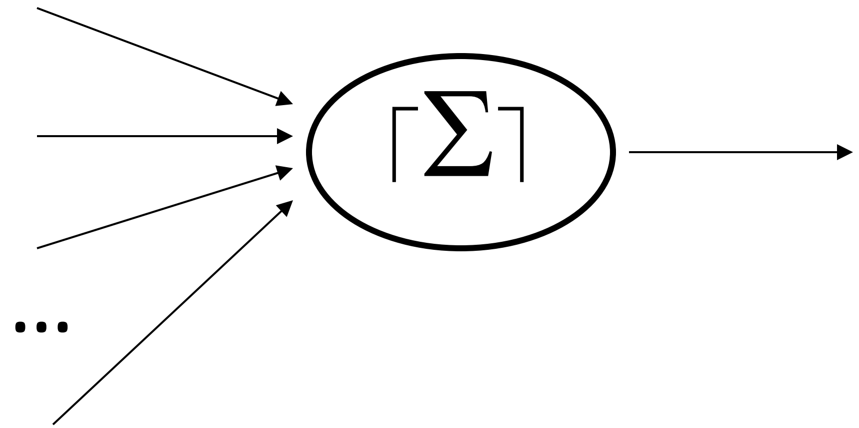
# Perceptrons and Artificial Neural Networks

- Perceptron is a linear threshold device
- Very simple; quite old
- Other linear decision hypotheses
  - Naïve Bayes
  - Logistic regression
  - LDA, …
- Low capacity; easily learned
- Basis for standard Artificial Neural Nets

# Perceptron

- Rosenblatt ~1960
- Analog of a neuron
  McCulloch & Pitts (1943),
  Hebb (1949),
  Widrow (1960)…
  Minsky & Papert (1969)
  Rumelhart, Hinton,… ~1985
- Switching device
- Weighted sum of inputs
- Compare to threshold



# Two Important Questions

- Perceptrons form a hypothesis space
  - uncountably infinite in number
  - low in capacity
- Representation
  - Characterize the expressiveness of perceptrons
  - Which functions of the inputs are in the space?
- Learnability
  - Given an acceptable set of training examples
  - Can we efficiently find a perceptron?

# Perceptron Example Space

The input is a vector of n numeric components

The dimensionality n can be quite large…

If input is a vector of Booleans

- X is the n-dimensional Boolean hypercube

If the input is a vector of real numbers

- X is n-dimensional real space $\Re^n$

# Perceptron Decision Boundary

Compare weighted sum of inputs
to a threshold

$$\sum_{i=1}^{n} w_i \cdot x_i > \theta$$

Without loss of generality
set $x_0$= -1 then $w_0$ is $\theta$

$$\sum_{i=0}^{n} w_i \cdot x_i > 0$$

This defines a decision surface

$$\sum_{i=0}^{n} w_i \cdot x_i = \mathbf{w} \cdot \mathbf{x} = 0$$

Which is the equation of
a homogeneous hyperplane in n+1 dimensions

# Perceptron Hypothesis Space

- Perceptron as a classifier
  - An input **x** is labeled
    - Positive + if it is above the oriented hyperplane,
    - else Negative –
- Parameterized family of functions
- Each function: **x** $\rightarrow$ {+,-}
- Family is parameterized by **w**
- So the hypothesis space is…
- The set of linearly bounded half-spaces (in n dimensions)
- And the VC dimension is…
- n + 1

# Perceptron Concepts

Given any set of points in $\Re^n$

Given any assignment of + / -

For which sets is there a perceptron that is consistent with them?

(This is a *representation* question, not a *utility* question; a concept might be useful even if it makes some mistakes)

IFF the +’s are linearly separable from the –’s

How many such perceptrons will there be?

# Perceptron Concepts

Even though an infinite supply of different perceptron hypotheses, there are many + / - configurations that we cannot handle

Given a perceptron, we are committed to a labeling of all possible inputs

# Perceptron Learning

Given a training set of linearly separable
+ / - labeled points

Can we find a consistent perceptron efficiently?

Trying out lots of **w**'s is not efficient

Thought question:

<How good will it be on future points?>

<Remember we do NOT care (directly) about getting the training set right>

# Perceptron Learning

(Widrow-Hoff or delta rule)

$\text{percep}_{\mathbf{w}}(\mathbf{x})$ assigns 1 for + if $\mathbf{w} \cdot \mathbf{x} > 0$ (vector dot product)
else it assigns 0 for - (a common numeric transform trick)

err = label($\mathbf{x}$) – $\text{percep}_{\mathbf{w}}(\mathbf{x})$
0: correct -1: false pos 1: false neg

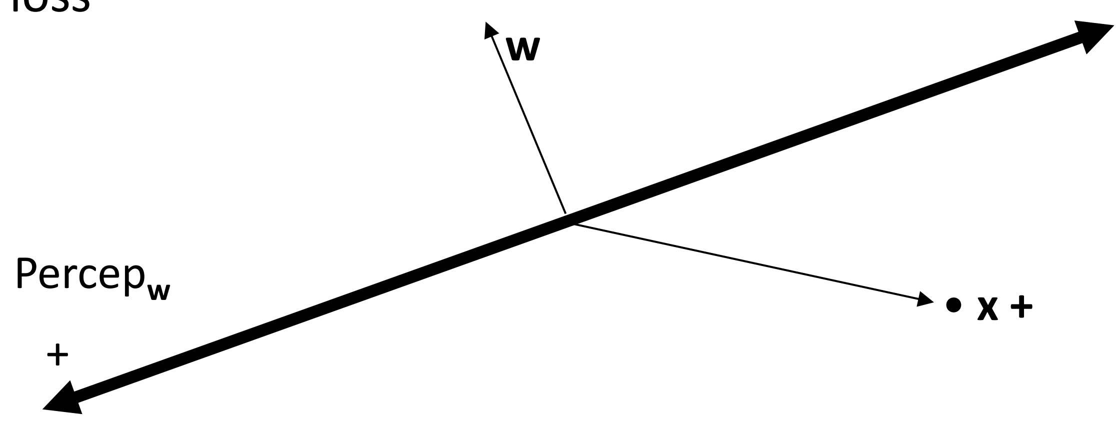
Here, false neg: $\mathbf{w} \cdot \mathbf{x} < 0$ but it should be $> 0$

loss = distance from boundary = - err $\mathbf{w} \cdot \mathbf{x}$

Want to adjust $w_i$'s to reduce this loss

Loss fcn *gradient* is direction of greatest increase in loss with $\mathbf{w}$

Want the opposite: step $\mathbf{w}$ in direction $-\nabla_w$ loss



# Perceptron Learning

(Widrow-Hoff or delta rule)

Loss function = - err $\mathbf{w} \cdot \mathbf{x}$

Want the opposite: step $\mathbf{w}$ in direction $-\nabla_w$ loss

What is $\nabla_w$ loss for input $\mathbf{x}$?

View - err $\mathbf{w} \cdot \mathbf{x}$ as a function of $\mathbf{w}$

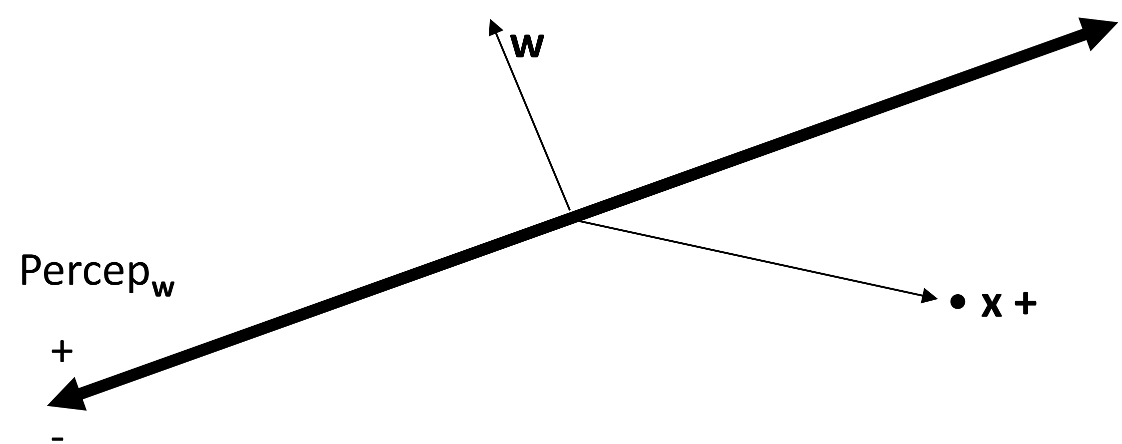
$\nabla_w$ (- err $\mathbf{w} \cdot \mathbf{x}$) = - err $\mathbf{x}$

So $-\nabla_w$ (- err $\mathbf{w} \cdot \mathbf{x}$) = err $\mathbf{x}$

Update $\mathbf{w}$ according to:

$$\Delta\mathbf{w} = \alpha \text{ err } \mathbf{x}$$

where $\alpha$ is a learning rate



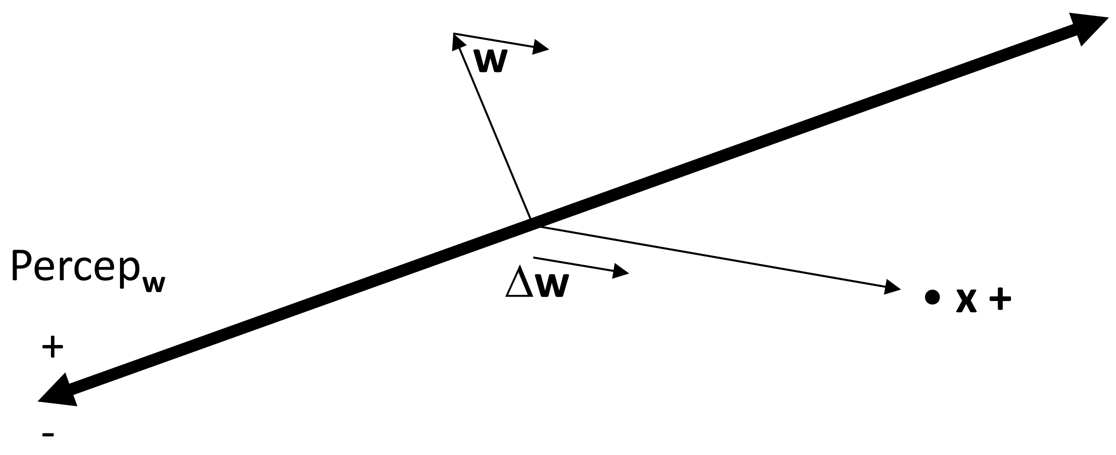
# Perceptron Learning

(Widrow-Hoff or delta rule)

Choose a learning rate $\alpha$

Compute $\Delta\mathbf{w} = \alpha \text{ err } \mathbf{x}$

Add $\Delta\mathbf{w}$ to $\mathbf{w}$



# Perceptron Learning

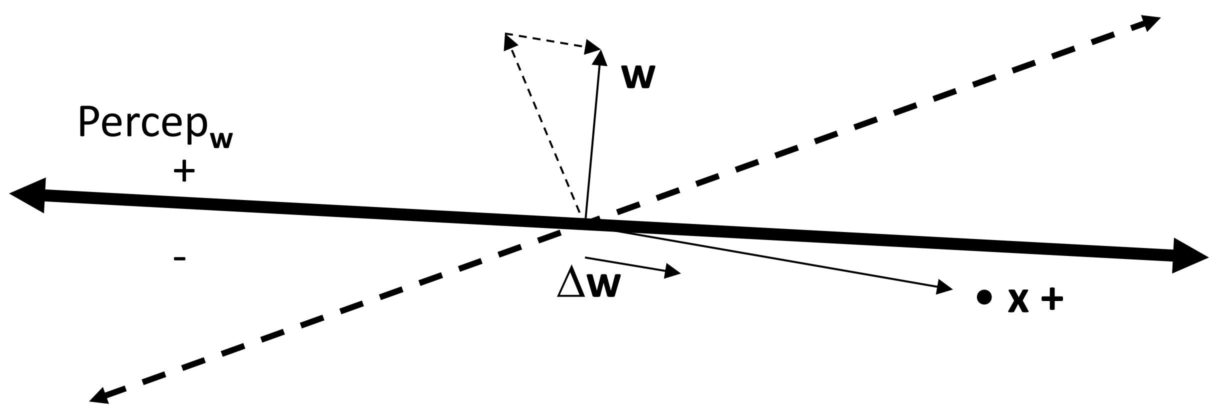
(Widrow-Hoff or delta rule)

Choose a learning rate $\alpha$; initialize **w** arbitrarily (small works best)

Compute $\Delta\mathbf{w} = \alpha$ err $\mathbf{x}$

Add $\Delta\mathbf{w}$ to **w**

Repeatedly cycle through training examples




New perceptron twists to reduce error
If **x** were a false positive, **w** would twist the other way

# Perceptron Learning

(Widrow-Hoff or delta rule)

If the points are linearly separable, the algorithm

a) will halt

b) will find a separator

(The celebrated Perceptron Convergence Theorem)

# iClicker!

## Perceptron Learning with Widrow-Hoff delta rule

If the training points are *not* linearly separable, the algorithm:

A. May not halt

B. Will not halt

C. Will halt but may not correctly label all of the training data

D. Will halt but converge much more slowly

E. None of the above

# iClicker!

## Perceptron Learning with Widrow-Hoff delta rule

If the points are not linearly separable, the algorithm

B. Will not halt

WHY?

This is stochastic gradient descent

Bad if there is finite data and label noise

One alternative: decay $\alpha$ ( $\sum\alpha$ & $\sum\alpha^2$ as in RL)

Another alternative: accumulate $\Delta\mathbf{w}$ over an *epoch* (one pass through the training set)

# Epoch Perceptron Learning

Choose a convergence criterion (#epochs, min $|\Delta\mathbf{w}|$, …)
Choose a learning rate $\alpha$, an initial $\mathbf{w}$

Repeat until converged:

$\Delta\mathbf{w} = \sum_{\mathbf{x}} \alpha \text{ err } \mathbf{x}$ (sum over training set holding $\mathbf{w}$)

$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$ (update with accumulated changes)

Now it always converges

regardless of $\alpha$ (will influence the rate)

Whether or not training points are linearly separable

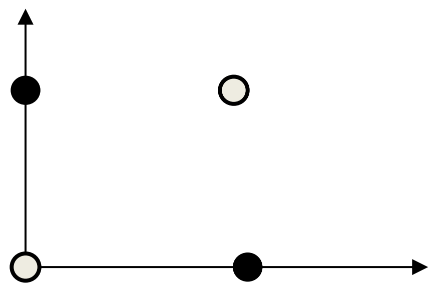
This is gradient descent (w/ estimated gradient)

It may not result in the fewest misclassified **x**'s (highest accuracy)

WHY?

# Perceptron Learning Algorithm

- Very limited expressiveness
- Cannot learn XOR on two Booleans:

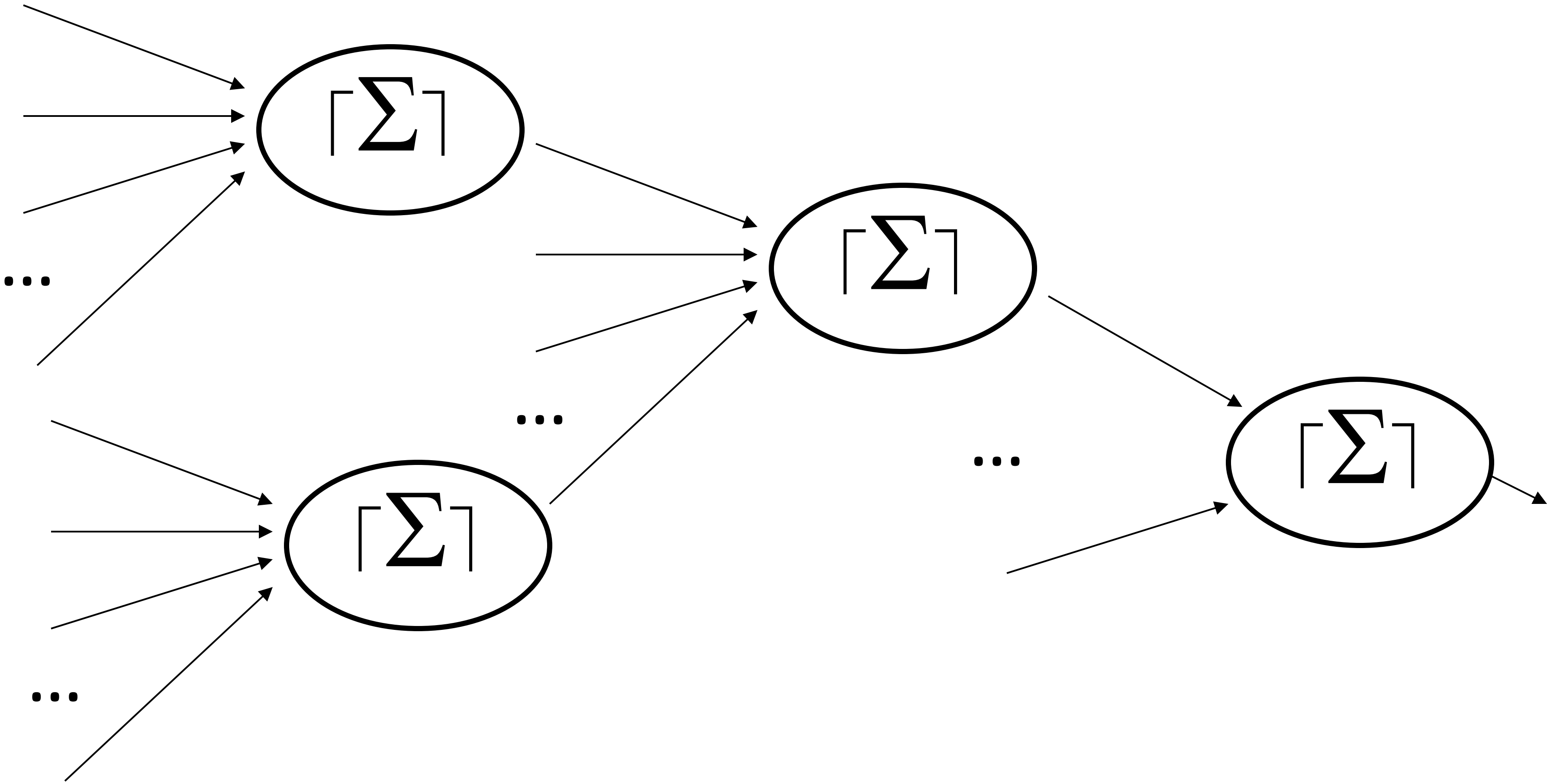


- This kind of limitation killed interest for years
- (But popular again – we'll see why later)
- If only we could layer them
- …meaning?
- What functions could we represent?

# Artificial Neural Networks:

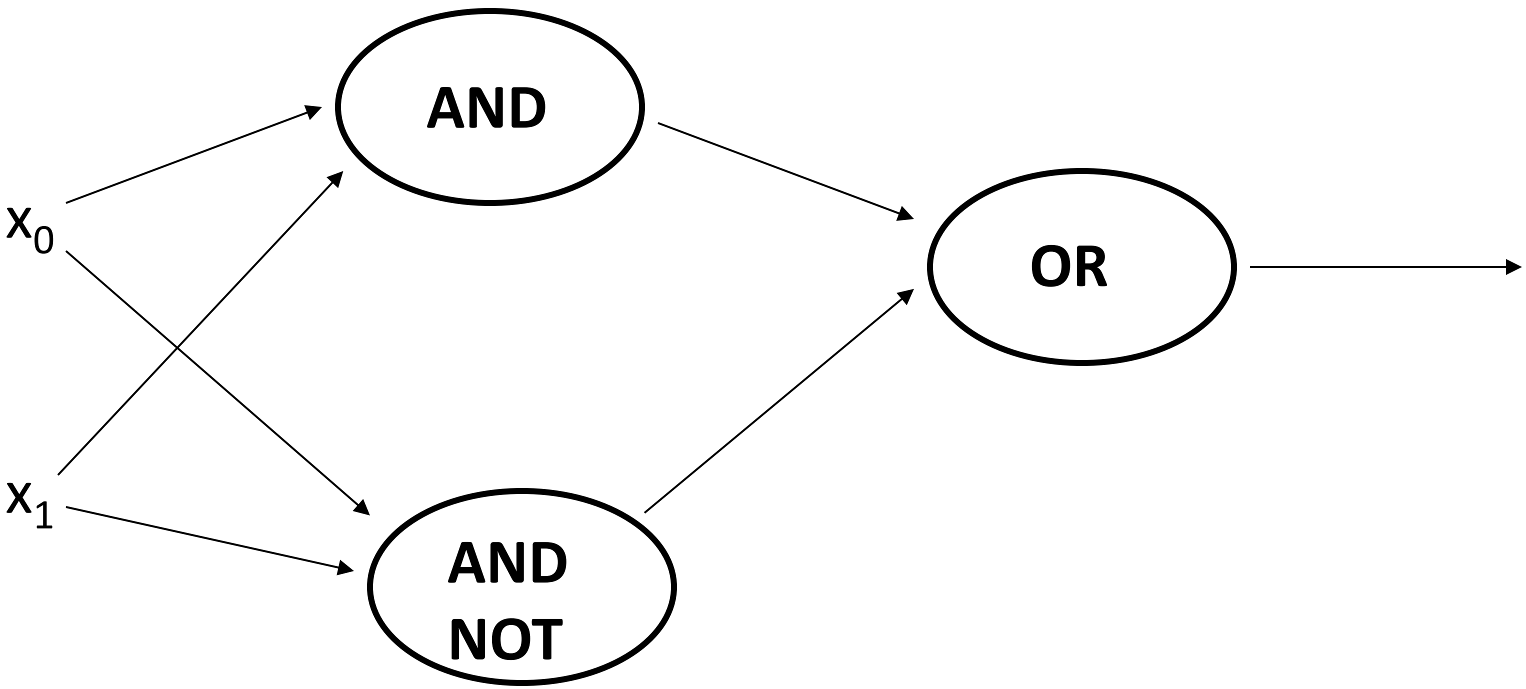
multi-layer perceptrons (MLP)

(can we represent XOR?)



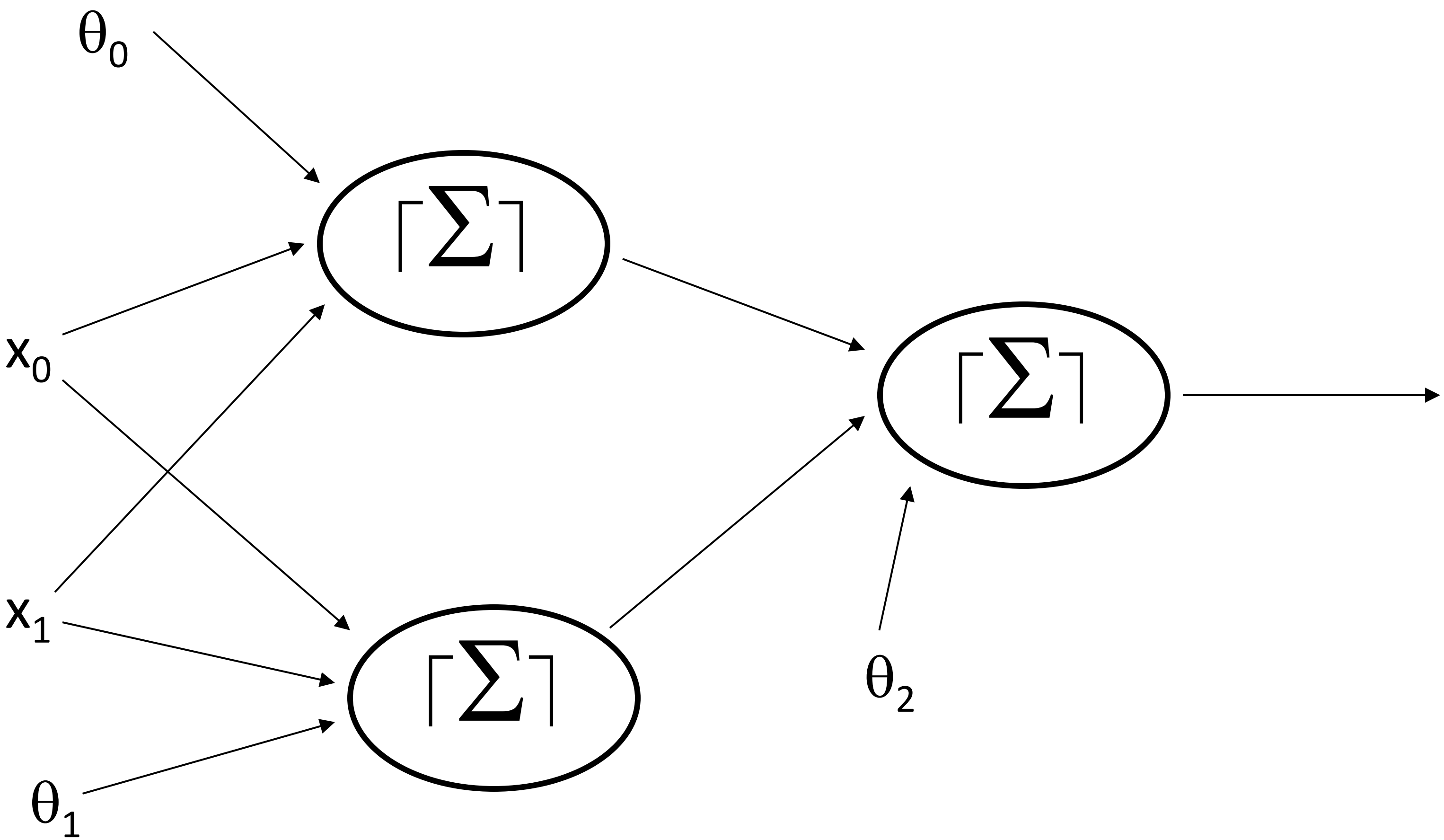
# ANN XOR

$$x_0 \oplus x_1$$



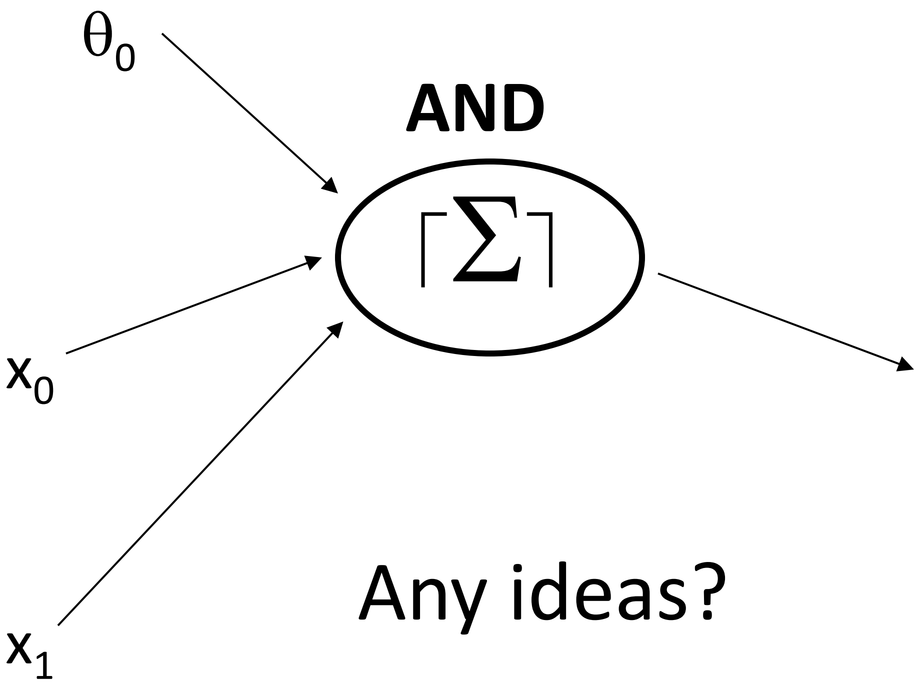
# ANN XOR

# $x_0 \oplus x_1$



# ANN XOR

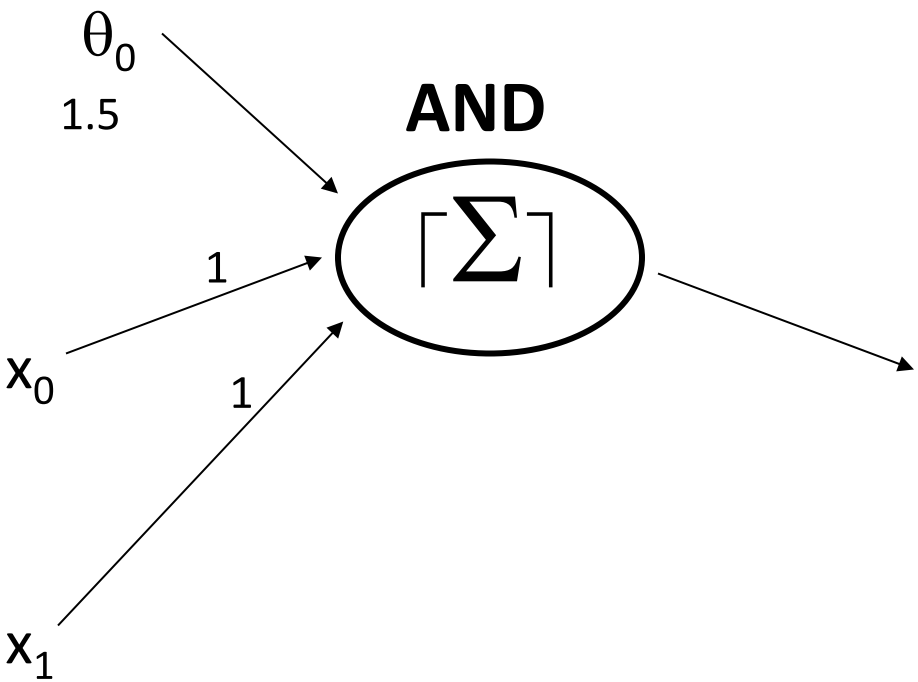
$$x_0 \oplus x_1$$




Any ideas?

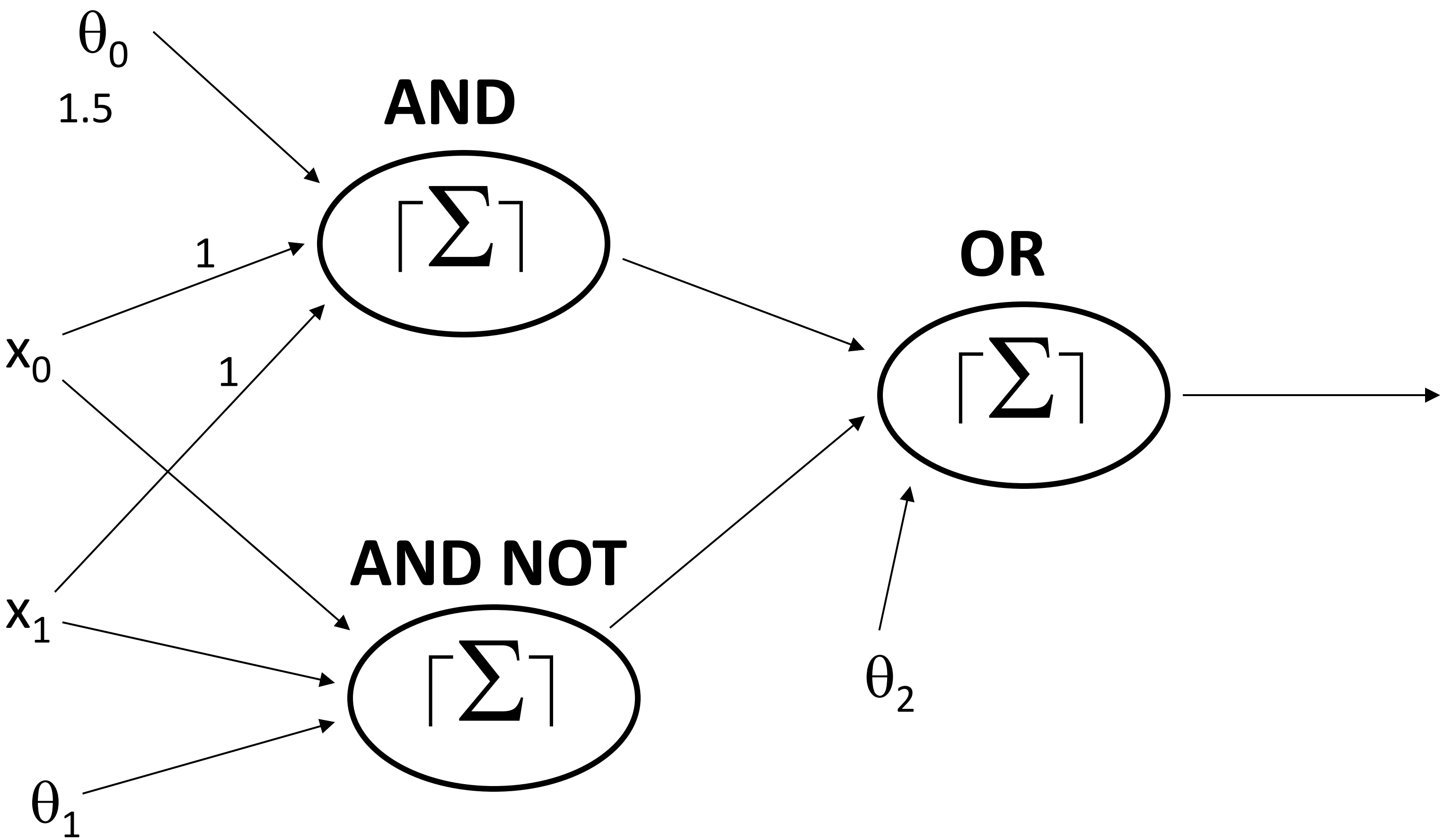
# ANN XOR

$$x_0 \oplus x_1$$



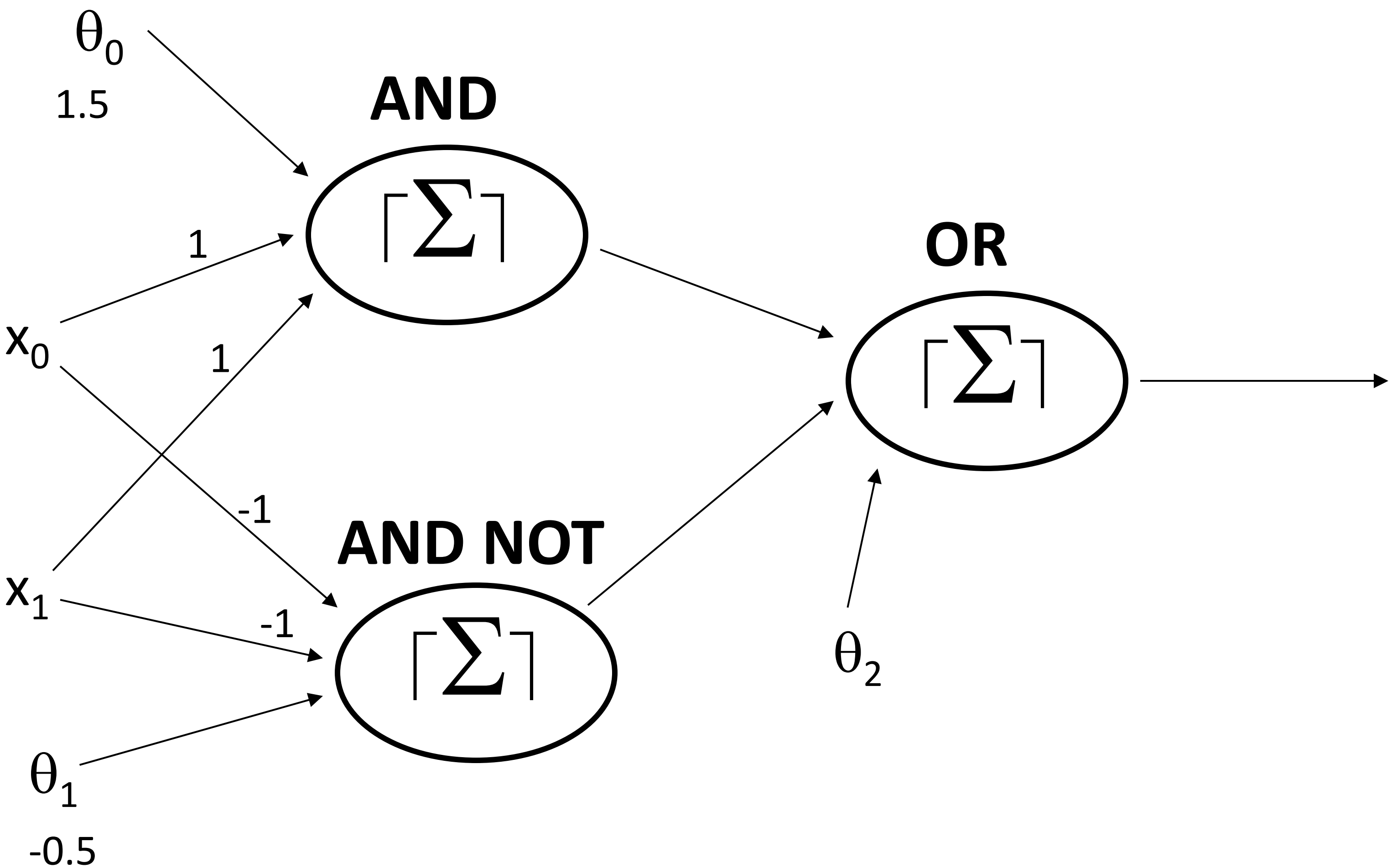
# ANN XOR

$x_0 \oplus x_1$



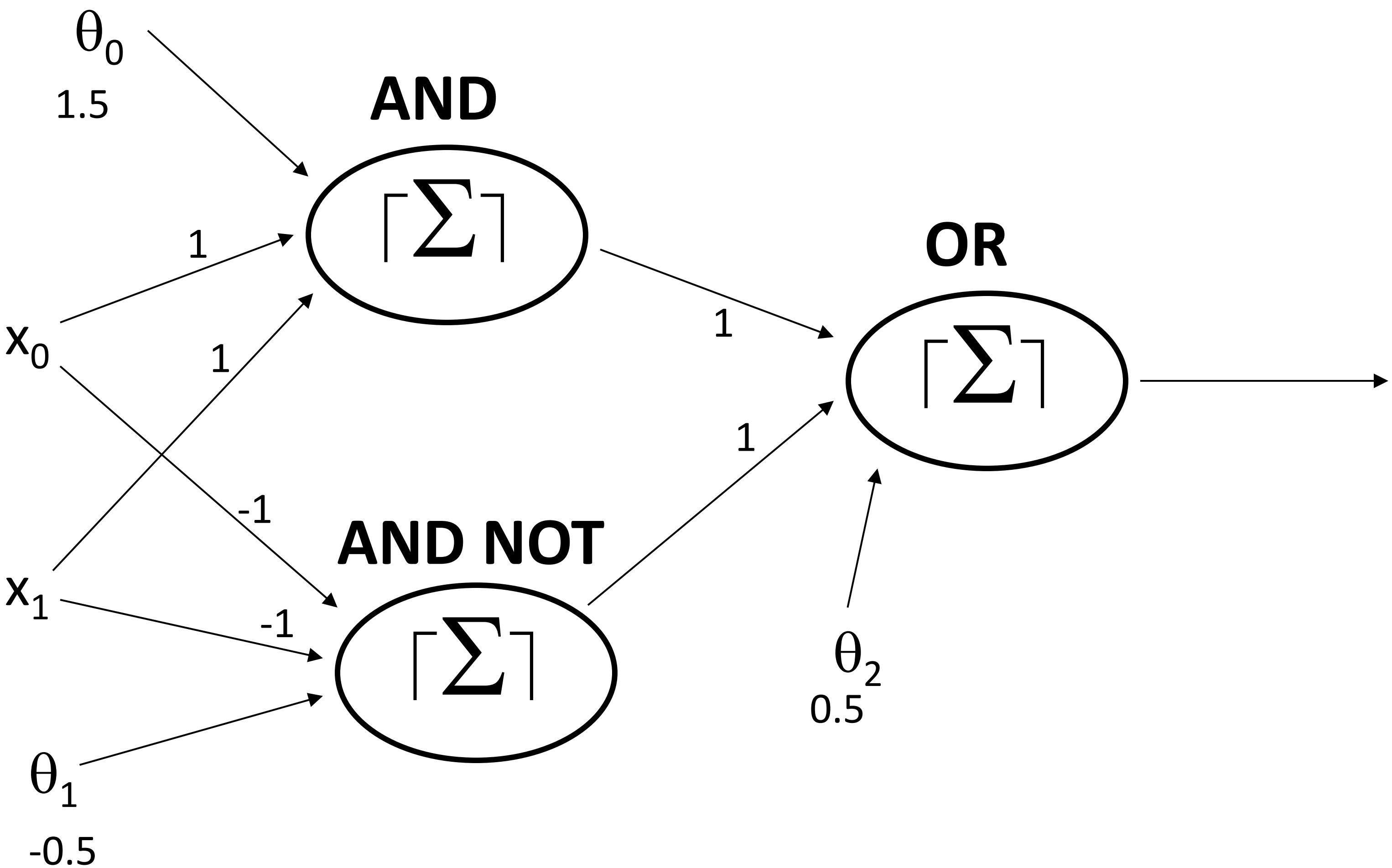
# ANN XOR

## $x_0 \oplus x_1$



# ANN XOR

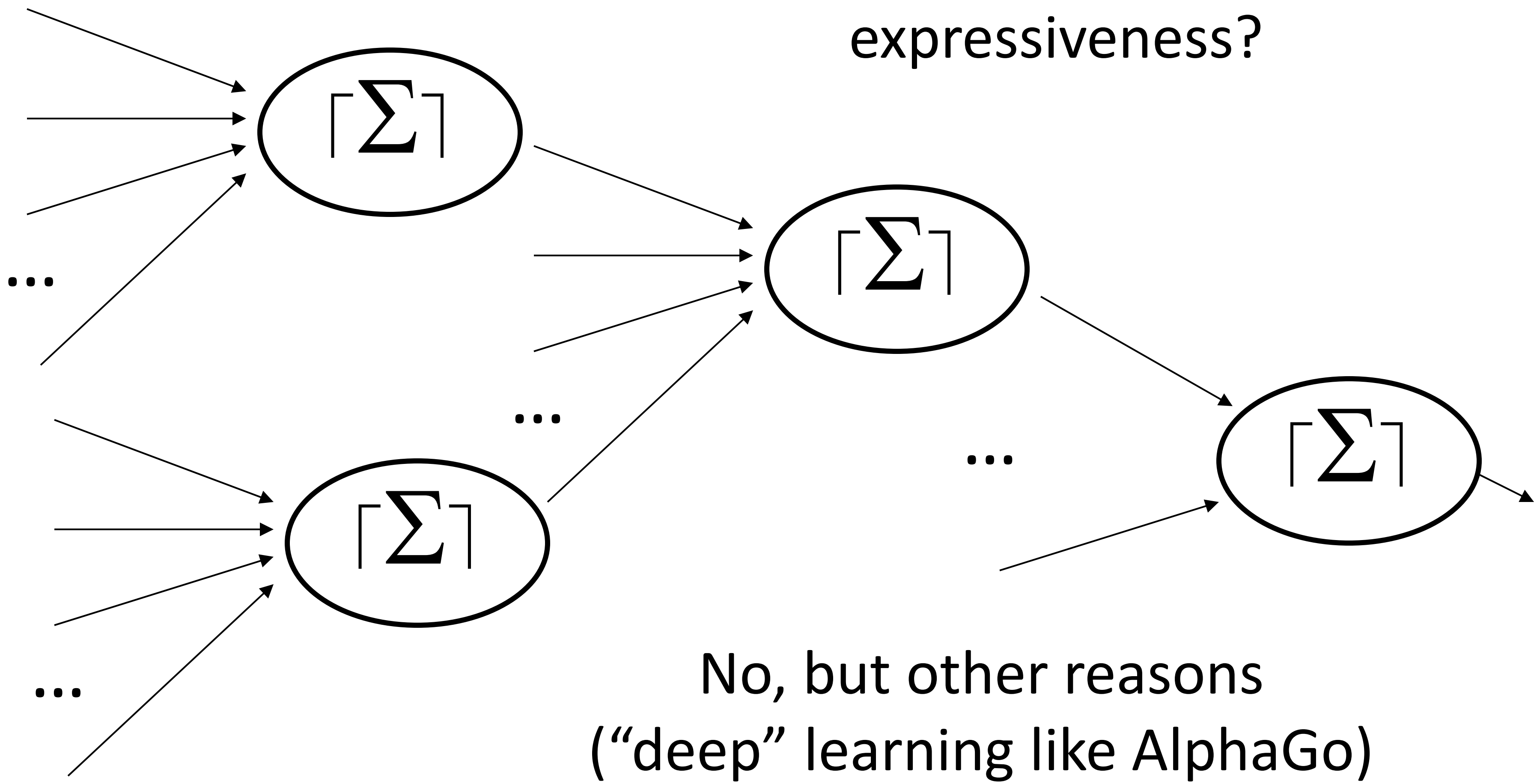
## $x_0 \oplus x_1$



# Artificial Neural Networks:

## multi-layer perceptrons

Add more levels for more expressiveness?



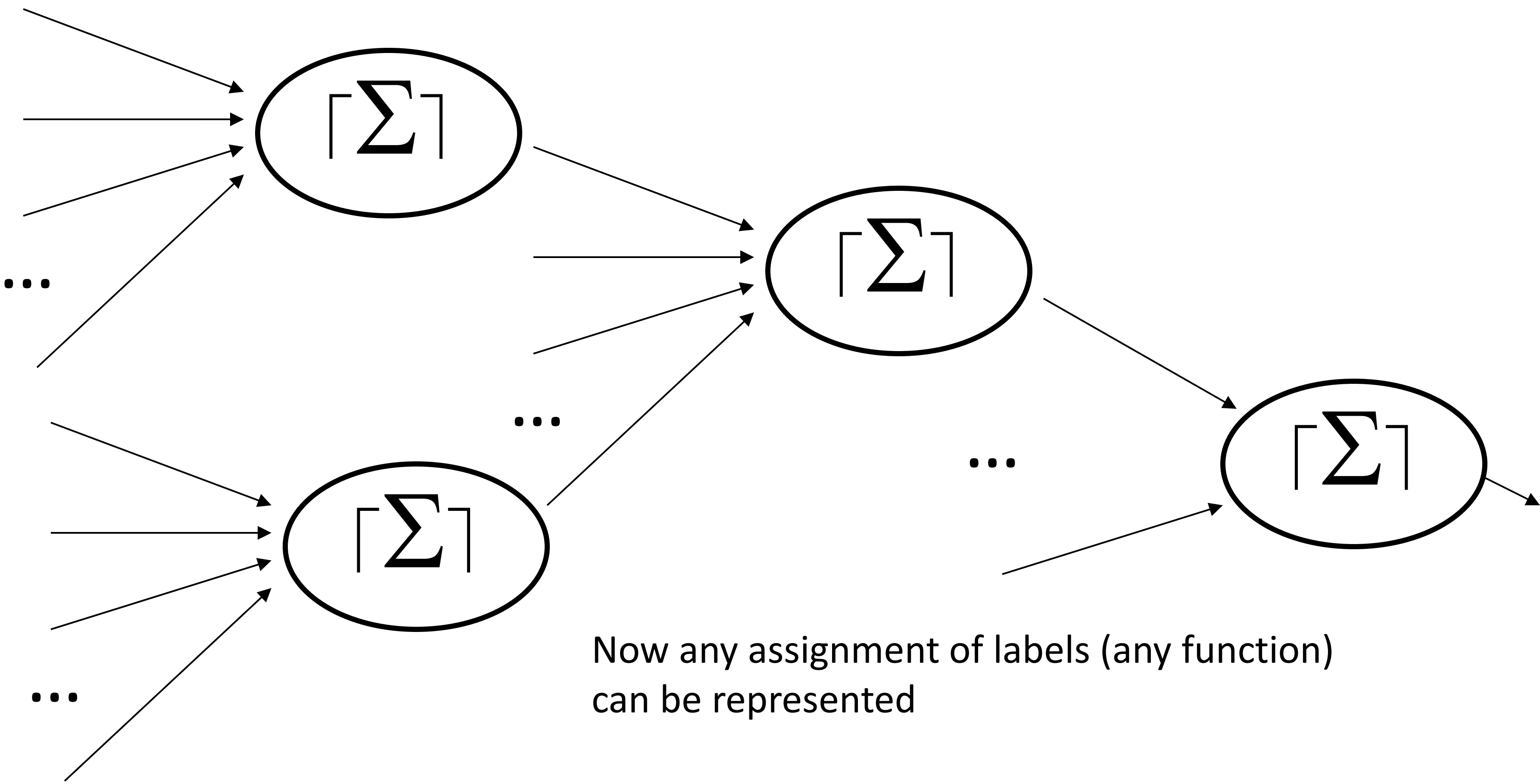

No, but other reasons ("deep" learning like AlphaGo)

# Remember our Two Important Questions

- Representation
  - Characterize the expressiveness
  - Which functions of the inputs are in the space?
  - ANN’s: any Boolean function if $\geq$ 2 layers
  - At least 1 *hidden layer*
    - any but the output perceptron(s)
    - perceptrons whose outputs are hidden (no easy error assessment)
    - (for some, also not the input layer)
    - perceptrons in a hidden layer are called *hidden units*
- Learnability
  - Given an acceptable set of training examples
  - Can we efficiently find the function we want?

# Can We Still Learn Efficiently?

## (is there a generalized perceptron convergence theorem)




Now any assignment of labels (any function) can be represented

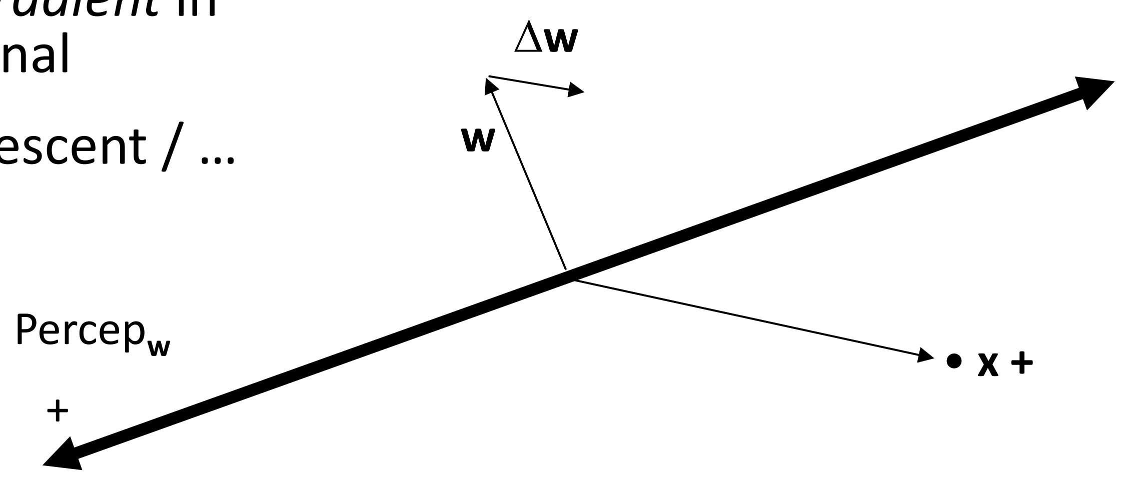
# No*

- Minsky and Papert suspected not
  in the influential book *Perceptrons* (1969)
- There could not be a generalized perceptron convergence theorem
- This largely killed off research interest
  (for nearly 20 years)
- Minsky and Papert were right

* but for a slightly modified linear device the answer becomes Yes, in fact quite easily

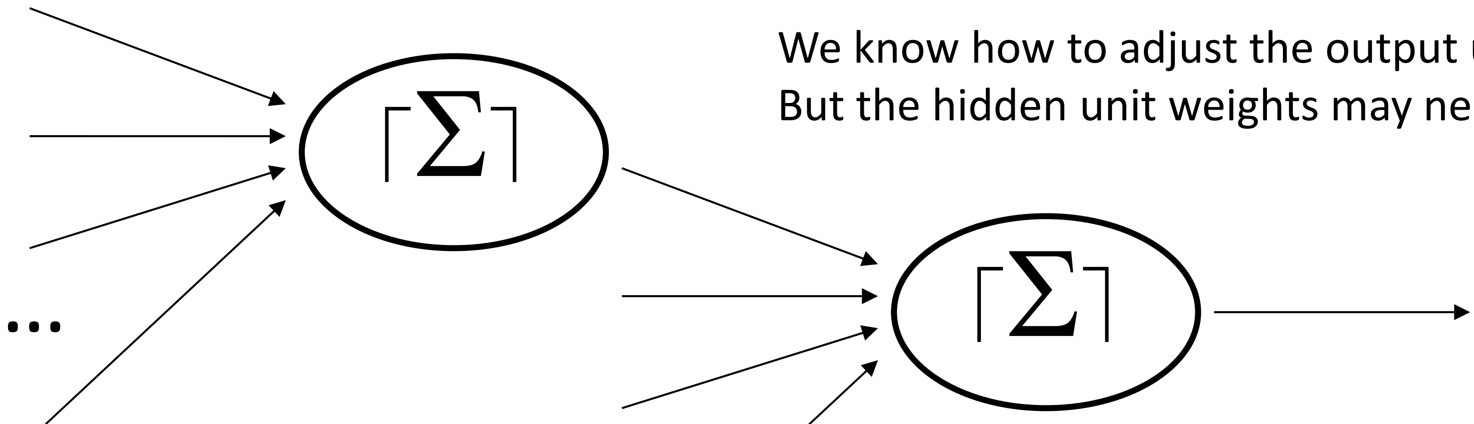
# Recall Perceptron Learning

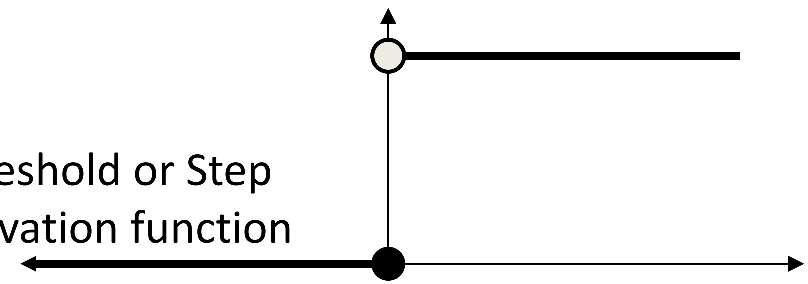
- Loss function = - err $\mathbf{w} \cdot \mathbf{x}$
- Want the opposite: step **w** in direction $-\nabla_w$ loss
- Remember Why:
- Greedy: the right direction makes smallest change in **w** for the largest reduction in loss
- Importance role of the *gradient* in propagating the error signal
- Hill climbing / gradient descent / …



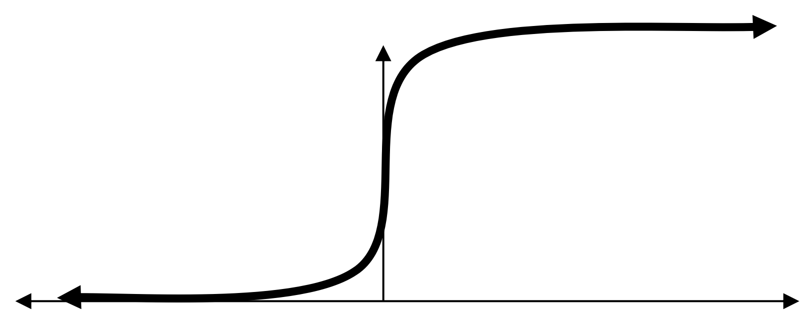
# Why "No"?

We know how to adjust the output unit's weights
But the hidden unit weights may need adjustment





Threshold or Step activation function



Sigmoid activation function (differentiable)

# Back-Propogation

- Hinton, Rumlehart,…
- Common sigmoid activation function: $g(x) = (1+e^{-x})^{-1}$
- Then g’ = g(1-g)
- Compute $\Delta \mathbf{w} = \alpha$ err g’ $\mathbf{x}$
- g’ apportions the error according to each unit’s ability to influence the error
- Learning weights for ANN’s is not only possible, it is quite easy (HW7)
- Section 18.7.4 explains it well

# Alternative Activation Functions

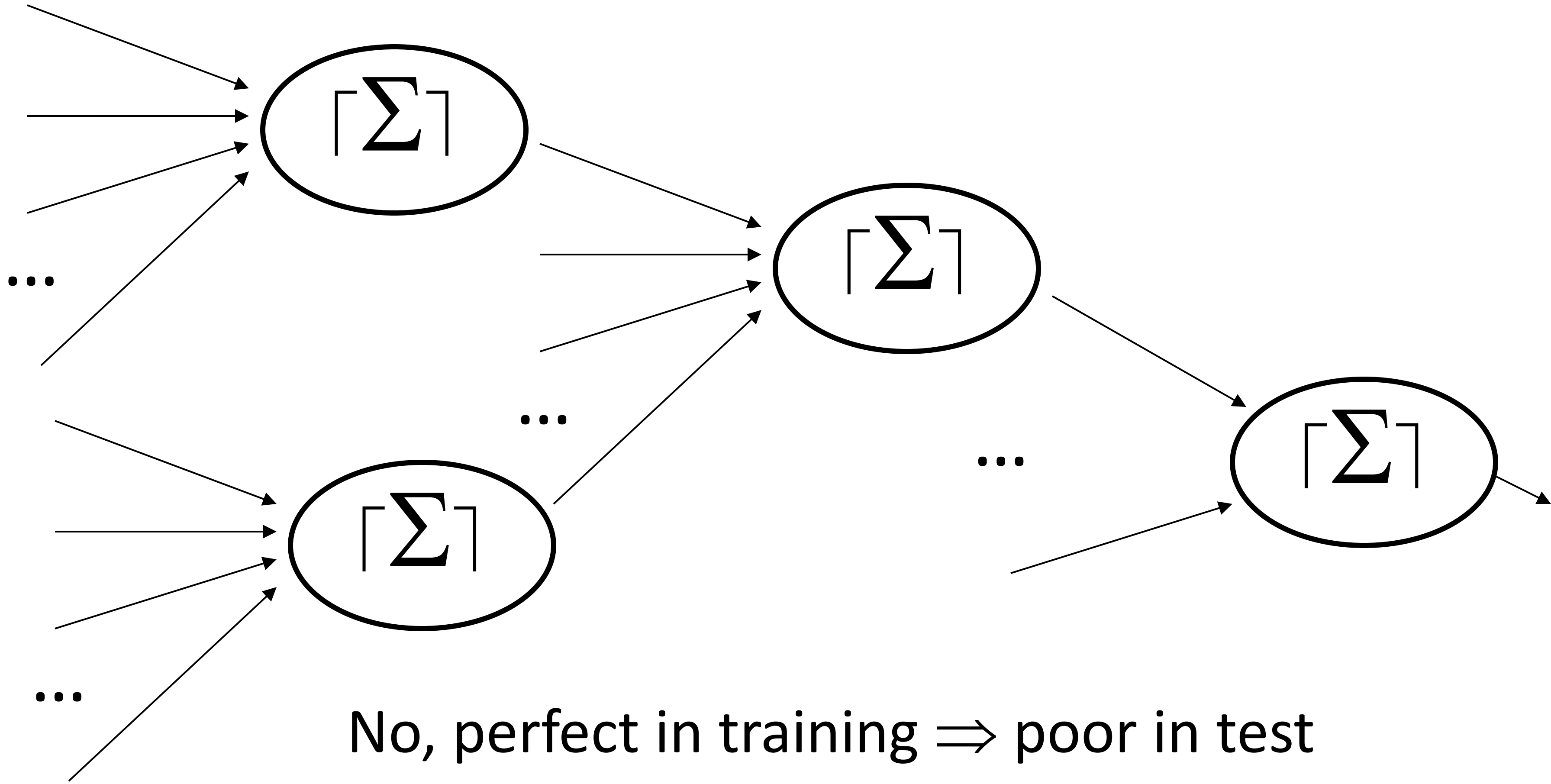
- Other forms of the logistic function
- ArcTangent
- Hyperbolic tangent
- Rectified linear unit (ReLU)
- Leaky ReLU
- many others

# Two Wrinkles

- This is a greedy / hill-climbing / gradient descent algorithm
  - So…
  - Is Expected Utility convex in **w**?
    For: $\widehat{U}_{\boldsymbol{Z}} = F(\boldsymbol{Z}, \boldsymbol{w})$
  - No, far from it…
- ANN = structure + weights
  - We can adjust the weights
  - How do we choose a structure?
  - Structure is combinatorial…
- Some heuristic approaches
  - Based on non-systematic search (remember simulated annealing)
  - Random restarts
  - Weight resets
  - Incremental structure changes
  - Ablations / Additions

# With enough units, an ANN (MLP) can learn any assignment of training labels

Is this a good thing?




No, perfect in training $\Rightarrow$ poor in test