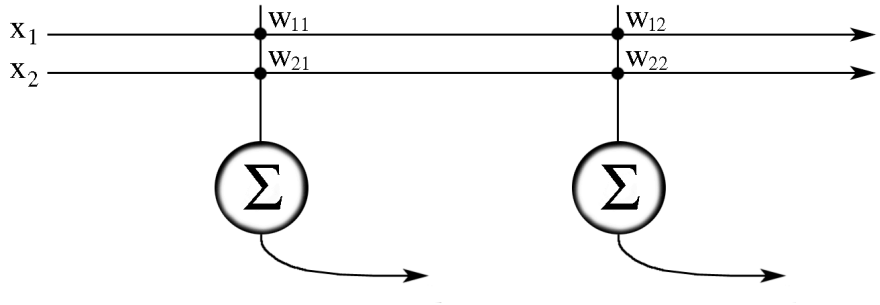

Fig. 2.8. A network of two inputs and two postsynaptic McCulloch-Pitts neurons. Note that each neuron has two synapses, one formed with each input.

To compute the excitation summation of our two-neuron network requires some parameterizations. Suppose the first input has a value of 0.8 and the second input has a value of 0.3. The strength of synapse w_{12} is 0.7 and that of w_{22} is 0.25. The threshold of output neuron 2 is 0.5. Given these parameters, $y_2 = \sum_{i=1}^{2} x_i w_{i2}$ and $y_2 = (0.8 \cdot 0.7) + (0.3 \cdot 0.25)$ or 0.635. If the threshold is set to 0.5, then y_2>0.5, and the output of neuron 2, z_2, equals 1. In other words, the internal excitation of neuron 2 exceeds its threshold, and neuron 2 fires an action potential. We similarly need to parameterize the other neuron.

Input and synaptic weight matrices

An *input environment* is simply a set (or sets) of input vectors that are used as an input to a network. In the sense that an animal lives in an environment that changes over its lifetime, so too does a network model live in an environment in which the input vectors change. Because a set of input vectors forms a matrix, we can also speak of an *input matrix*. Each column in an input matrix is a single input vector, and each row parameterizes successive states of that presynaptic axon.

Environments much simpler than the real world will be used here so that you can clearly understand how the simulations work.

MatLab's colon notation, e.g., x(:,5), can be used to select an input vector, at a single moment of time, from an input matrix. For example, suppose a postsynaptic neuron has four input synapses, and we want to generate five sequential input vectors that are random numbers uniformly drawn from the half open interval [0,1).

```
>>Mat4by5=rand(4,5)
                        %generate a matrix of 4 rows by 5 columns of random
                        values
```

It is possible to examine any one of the input vectors in an input matrix. Let's look at the third one in our example.

```
>>Mat4by5(:,3)
            %print all rows of column 3 to the screen
```

Note: for consistency, make sure that an input vector is a column vector.

We can specify the *synaptic weight matrix* in a similar fashion. Each column in the weight matrix is an ordered set of input synaptic weights. Each row is the ordered set of weights belonging to one presynaptic axon. Suppose five neurons each receive the same four inputs. In this case, we can create a 4 by 5 matrix where each successive column is the synaptic weight vector of one postsynaptic neuron.

```
>>Weights=rand(4,5)
```

Now we can look at the effect of a single vector input on each neuron's inner product excitation.

```
>>Mat4by5(:,1)'*Weights
            %finds the effect of the first input vector on all
            %five postsynaptic neurons. The answer is a row
            %vector.
```