More Neural Networks

From Single Perceptron to Multilayered Networks

CIS 32

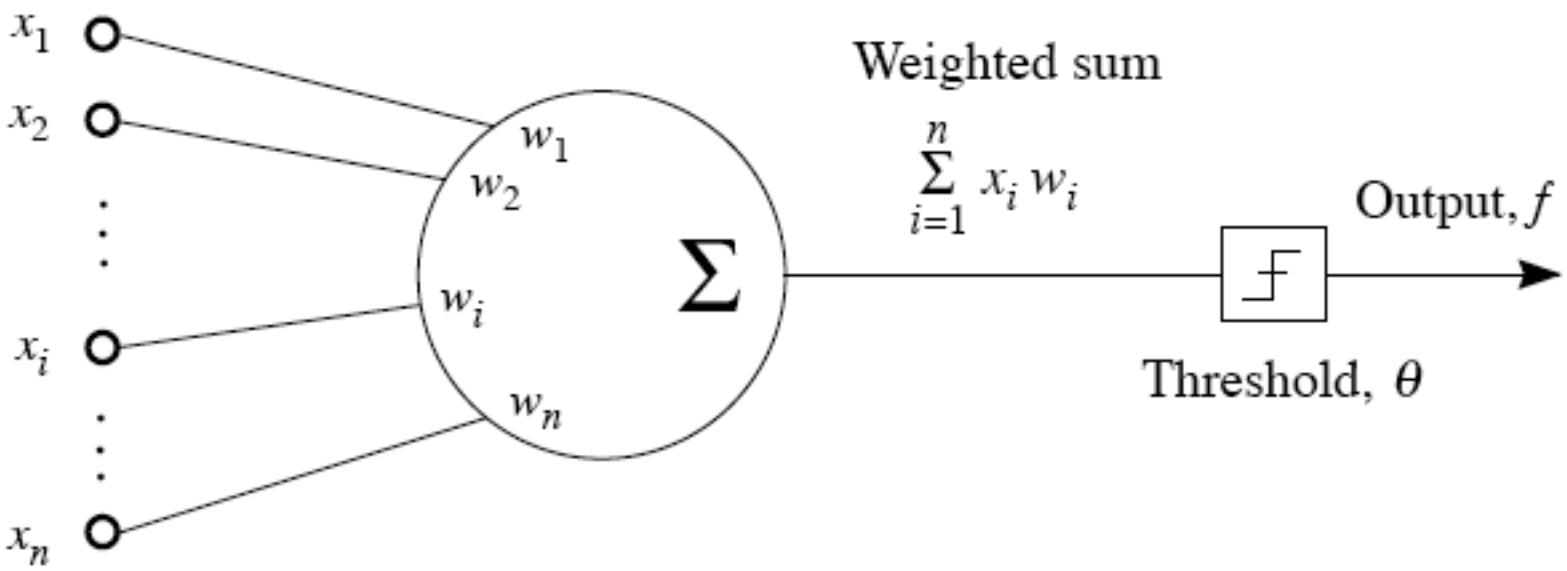
Functionalia

Office Hours Today 2 to 3pm - 0317 N (Bridges Room)

Today:

- Neural Network Example - Recap
- Learning In Single Perceptron
- Multi-Layered Networks

Recap



$$f = 1 \text{ if } \sum_{i=1}^{n} x_i w_i \geq \theta$$

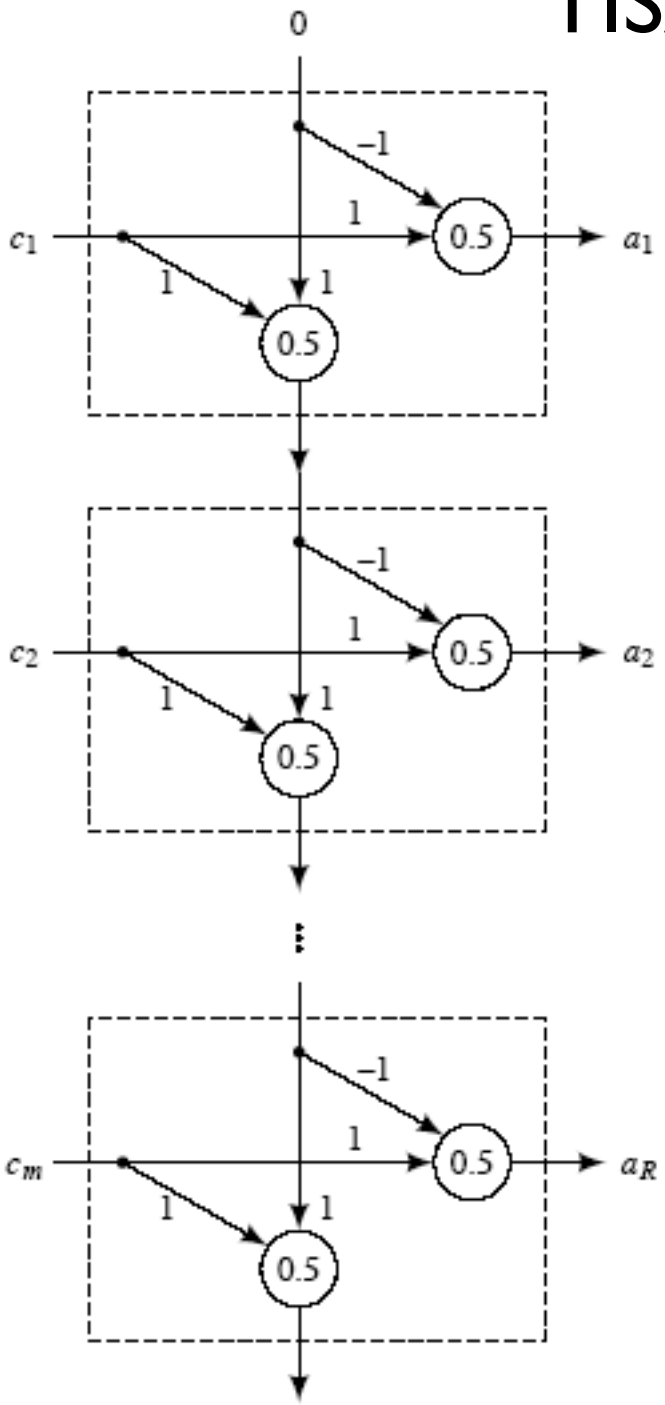
$$= 0 \text{ otherwise}$$

Output is 1 if

$$s = X \cdot W > \theta$$

Output is 0 otherwise.

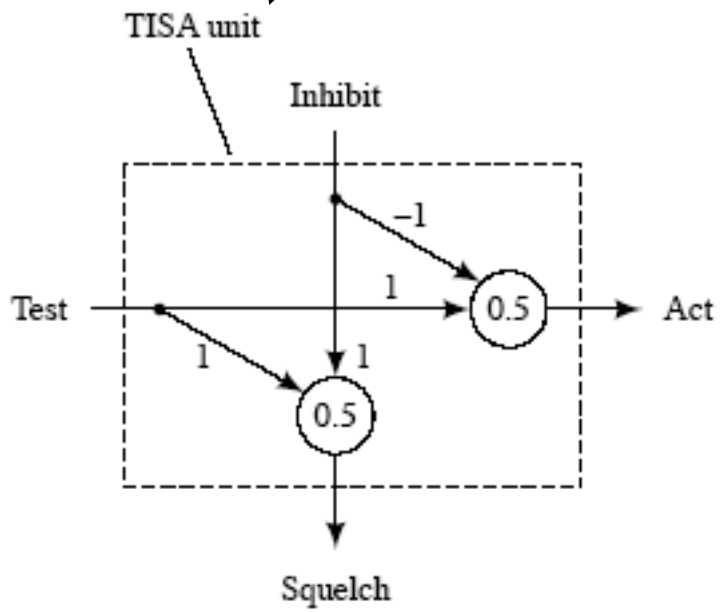
TISA Network



Each rule is a:

Test,
Inhibit,
Squelch,
Act

(TISA) circuit



Neural Network Example

Consider our Boundary Following Agent with the following Production System.

Draw out a TLU Network that encodes this rules set.

- If $x_1 = 1$ and $x_2 = 0$ then east
- If $x_2 = 1$ and $x_3 = 0$ then south
- If $x_3 = 1$ and $x_4 = 0$ then west
- If $x_4 = 1$ and $x_1 = 0$ then north

Least Squared Error

A common way to train a TLU is through an error function.

We define:

$$\epsilon = \sum_{X \in \Theta} (d_i(X_i) - f_i(X_i))^2$$

where:

- $d_i(X_i)$ is the outcome we want for X_i
- $f_i(X_i)$ is the outcome we get.

Often we write these functions as d_i and f_i

We then minimize ϵ

Gradient Descent

- If θ (the threshold) is rolled into the weights, then the value of ϵ depends on the weights.

(Since these determine the value of f_i)

- We minimize by looking at the gradient of ϵ with respect to the weights...

- ...and then altering the weights to move ϵ down the gradient.

- Eventually this *gradient descent* will take us down to the minimum value of ϵ.

Batch vs. Incremental Training

- The computation of ϵ is complicated by the fact that its value depends on ***all*** the X_i in Θ.

- Often it is easier to do the calculation for one X_i, adjust the weights to move down the gradient, and then start over with another X_j .

- Thus we do the learning incrementally, taking each member of Θ in an order (called $\sum$).

- The incremental version only ever approximates the result of doing it "properly" (the batch way), but often it is a good approximation.

- Here we will just look at the incremental version.

Gradient of ϵ

- When we have a single input vector *X*, with output *f* and desired output *d*, the error is:

$$\epsilon = (d - f)^2$$

- The gradient of ϵ with respect to the weights is

$$\frac{\partial \epsilon}{\partial W}$$

and

$$\frac{\partial \epsilon}{\partial W} = \left[\frac{\partial \epsilon}{\partial w_1}, \frac{\partial \epsilon}{\partial w_2}, \ldots, \frac{\partial \epsilon}{\partial w_{n+1}}\right]$$

- Since ϵ depends on W through

$$s = X \cdot W$$

it follows that:

$$\frac{\partial \epsilon}{\partial W} = \frac{\partial \epsilon}{\partial s}\frac{\partial s}{\partial W}$$

Chain Rule

- Since:

$$\frac{\partial s}{\partial W} = X$$

Derivative of s

it follows that:

$$\frac{\partial \epsilon}{\partial W} = \frac{\partial \epsilon}{\partial s}X$$

- From our definition of error:

$$\epsilon = (d - f)^2$$

- Furthermore we can write:

$$\frac{\partial \epsilon}{\partial s} = -2(d - f)\frac{\partial f}{\partial s}$$

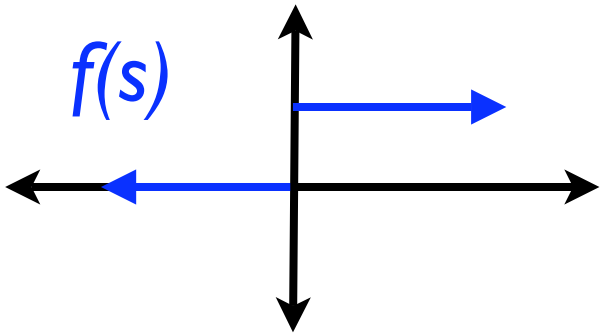
and so:

$$\frac{\partial \epsilon}{\partial W} = -2(d - f)\frac{\partial f}{\partial s}X$$

- This seems to give us a way of working out what the gradient is.

- However, we have a problem.

$$\frac{\partial f}{\partial s}$$



- The problem is that the TLU output *f* , cannot be differentiated.

- Most times we vary s a little we get no change in *f*.

- Sometimes, though, we get a big change (from 0 to 1 or vice-versa).

- There are several ways around this.

  - Ignore the threshold and set *f* = s.
  - Replace the threshold function with something we can differentiate or otherwise find the gradient of.

The Widrow-Hoff procedure

Let's try and adjust the weights so that:

- Every training vector labeled with a 1 produces a dot product of 1
- Every training vector labeled with a 0 produces a dot product of -1.

Then, with

$$f = s$$

the incremental squared error is:

$$\epsilon = (d - f)^2 = (d - s)^2$$

and

$$\frac{\partial f}{\partial s} = 1$$

- This makes the gradient:

$$\frac{\partial \epsilon}{\partial W} = -2(d - f)X$$

- If we want to then move the weight vector down the gradient, we can set the new value of the weight vector as:

$$W := W + c(d - f)X$$

- The factor of 2 is combined into the *learning rate parameter* c.
- As always this controls the speed of the adjustment by determining the fraction of *X* added to *W*.

- Whenever the error

$$(d - f)$$

is positive, then we add a fraction of the input into the weight.

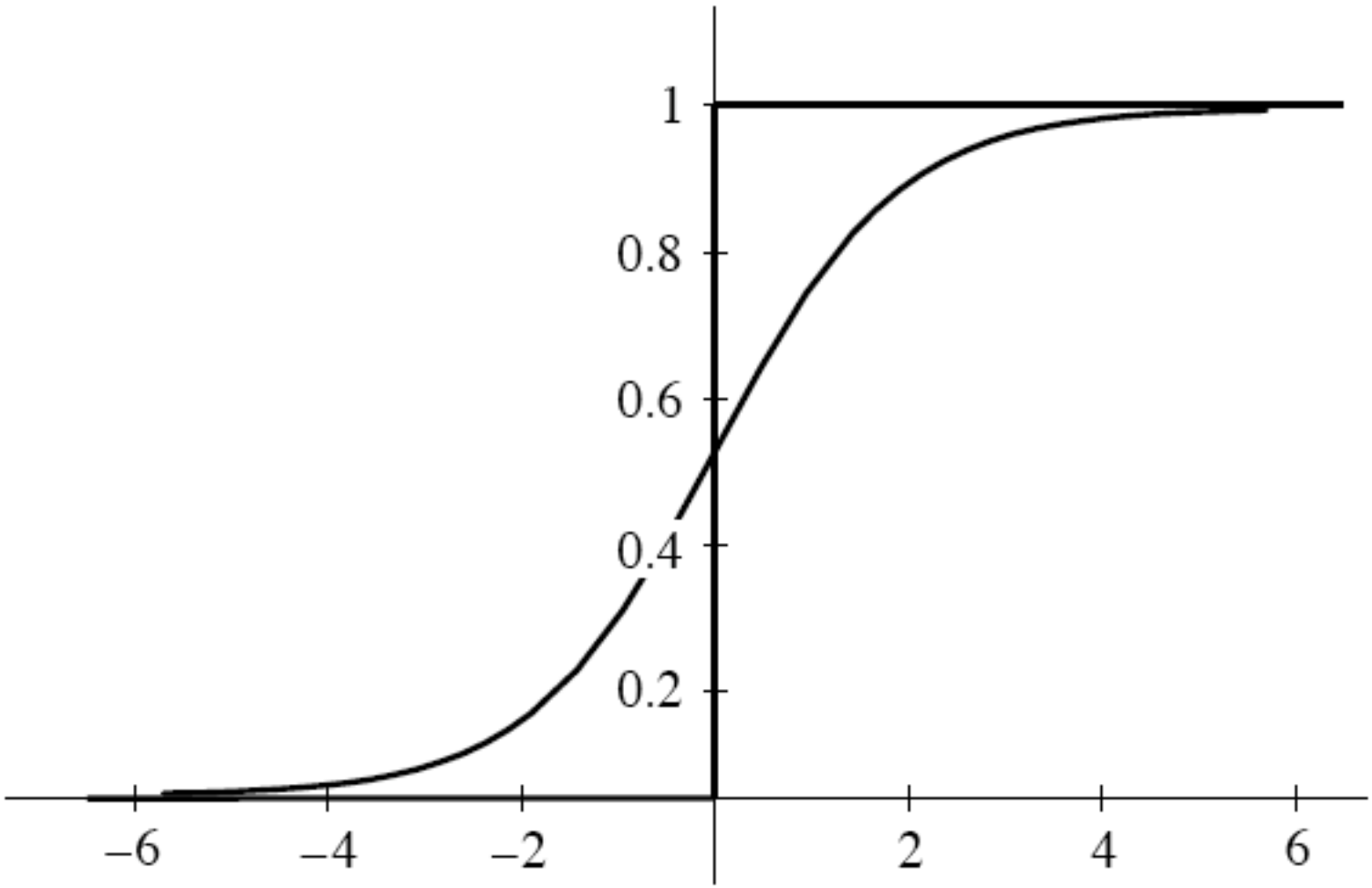
- This increases $X \cdot W$, and so decreases

$$(d - f)$$

- If the error is negative we subtract a fraction of the input and reverse the effect.
- Once we have found the best set of weights, we can go back to using the threshold function.

The generalized Delta procedure

- Another way to handle the threshold function is to replace it with something we can differentiate.



This function is known as a ***sigmoid***:

$$f(s) = \frac{1}{1 + e^{-s}}$$

With this function, we have the partial derivative:

$$\frac{\partial f}{\partial s} = f(1 - f)$$

Since

$$\frac{\partial \epsilon}{\partial W} = -2(d - f)\frac{\partial f}{\partial s}X$$

we have:

$$\frac{\partial \epsilon}{\partial W} = -2(d - f)f(1 - f)X$$

- This gives us another rule for changing weights:

$$W := W + c(d - f)f(1 - f)X$$

- This compares to the Widrow-Hoff procedure as follows:
  - In W-H, *d* is 1 or -1. In generalized Delta it is 1 or 0.
  - In W-H, *f* is equal to *s*. In generalized Delta, *f* is the output of the sigmoid function.
  - Generalized Delta has the extra term $f(1 - f)$
  - With the sigmoid, $f(1 - f)$ varies in value from 0 to 1.
- It has value 0 when *f* is 0 or 1.
- It has maximum value of 0.25 when *f* has value 0.5 (and the input to the sigmoid is 0).

- One can think of the sigmoid as a "fuzzy boundary".
- When the input is a long way from the boundary, $f(1-f)$ has a value close to 0.
- Thus hardly any adjustment is made to the weights.
- When the input is closer to the boundary, then weight changes are more significant.
- These changes are always to reduce the error.
- Once the weights are established, we can go back to using the step function.

A general approach

- Both these techniques have done the same thing.
- They have replaced something we couldn't find the slope of with something we could.
- This obviously trains the weights approximately.
- However, it seems that the approximation is often good enough.
- In any case, we are interested in performance on non-training examples.

The error-correction procedure

- Another approach keeps the original threshold function.
- We then forget about differentiation and just adjust the weights when the TLU gives a classification error.
- In other words we make a change when:

$$(d - f)$$

has value 1 or -1.

- This time the weight change rule is:

$$w := W + c(d - f)X$$

(remember c is the *learning rate parameter*)

- Just as before, the change tends to reduce the error.

- Comparing this with Widrow-Hoff, we note that both *d* and *f* are either 0 or 1.
- Whereas in W-H, *d* is 1 or -1 and *f* = s.
- If there is a *W* that gives a correct output for all $X \in \Theta$
  - Then after a finite number of adjustments, this error-correction procedure will find this weight vector.
- Thus the process will terminate, making no more weight adjustments.
- For nonlinearly separable sets of input vectors, the procedure will not terminate.

Network Structures

Two kinds of larger Neural Network Structures:

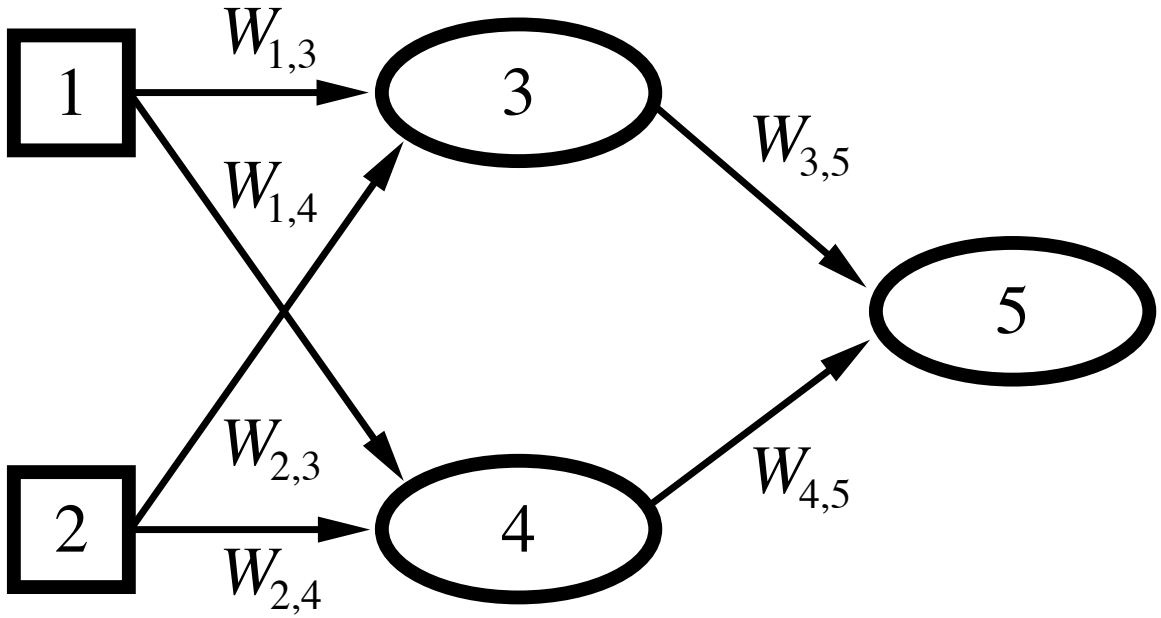
1. feed-forward networks - *acyclic*

- contains "hidden" layers and inputs.

2. recurrent networks - *cyclic*

- dynamic systems - with oscillations and chaotic behavior
- can exhibit short-term memory

Hidden Units



Activation of Unit 5 is based on the weighted outputs of Unit 3 and 4.

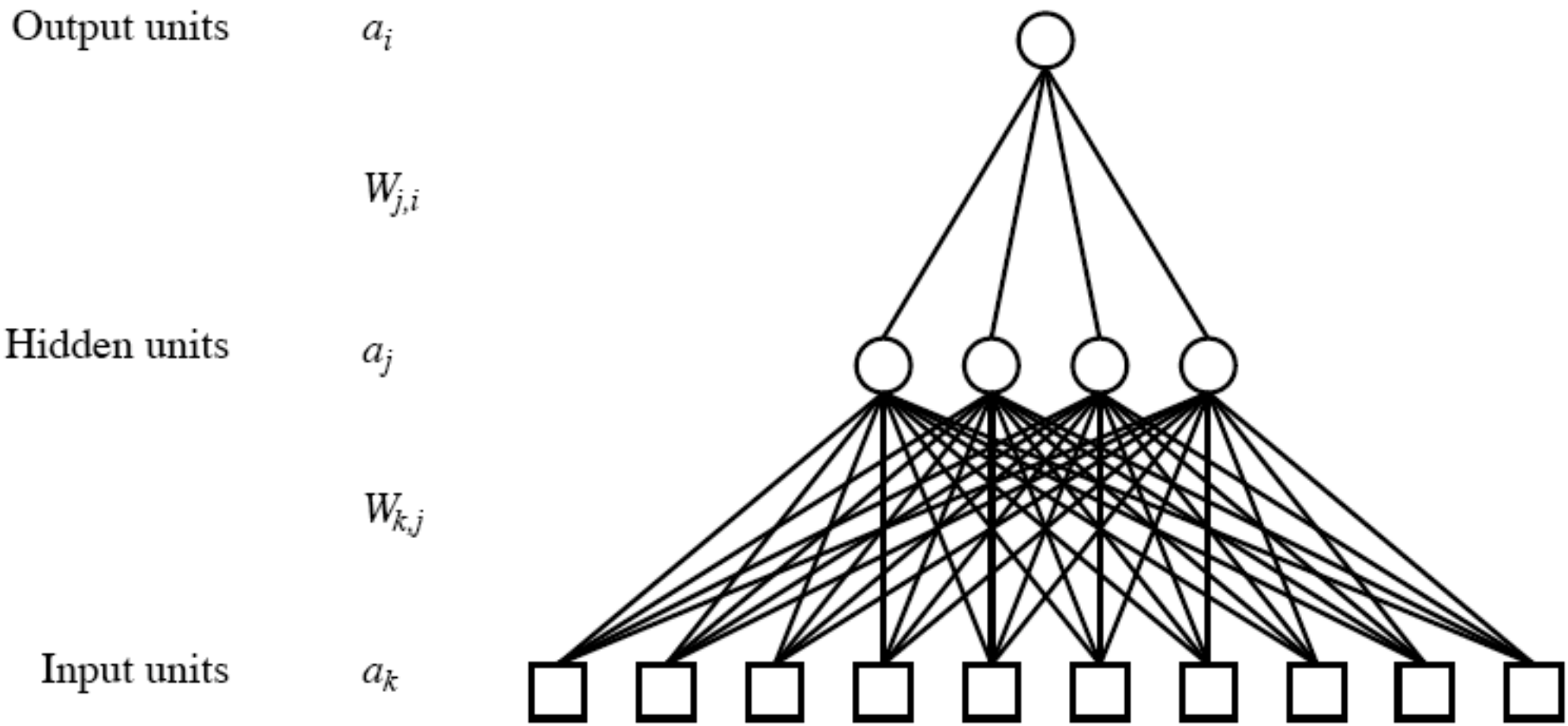
Units 3 and 4 represent the *hidden units*.

Activation depends on the Unit (can use the *sigmoid* function)

Multi-layer

Layers are usually fully connected.

Numbers of nodes typically set by hand.

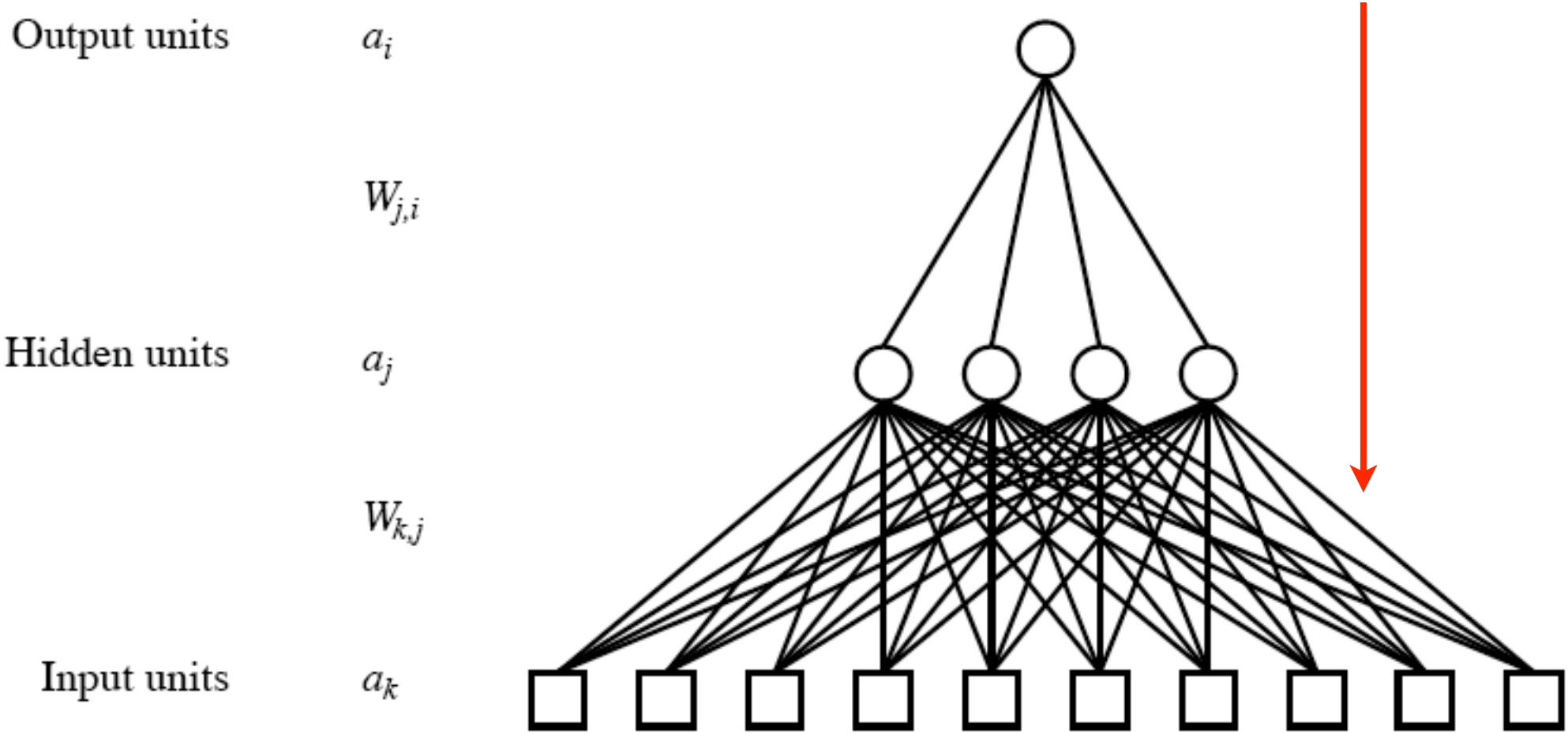


Multilayer Feed Forward

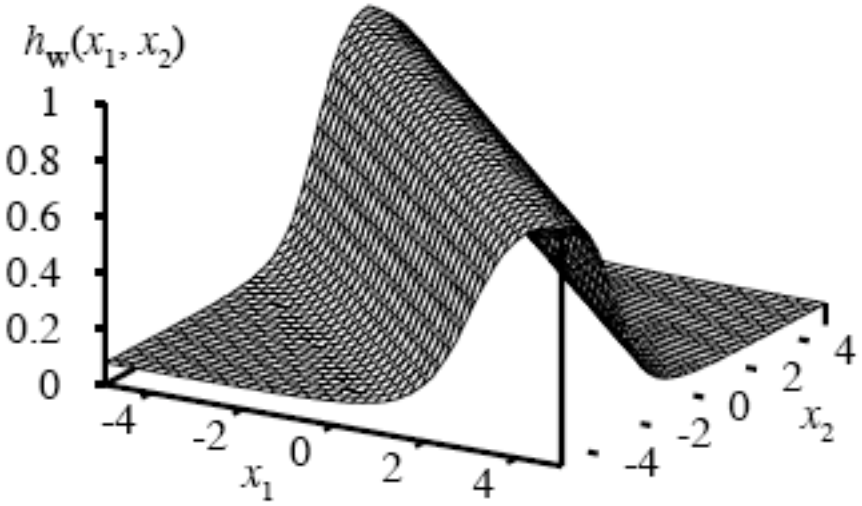
Layers are usually fully connected; numbers of nodes typically set by hand.

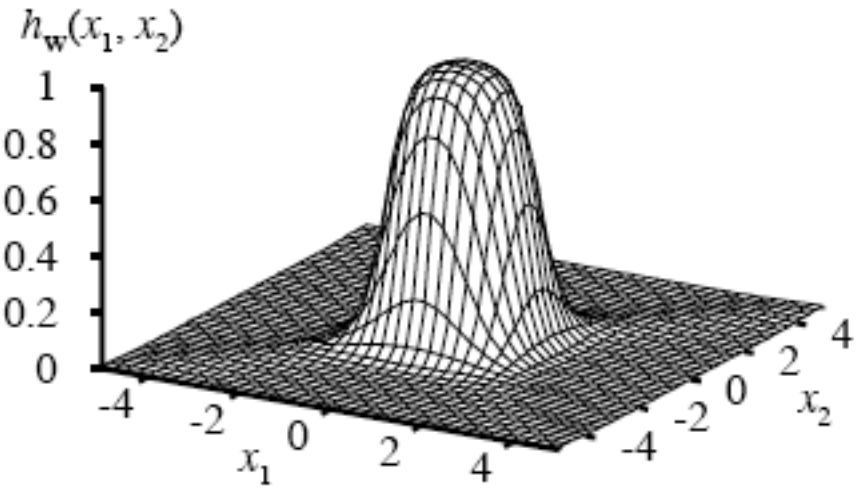
Single Hidden Layer is Most Common.

back-propagation



Larger hypothesis space





Combine two opposite-facing threshold functions to make a ridge

Combine two perpendicular ridges to make a bump

Add bumps of various sizes and locations to fit any surface

Recurrent Networks

Hopfield Networks -

- contain *bidirectional* connections (units are inputs and outputs)
- stimulus results in the networks settling into an activations pattern that most closely resembles a training example
- *N* units can store *0.138 N* training examples.

Boltzmann Machines -

- like Hopfield Networks, but contain hidden units
- activation functions are *stochastic* (functions based on a probability that a unit exhibits a 1 based on the total weighted unit)

Handout.