Perceptrons

Primitive Artificial Neurons

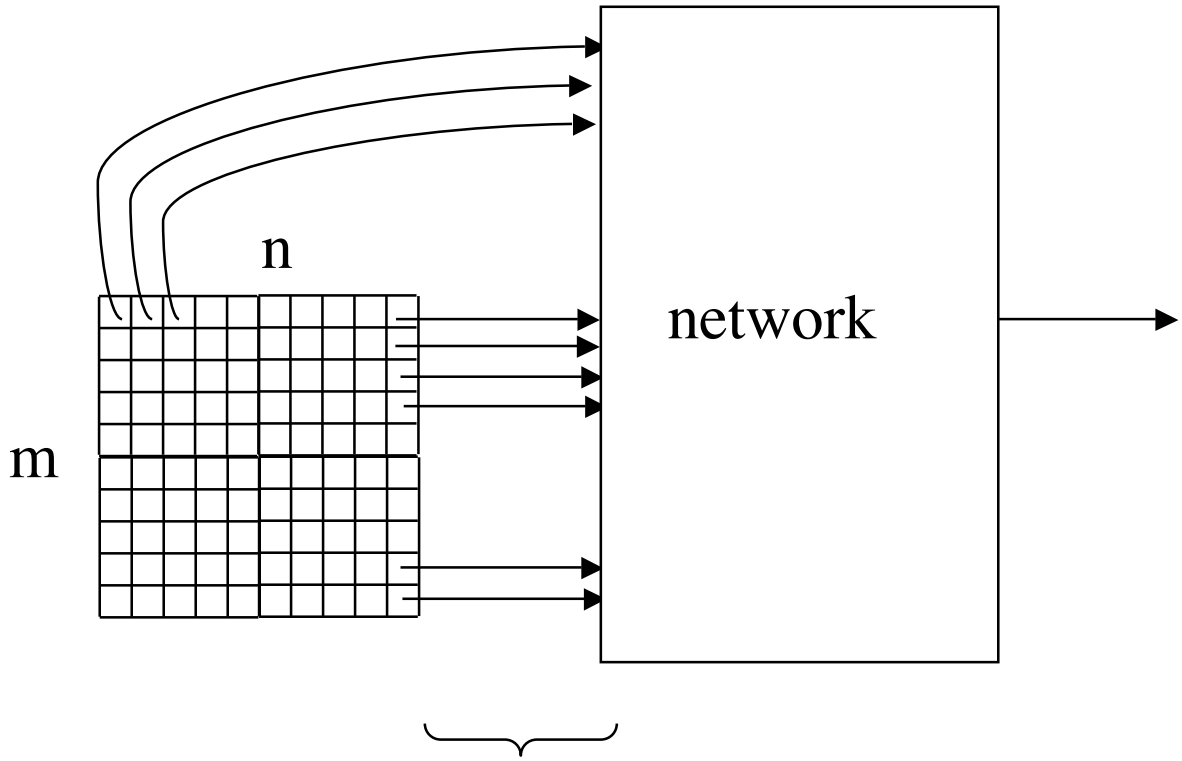
Rosenblatt's Perceptron, 1957

- Introduced the idea of **training**
- A **perceptron** is a linear threshold gate, possibly with real-number inputs, rather than just being limited to {0, 1}

Application for Perceptrons

- Classification problems:
  - Given a pre-specified set of inputs (not necessarily finite), is a given input in the set or not?
  - An input could be a retinal image

Retinal Image Classification



vector of m x n elements $\{x_i\}$, say gray values

Given a classification problem, try to find a perceptron to fit.

Find a vector of weights $\{w_i\}$ and a threshold θ, such that:

$$\text{output} = \begin{cases} 1 \text{ if } \sum w_i x_i > \theta \\ 0 \text{ otherwise} \end{cases}$$

$$= \begin{cases} 1 \text{ if } \{x_i\} \text{ represents a vector in the set} \\ 0 \text{ otherwise (not in the set)} \end{cases}$$

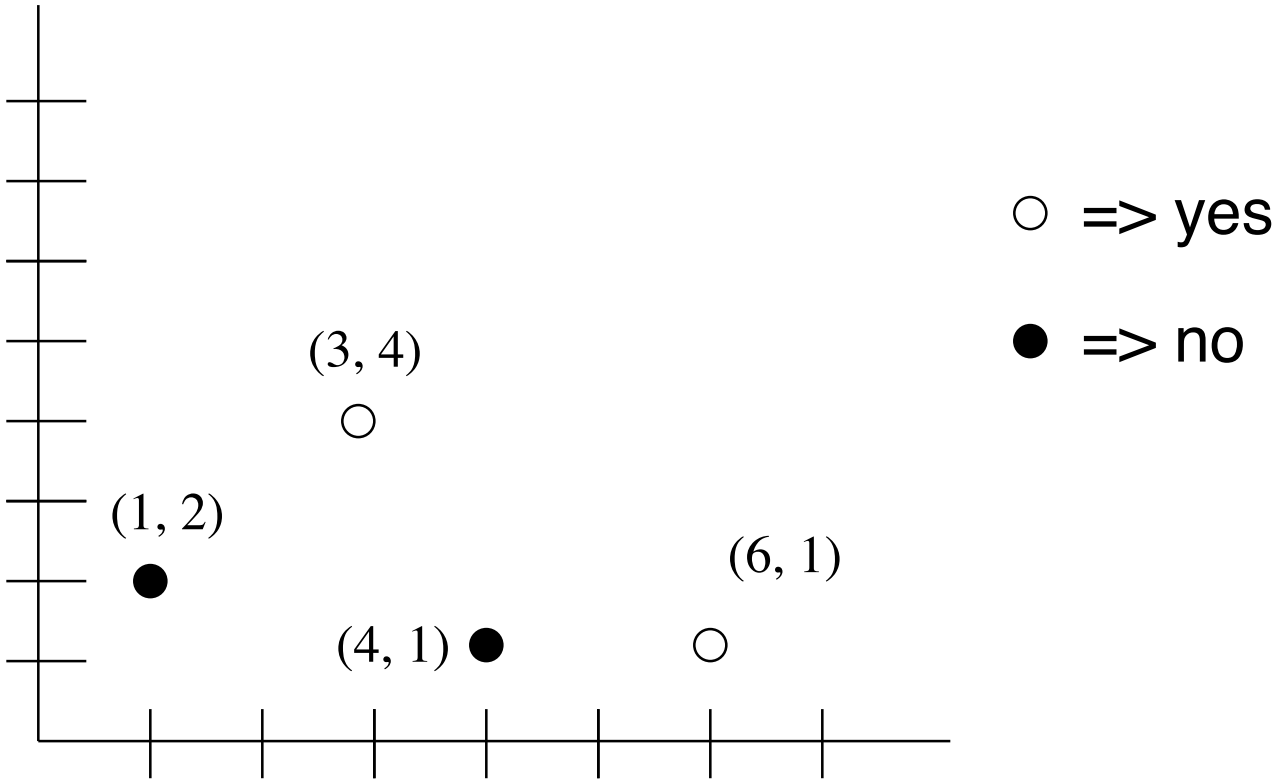
Issues

- **Existence**: Given a set, does there exist a perceptron that correctly classifies the set?
- **Solving**: Can the weights be found analytically?
- **Training**: Can the weights be adjusted simply by presenting examples that are either in, or not in, the set?

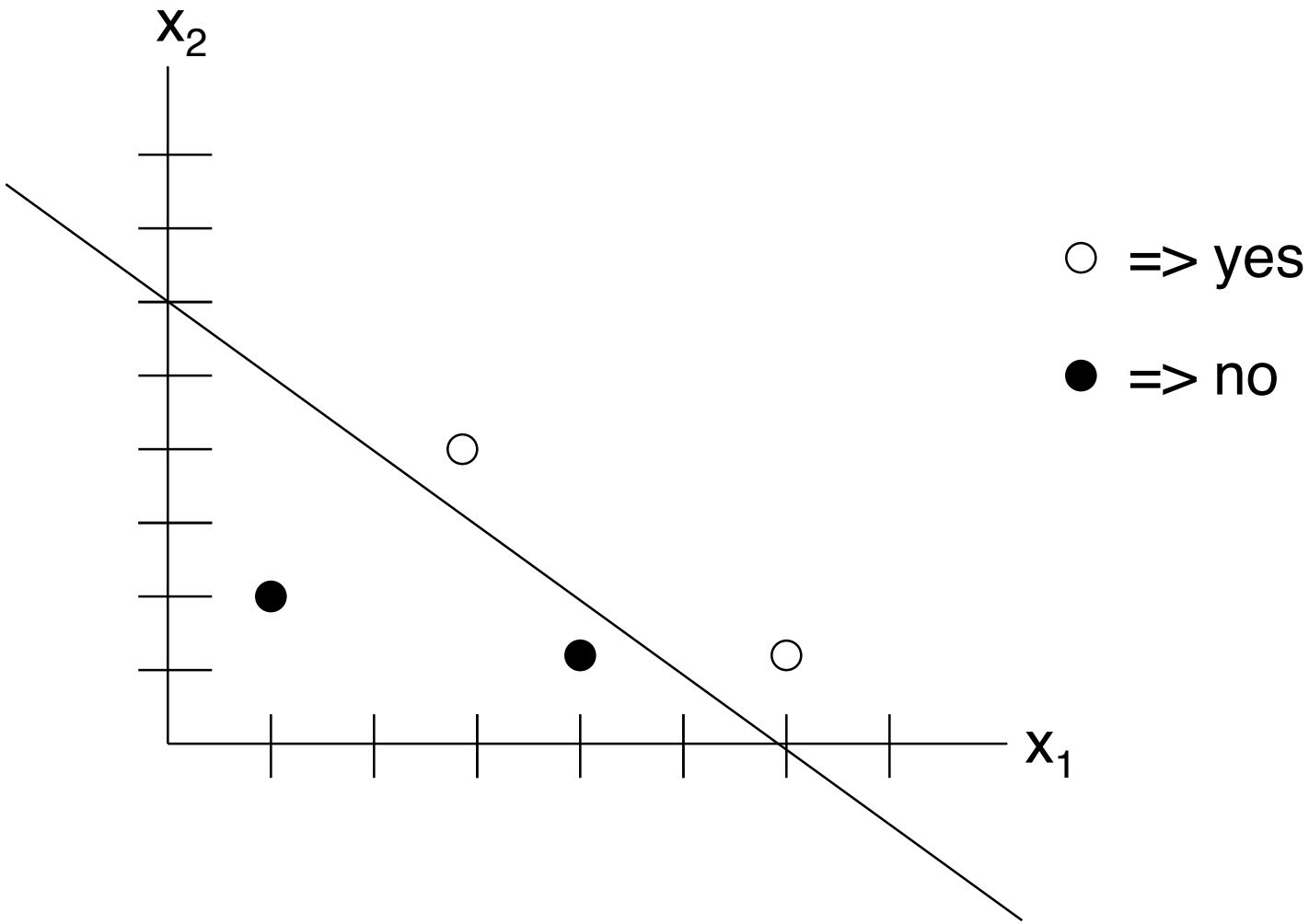
Example

- Suppose the signals are real-valued, and the following vectors are classified as shown:
  - (3, 4) yes (in the set)
  - (6, 1) yes
  - (4, 1) no (not in the set)
  - (1, 2) no
- Is there a perceptron that classifies the set as shown, and if so, what are its weights?

Geometric Insight



Try to
Locate a Separating Line



Separating Line Equation

- $x_1 + x_2 = 6$
- Points are
  - in the set if $x_1 + x_2 > 6$
  - not in the set if $x_1 + x_2 \leq 6$

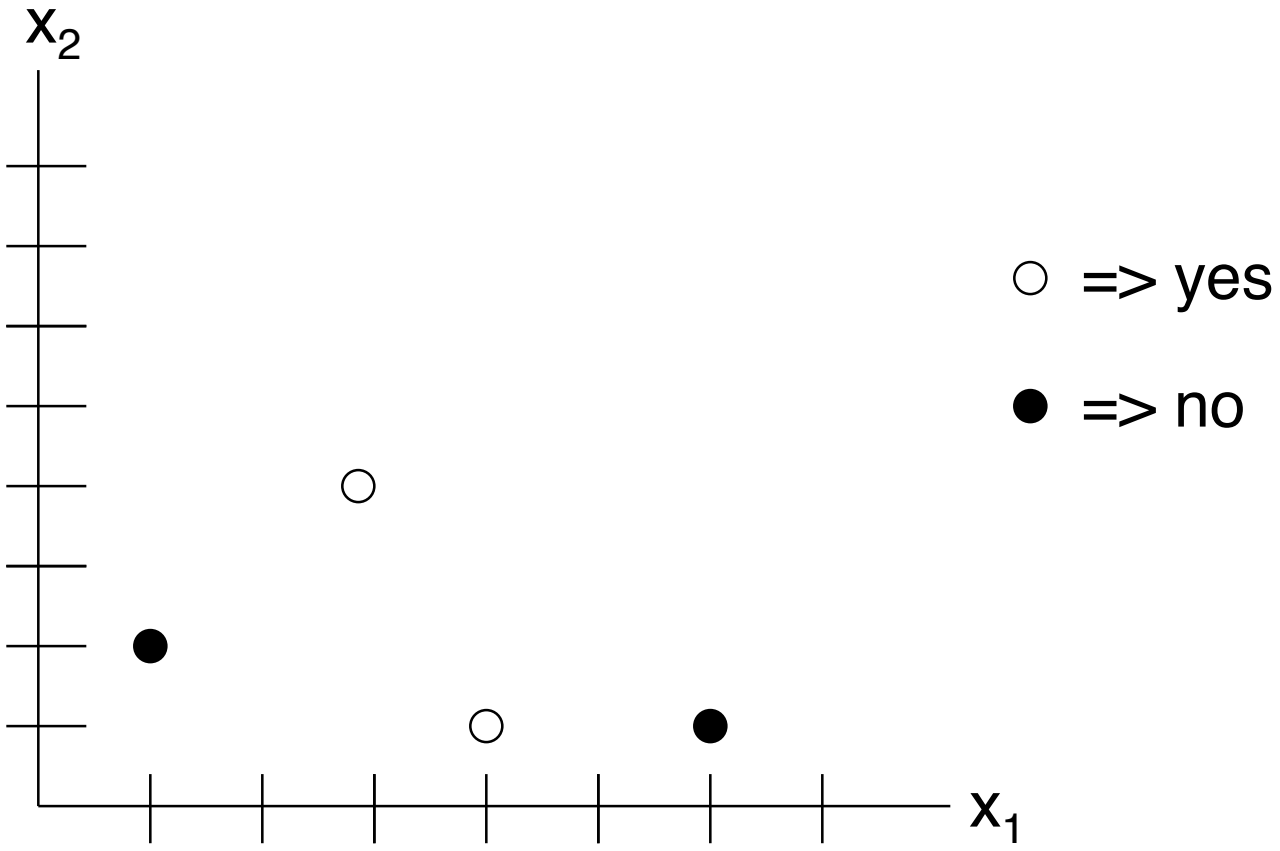
Checking the points

- (3, 4) yes $3 + 4 > 6$
- (6, 1) yes $6 + 1 > 6$
- (4, 1) no $4 + 1 < 6$
- (1, 2) no $1 + 2 < 6$

General Line Equation

- $w_1 x_1 + w_2 x_2 = \theta$
- Points are
  - in the set if $w_1 x_1 + w_2 x_2 > \theta$
  - not in the set if $w_1 x_1 + w_2 x_2 \leq \theta$
- Will $\{w_i\}$ and θ always exist?

Try to
Locate a Separating Line



Intuitively, no line exists in this case

- Can you prove it?

Generalizing to n dimensions

- Try to find $\{w_i\}$ and θ such that

  $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = \theta$
  separates the points:

  - $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta$ when $(x_1, x_2, \ldots, x_n)$ is in the set

  - $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n < \theta$ when $(x_1, x_2, \ldots, x_n)$ is not in the set

  - [for now, ignore the possibility of $= \theta$; revisit this later.]

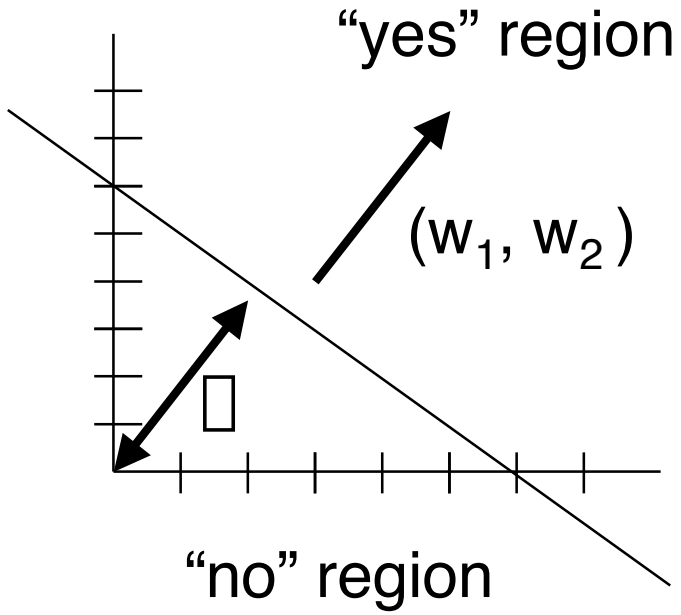
Separating Hyperplane

- The equation
  $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = \square$
  defines a *hyperplane* in n-space

- If such a hyperplane exists for a classification problem, the problem is called **linearly-separable**.

Geometry

- The vector of weights (w_1, w_2 , …. w_n) is normal (**perpendicular**) to the hyperplane.
- Threshold is the distance of the hyperplane from the origin.



Perceptron Summary

- A perceptron can solve a classification problem iff the problem is linearly-separable.
- There are problems a single perceptron cannot solve.
- Perhaps the simplest unsolvable one is the **XOR problem**:

  yes: {(0, 1), (1, 0)}, no: {(0, 0), (1, 1)}

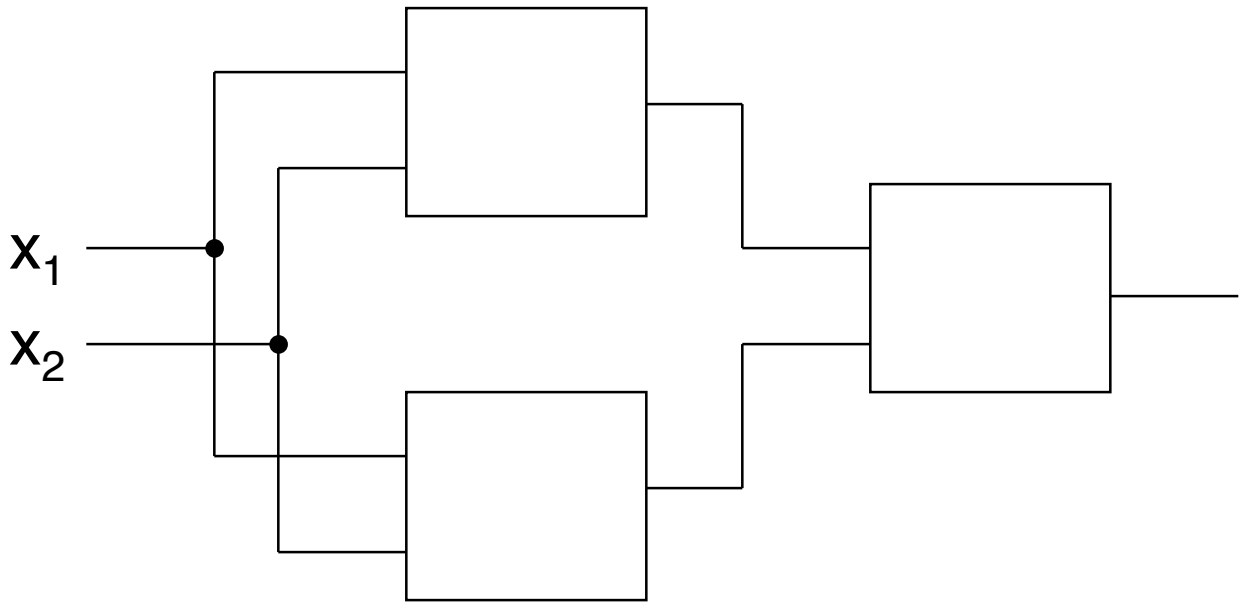
Some Questions to Ask

- Which switching functions are linearly separable?
- Are there any closure properties (such as closure under union or intersection) for linear separability?

Multi-level Perceptrons

- One way to alleviate the limitations of a single perceptron is to build *networks* of perceptrons.
- Another might be to allow non-linear expressions (such as squaring).
- Can the XOR problem be solved by such models?

What weights and thresholds for XOR?



General 1-Perceptron Training

- Assuming weights exist for a problem, how to find them?
  - Analytic? (technically not “training”)
    A conventional perceptron doesn’t lend well to analytic solution, due to the discontinuous nature of the function.
  - Successive approximations?

A Somewhat General Approach

- Use a set of **training samples**:
  - Input vectors, each paired with desired output
- Choose a network structure.
- Initialize the weights arbitrarily.
- Repeat:
  - Choose a sample
  - Test the network against the sample
  - If answer is incorrect, **adjust** weights
- until all samples test correctly.

Omitted Details

- How to choose a sample?
- How to adjust the weights?

Sample Choice

- Method 1: Simply cycle through all of the samples in a fixed order.
- Method 2: Choose samples randomly, but hopefully with some assurance that each will be checked eventually.

Weight Adjustment

- The Perceptron learning rule (Rosenblatt):
  - If the perceptron gives the correct answer, do nothing.
  - If the perceptron gives the wrong answer, adjust the weights and threshold “in the right direction”, so that it eventually gives the right answer.

When does a Perceptron give the wrong answer?

- Correct answers:

  $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta$ when $(x_1, x_2, \ldots, x_n)$ is in the set;

  $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n < \theta$ when $(x_1, x_2, \ldots, x_n)$ is *not* in the set

When does a Perceptron give the wrong answer?

- *Wrong* answers:

  $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta$ when $(x_1, x_2, \ldots, x_n)$ is *not* in the set;

  $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n < \theta$ when $(x_1, x_2, \ldots, x_n)$ is in the set

Desired Output Value

- Let

  $d(x_1, x_2, \ldots, x_n) =$

  1 if $(x_1, x_2, \ldots, x_n)$ is in the set,

  0 otherwise.

Actual Output Value

- Let

  $a(x_1, x_2, \ldots, x_n) =$

  1 if $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta$

  0 otherwise.

- Note that a is an implied function of the weights.

Error Value

- We can capture correct vs. incorrect answers succinctly by introducing an *error value* □:
- □ = $d(x_1, x_2, \ldots, x_n) - a(x_1, x_2, \ldots, x_n)$
- So that
  - □ = 0 when the correct answer is given
  - □ = 1 when $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n <$ □ but $(x_1, x_2, \ldots, x_n)$ is in the set
  - □ = -1 when $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n >$ □ but $(x_1, x_2, \ldots, x_n)$ is *not* in the set

Simplification: Threshold conversion

- The threshold can be treated as if one of the weights by introducing a “phantom” input of -1:

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta$$

iff

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n - \theta > 0$$

iff

$$w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0$$

where w_0 is defined to be θ and $x_0 = -1$.

Perceptron training (continued)

- Wrong answer of the first type ($\square = 1$):
  $w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n < 0$
  when $(x_1, x_2, \ldots, x_n)$ is in the set

- i.e., $w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$ is **too low**.

- To correct for this, we need to make the sum higher.

Perceptron training (continued)

- Wrong answer of second type ($\delta = -1$):
  $w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0$
  when $(x_1, x_2, \ldots, x_n)$ is *not* in the set

- i.e., $w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$ is **too high**.

- To correct for this, we need to make the sum lower.

Perceptron training (continued)

- Make $w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$ lower or higher by adjusting weights.
  - $\delta = 1$: Make each contribution $w_i x_i$ higher
  - $\delta = -1$: Make each contribution $w_i x_i$ lower
- In either case, can ***add*** some multiple δ of δx_i to w_i to get the desired effect.

Perceptron training (continued)

- *Add* ηx_i from w_i to get the desired effect:

$$(w_i + \eta x_i)\, x_i = (w_i x_i + \eta x_i^2)$$
$$\leq w_i x_i \quad \text{if } \eta > 0$$
$$\geq w_i x_i \quad \text{if } \eta < 0$$

- η is called the "learning rate".

Learning Rate η

- η governs the rate at which the training rule converges toward the correct solution.
- Typically $\eta \leq 1$.
- Too small an η produces slow convergence.
- Too large of an η can cause oscillations in the process.

Example

- Train a perceptron to classify according to:
  - (4, 5) yes
  - (6, 1) yes
  - (4, 1) no
  - (1, 2) no
- There will be three weights (w_0, w_1, w_2) where is the w_0 threshold, corresponding to phantom input -1.
- Start with “random” weights, say (0, +1, -1)
- Choose $\eta = 1$.

Perceptron Training Example
One Pass over Data Samples

Fill out this table sequentially:

weights	input	desired	actual	error	new weights
(0, 1, -1)	(-1, 4, 5)	yes			
	(-1, 6, 1)	yes			
	(-1, 4, 1)	no			
	(-1, 1, 2)	no			

Perceptron Training Example
One Pass over Data Samples

weights	input	desired	actual	error	new weights
(0, 1, -1)	(-1, 4, 5)	yes	no	1	(-1, 5, 4)
(-1, 5, 4)	(-1, 6, 1)	yes	yes	0	no change
(-1, 5, 4)	(-1, 4, 1)	no	yes	-1	(0, 1, 3)
(0, 1, 3)	(-1, 1, 2)	no	yes	-1	(1, 0, 1)

Perceptron Training Example
Second Pass over Data Samples

weights	input	desired	actual	error	new weights
(1, 0, 1)	(-1, 4, 5)	yes			
	(-1, 6, 1)	yes			
	(-1, 4, 1)	no			
	(-1, 1, 2)	no			

Perceptron Training Example
Second Pass over Data Samples

weights	input	desired	actual	error	new weights
(1, 0, 1)	(-1, 4, 5)	yes	yes	0	no change
(1, 0, 1)	(-1, 6, 1)	yes	? no	1	(0, 6, 2)
(0, 6, 2)	(-1, 4, 1)	no	yes	-1	(1, 2, 1)
(1, 2, 1)	(-1, 1, 2)	no	yes	-1	(2, 1, -1)

Perceptron Training Example
Third Epoch

weights	input	desired	actual	error	new weights
(2, 1, -1)	(-1, 4, 5)	yes	no	1	(1, 5, 4)
(1, 5, 4)	(-1, 6, 1)	yes	yes	0	no change
(1, 5, 4)	(-1, 4, 1)	no	yes	-1	(2, 1, 3)
(2, 1, 3)	(-1, 1, 2)	no	yes	-1	(3, 0, 1)

Perceptron Training Example
Epoch 4

weights	input	desired	actual	error	new weights
(3, 0, 1)	(-1, 4, 5)	yes	yes	0	no change
(3, 0, 1)	(-1, 6, 1)	yes	no	1	(2, 6, 2)
(2, 6, 2)	(-1, 4, 1)	no	yes	-1	(3, 2, 1)
(3, 2, 1)	(-1, 1, 2)	no	yes	-1	(4, 1, -1)

Perceptron Training Example
Epoch 5

weights	input	desired	actual	error	new weights
(4, 1, -1)	(-1, 4, 5)	yes	no	1	(3, 5, 4)
(3, 5, 4)	(-1, 6, 1)	yes	yes	0	no change
(3, 5, 4)	(-1, 4, 1)	no	yes	-1	(4, 1, 3)
(4, 1, 3)	(-1, 1, 2)	no	yes	-1	(5, 0, 1)

Perceptron Training Example
Epoch 6

weights	input	desired	actual	error	new weights
(5, 0, 1)	(-1, 4, 5)	yes	no	1	(4, 4, 6)
(4, 4, 6)	(-1, 6, 1)	yes	yes	0	no change
(4, 4, 6)	(-1, 4, 1)	no	yes	-1	(5, 0, 5)
(5, 0, 5)	(-1, 1, 2)	no	yes	-1	(6, -1, 3)

Perceptron Training Example
Epoch 7

weights	input	desired	actual	error	new weights
(6, -1, 3)	(-1, 4, 5)	yes	yes	0	no change
(6, -1, 3)	(-1, 6, 1)	yes	no	1	(5, 5, 4)
(5, 5, 4)	(-1, 4, 1)	no	yes	-1	(6, 1, 3)
(6, 1, 3)	(-1, 1, 2)	no	yes	-1	(7, 0, 1)

Perceptron Training Example
Epoch 8

weights	input	desired	actual	error	new weights
(7, 0, 1)	(-1, 4, 5)	yes	no	1	(6, 4, 6)
(6, 4, 6)	(-1, 6, 1)	yes	yes	0	no change
(6, 4, 6)	(-1, 4, 1)	no	yes	-1	(7, 0, 5)
(7, 0, 5)	(-1, 1, 2)	no	yes	-1	(8, -1, 3)

Perceptron Training Example
Epoch 9

weights	input	desired	actual	error	new weights
(8, -1, 3)	(-1, 4, 5)	yes	yes	0	no change
(8, -1, 3)	(-1, 6, 1)	yes	no	1	(7, 5, 2)
(7, 5, 2)	(-1, 4, 1)	no	yes	-1	(8, 1, 3)
(8, 1, 3)	(-1, 1, 2)	no	no	0	(8, 1, 3)

Perceptron Training Example
Epoch 10

weights	input	desired	actual	error	new weights
(8, 1, 3)	(-1, 4, 5)	yes	yes	0	no change
(8, 1, 3)	(-1, 6, 1)	yes	yes	0	no change
(8, 1, 3)	(-1, 4, 1)	no	no	0	no change
(8, 1, 3)	(-1, 1, 2)	no	no	0	no change

Perceptron Training Example Conclusion

- A perceptron with weights (8, 1, 3) correctly classifies all inputs.
- The "yes" criterion is therefore:
  $-8 + x_1 + 3\,x_2 > 0$ [i.e. $x_1 + 3\,x_2 > 8$]
- Check:

input x1, x2	desired	actual
(4, 5)	yes	yes
(6, 1)	yes	yes
(4, 1)	no	no
(1, 2)	no	no

Perceptron Training Algorithm (1)

- Inputs:
  - A list of training samples, each of the form
    - $[d(x_1, x_2, \ldots, x_n), -1, x_1, x_2, \ldots, x_n]$
    - (d is the desired output, -1 the phantom input)
  - An initial weight vector $[w_0, w_1, w_2, \ldots, w_n]$
    - (w_0 is the threshold)
  - A learning rate η

Perceptron Training Algorithm (2)

- Outputs:
  - If the set of samples is linearly separable, a vector of weights $[w_0, w_1, w_2, \ldots, w_n]$ such that with these weights the perception properly separates the training samples.
  - If the set of samples is not linearly separable, then the algorithm diverges.

Perceptron Training Algorithm (3)

- Operation:
  - Set $[w_0, w_1, w_2, \ldots, w_n]$ = initial weights;
  - **while**(there is a sample not correctly classified)
    - Let $[d(x_1, x_2, \ldots, x_n), -1, x_1, x_2, \ldots, x_n]$ be an incorrectly classified sample.
    - Let $\Delta = d(x_1, x_2, \ldots, x_n) - a(x_1, x_2, \ldots, x_n)$, where $a(x_1, x_2, \ldots, x_n) = (\sum w_i x_i > 0)$.
    - Vector-add to $[w_0, w_1, w_2, \ldots, w_n]$ the vector $\Delta w = \Delta [-1, x_1, x_2, \ldots, x_n]$

Perceptron learning rule

Perceptron Training Algorithm Modifications for practical usage (1)

- Put a ***limit*** on the number of iterations, so that the algorithm will terminate even if the sample set is not linearly separable.
- Include an ***error bound*** as an extra input. The algorithm can stop as soon as the portion of mis-classified samples is less than this bound (as opposed to requiring perfect classification, which would be an error bound of 0).
- Generate the initial weights ***randomly***, so that the user does not have to specify them.

Perceptron Training Algorithm Modifications for practical usage (2)

- Don't require the user to specify the learning rate.
- Instead, vary it so that the chosen sample is always correctly classified after weight modification (although other samples may become incorrectly classified).
  We want to change the weight vector so that vector adding $\Delta w = \eta [-1, x_1, x_2, \ldots, x_n]$ causes $(\sum w_i x_i > 0)$ to become 0.
  Effectively choosing η to be just larger than
  $|wx| / |x x|$
  (where the products are vector inner products) will guarantee this, since
  $(w+\Delta w)x = (w+\eta x)x = wx+\eta xx$.

Perceptron Convergence Theorem (1)

If the sample set is linearly separable, then the perceptron algorithm will always converge, even if η is fixed at 1.

- Proof: Suppose that the sample set is linearly separable. Then let w* be a weight vector that separates the samples.
- We can assume ||w*|| = 1, since if w* separates, then so does w*/ ||w*|| and the latter vector has length 1.

Perceptron Convergence Theorem (2)

- We can assume, without loss of generality, that all samples are in the "yes" set, for if not, we can replace that sample with its negative ($\sum w_i x_i < 0$) iff ($\sum -w_i x_i > 0$).

- Let $x^1, x^2, x^3, \ldots, x^k$ be the mis-classified samples in a training sequence, i.e. the ones that are used to adjust the weights.

- Let w^k be the resulting weight vector, with w^0 as the initial weight vector.

Perceptron Convergence Theorem (3)

- Since the samples are all in the “yes” set, the error $\Delta = 1$ always (as opposed to -1), so

  $$w^k = w^0 + x^1 + x^2 + x^3 + \ldots + x^k$$

- Multiplying both sides by w*:

  $$w^* w^k = w^* w^0 + w^* x^1 + w^* x^2 + w^* x^3 + \ldots + w^* x^k$$

  the products being inner vector products.

Perceptron Convergence Theorem (4)

- Since the x^k are all in the "yes" set, each inner product on the right is > 0 (because w* is a correct classifier):

$$w^* w^k = w^* w^0 + w^* x^1 + w^* x^2 + w^* x^3 + \ldots + w^* x^k$$

- So there is a value $\delta > 0$ such that for each i:

$$w^* w^i > \delta$$

which gives

$$w^* w^k > w^* w^0 + k\delta$$

Perceptron Convergence Theorem (5)

- Now look at the ***squares*** of the $\|w^k\|$.
- The Cauchy-Schwarz inequality tells us
  $$\|w^*\|^2 \, \|w^k\|^2 > (w^* w^k)^2$$
  and since $\|w^*\|^2 = 1$, we have
  $$\|w^k\|^2 > (w^* w^k)^2.$$

- Substituting for $w^* w^k$, where $w^* w^k > w^* w^0 + k\delta$, we get

  $$\|w^k\|^2 > (w^* w^0 + k\delta)^2$$
  i.e. the sequence of terms $\|w^k\|^2$ grows *at least quadratically* with k.

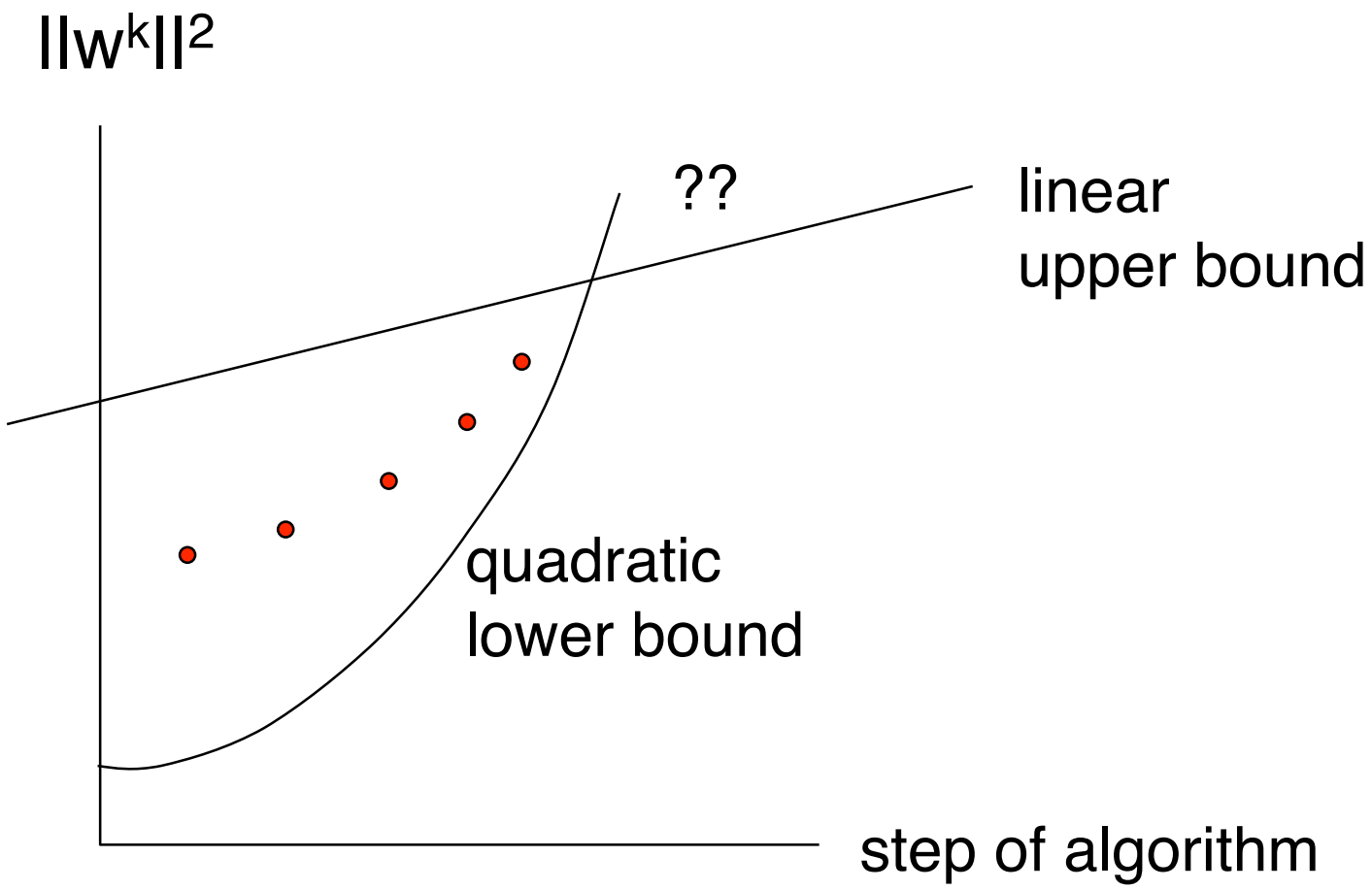
Perceptron Convergence Theorem (6)

- The sequence of terms $\|w^k\|^2$ grows ***at least quadratically*** with k.

- Pivotal idea:

  If we can now show that the *same* sequence grows ***at most linearly***, we will have established that the sequence must be finite, it cannot go on forever.

Perceptron Convergence Theorem (7)



Perceptron Convergence Theorem (8)

- Since $w^k = w^{k-1} + x^k$,
  $\|w^k\|^2 = w^k w^k$
  $= w^{k-1} w^{k-1} + 2\, w^{k-1} x^k + x^k x^k$
- But since the x^k are misclassified and in the "yes" category, $w^{k-1} x^k < 0$.
  Therefore
  $\|w^k\|^2 < \|w^{k-1}\|^2 + \|x^k\|^2$
- We telescope-sum these k inequalities to get
  $\|w^k\|^2 < \|w^0\|^2 + k \max_k \|x^k\|^2$
  constant,
  since the set of samples is finite.

  which is our *linear* upper bound on $\|w^k\|^2$.

■

Handling Multiple Outputs

- Occasionally problems will have more than two distinct output values.
- In order to fit a perceptron to such a task, the designer must map the set of output values into **vectors** of 0's and 1's.
- This can be accomplished in many ways, but using a "one-hot" encoding is probably the safest, although not the cheapest in terms of neurons.

Issue of equality in linear separability

- We stated the decision criterion for the Perceptron in terms of strict inequalities < and >. But as we know, it might happen that the sum of the weights = the threshold.

- For a given set of samples, I claim that if there is set of weights that separates with equality allowed, there is also a set that separates without using =.
- Intuitively, we only need to “nudge” the separating hyperplane so that no sample points lie on it.

- Therefore no generality has been lost in the assumption of strict inequality.
- Practically, the = can be resolved by grouping it with either < or > consistently.

Types of Data

- Numerical: Ordering makes sense
- Categorical: No natural ordering
- To use other than a one-hot encoding for categorical data is to suggest that there is a numerical ordering when there really is none.
- Choosing an arbitrary ordering is not likely to work for training perceptrons, except in special cases.

One-Hot Implementation

- A one-hot encoding can be implemented by training a separate perceptron for each bit of the encoding.

- In effect, the weight vector becomes a weight matrix, with one row per perceptron and one column per input, as in the one-perceptron case.

Weight Vector/Matrix

- $[w_0, w_1, w_2, \ldots, w_n]$ 1 perceptron

- $\begin{pmatrix} w_{10}, & w_{11}, w_{12}, & \ldots, w_{1n} \\ w_{20}, & w_{21}, w_{22}, & \ldots, w_{2n} \\ & \ldots & \\ w_{m0}, & w_{m1}, w_{m2}, & \ldots, w_{mn} \end{pmatrix}$ m perceptrons

Some Unhappiness About Perceptron Training

- When a perceptron gives the right answer, no learning takes place.
- Anything below the threshold is interpreted as “no”, even if it *just* below the threshold.
- Might it be better to train the neuron based on *how far* below the threshold it is?
- This idea is developed in the Adaline training algorithm.