Chapter 5:

Some More Artificial Intelligence:

Neural Networks

Background

- Neural Networks can be :

    - **Biological** models

    - **Artificial** models

- Desire to produce artificial systems capable of sophisticated computations similar to the human brain.

Biological analogy and some main ideas

- The brain is composed of a mass of interconnected neurons
  - each neuron is connected to many other neurons
- Neurons transmit signals to each other
- Whether a signal is sent, depends on the strength of the bond (synapse) between two neurons

How Does the Brain Work ? (1)

NEURON

- The cell that performs information processing in the brain.

- Fundamental functional unit of all nervous system tissue.

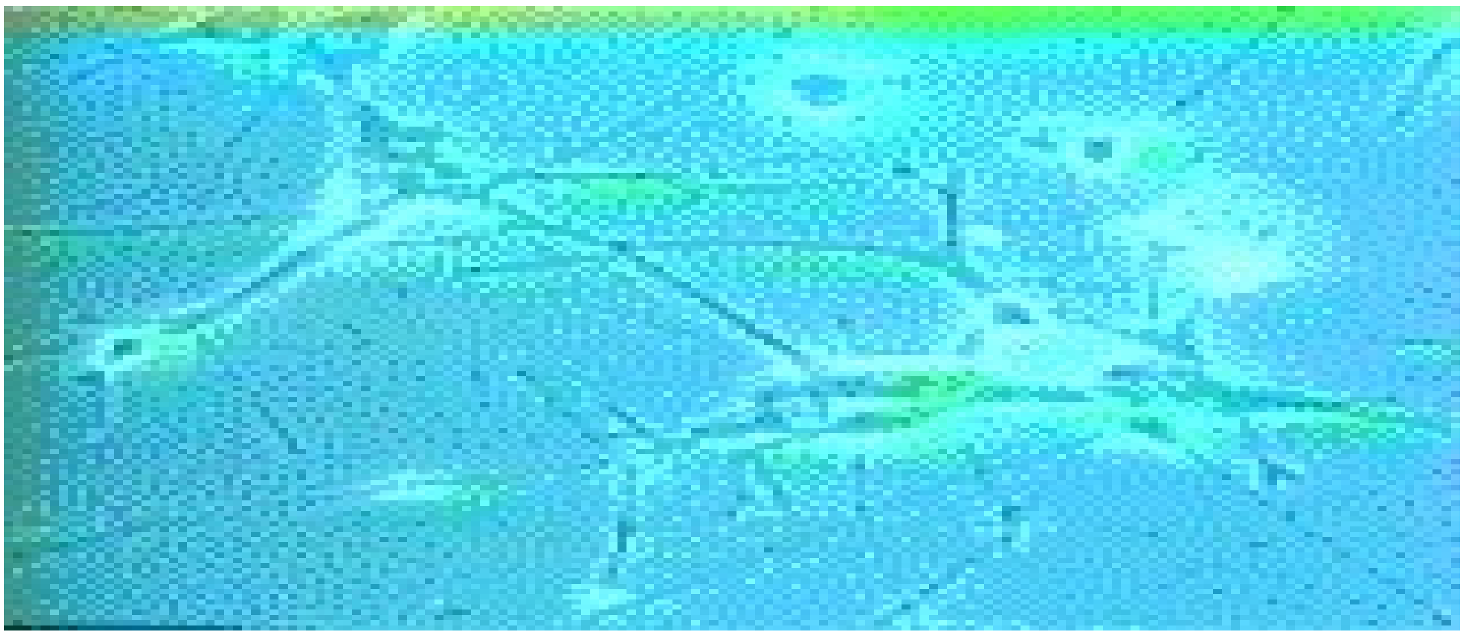


Figure 5-1: Neuron Fundamentals

Brain vs. Digital Computers (1)

- The brain can fire all the neurons in a single step.

  ⇒ Parallelism

- Serial computers require billions of cycles to perform some tasks but the brain takes **less than a second.**

  e.g. Face Recognition

Comparison of Brain & Computer

	Human	*Computer*
Processing Elements	100 Billion neurons	10 Million gates
Interconnects	1000 per neuron	A few
Cycles per sec	1000	500 Million
2X improvement	200,000 Years	2 Years

Brain vs. Digital Computers (2)

Future : combine parallelism of the brain with the switching speed of the computer.

	Computer	Human Brain
Computational units	1 CPU, 10^5 gates	10^{11} neurons
Storage units	10^9 bits RAM, 10^{10} bits disk	10^{11} neurons, 10^{14} synapses
Cycle time	10^{-8} sec	10^{-3} sec
Bandwidth	10^9 bits/sec	10^{14} bits/sec
Neuron updates/sec	10^5	10^{14}

Figure 19.2 A crude comparison of the raw computational resources available to computers (*circa* 1994) and brains.

History

- 1943: McCulloch & Pitts show that **neurons** can be combined to construct a **Turing machine** (using ANDs, Ors, & NOTs)
- 1958: Rosenblatt shows that **perceptrons** will converge if what they are trying to learn can be represented
- 1969: Minsky & Papert showed the **limitations** of perceptrons, killing research for a decade
- 1985: **backpropagation** algorithm revitalizes the field

Definition of Neural Network

A Neural Network is a system composed of many simple processing elements operating in parallel which can acquire, store, and utilize experiential knowledge.

What is Artificial Neural Network?

Neurons vs. Units (1)

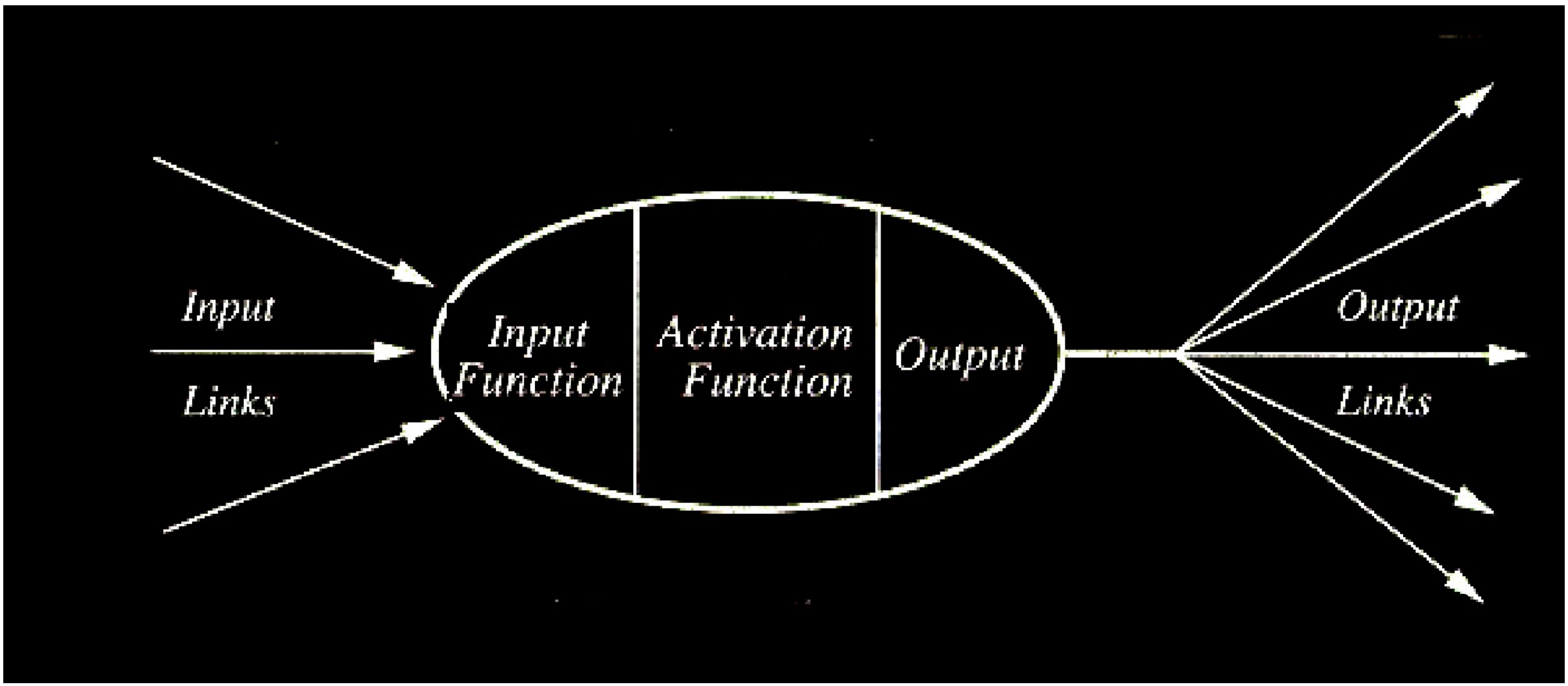
- Each element of NN is a node called unit.

- Units are connected by links.

- Each link has a numeric weight.

Computing Elements

A typical unit:



Planning in building a Neural Network

Decisions must be taken on the following:

- The number of units to use.

- Connection between the units.

How NN learns a task.

Issues to be discussed

- Initializing the weights.

- Use of a learning algorithm.

- Set of training examples.

- Encode the examples as inputs.

- Convert output into meaningful results.

Neural Network Example

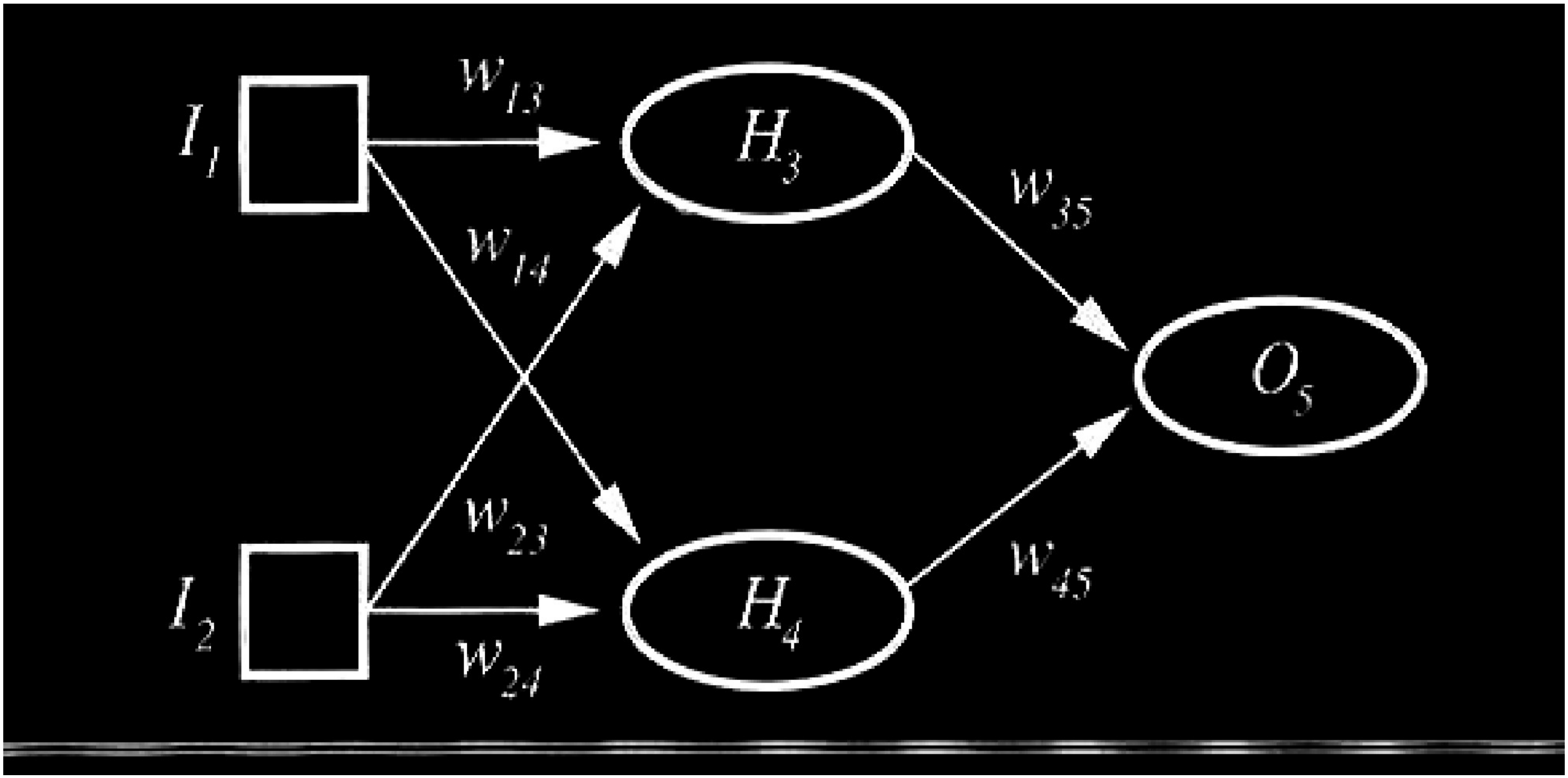


Figure 5-2: A very simple, two-layer, feed-forward network with two inputs, two hidden nodes, and one output node.

Simple Computations in this network

- There are **2 types of components:** Linear and Non-linear.

- **Linear:** Input function
  - calculate weighted sum of all inputs.

- **Non-linear:** Activation function
  - transform sum into activation level.

Calculations

Input function:

$$in_i = \sum_j W_{j,i} a_j = \mathbf{W}_i \cdot \mathbf{a}_i$$

Activation function **g**:

$$a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$

A Computing Unit.

Now in more detail but for a particular model only

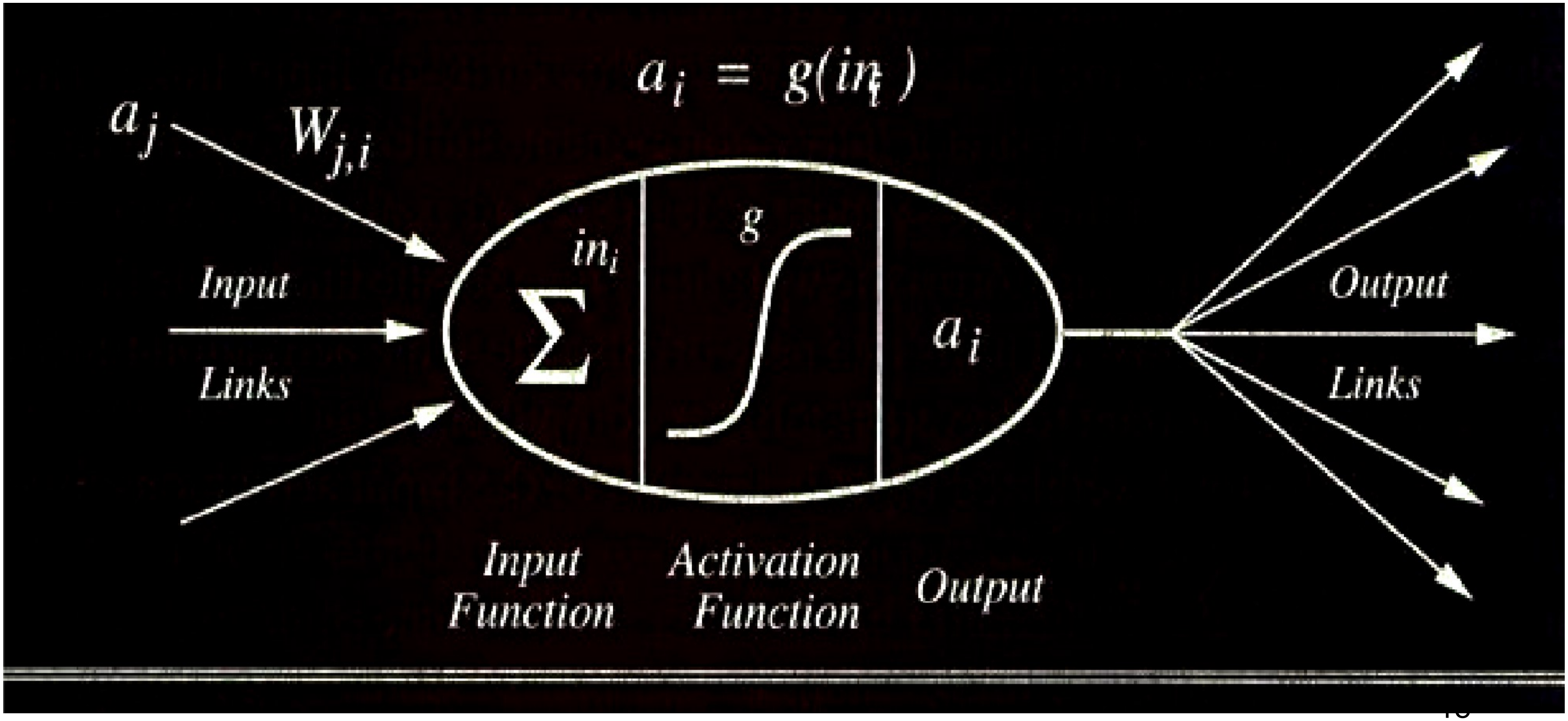


Figure 19.4. A unit

Are current computer a wrong model of thinking?

- Humans can’t be doing the sequential analysis we are studying
  - Neurons are a million times slower than gates
  - Humans don’t need to be rebooted or debugged when one bit dies.

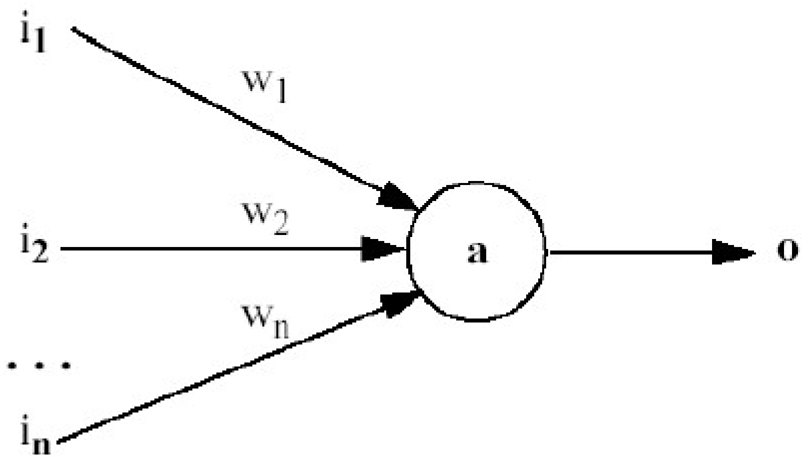
Standard Structure of an ANN

- Input units
  - represents the input as a fixed-length vector of numbers (user defined)
- Hidden units
  - calculate thresholded weighted sums of the inputs
  - represent intermediate calculations that the network learns
- Output units
  - represent the output as a fixed length vector of numbers

Operation of individual units

- $\text{Output}_i = f(W_{i,j} * \text{Input}_j + W_{i,k} * \text{Input}_k + W_{i,l} * \text{Input}_l)$
  - where f(x) is a threshold (activation) function
  - $f(x) = 1 / (1 + e^{-\text{Output}})$
    - "sigmoid"
  - f(x) = step function

Artificial Neural Networks

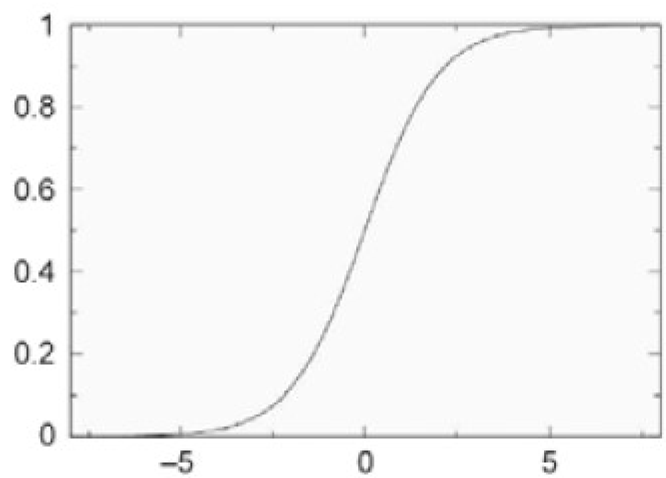


Activation

$$a(I, W) = \sum_{k=1}^{n} i_k \cdot w_k$$

Output

$$o(I, W) = \frac{1}{1 + e^{-\rho \cdot a(I, W)}}$$



Sigmoid activation function

Network Structures

Feed-forward neural nets:

Links can only go in one direction.

Recurrent neural nets:

Links can go anywhere and form arbitrary topologies.

Feed-forward Networks

- Arranged in ***layers***.

- Each unit is linked only in the unit in next layer.

- No units are linked between the same layer, back to the previous layer or skipping a layer.

- Computations can proceed uniformly from input to output units.

Feed-Forward Example

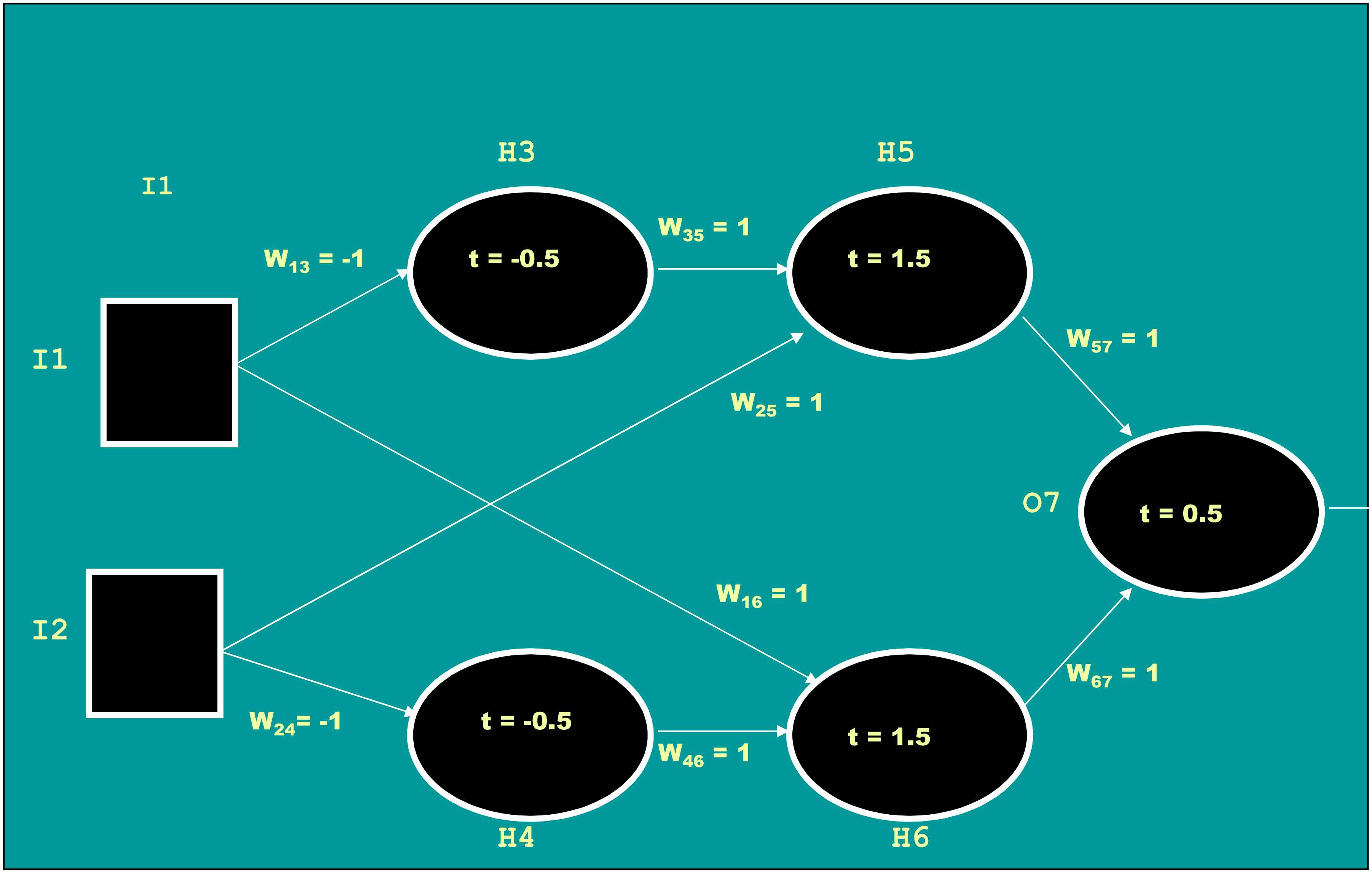


Figure 5-3: Inputs skip the layer in this case

Multi-layer Networks and **Perceptrons**

- Have one or more layers of hidden units.

- With two possibly very large hidden layers, it is possible to implement any function.

- Networks without hidden layer are called perceptrons.

- Perceptrons are very limited in what they can represent, but this makes their learning problem much simpler.

Recurrent Network (1)

- The brain is not and cannot be a feed-forward network.

- Allows activation to be fed back to the previous unit.

- Internal state is stored in its activation level.

- Can become unstable

-Can oscillate.

Recurrent Network (2)

- May take long time to compute a stable output.

- Learning process is much more difficult.

- Can implement more complex designs.

- Can model certain systems with internal states.

Perceptrons

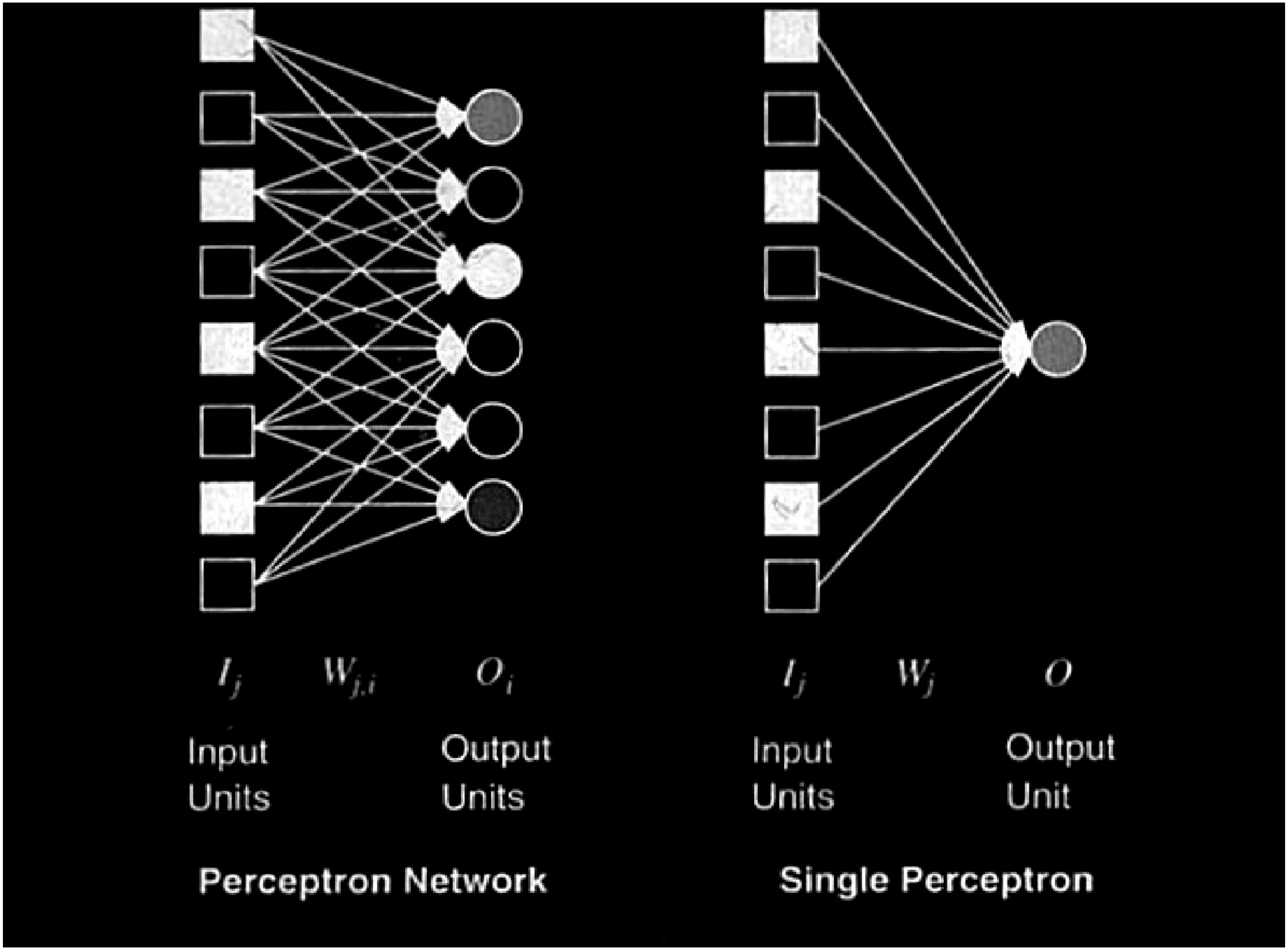
- First studied in the late 1950s.

- Also known as Layered Feed-Forward Networks.

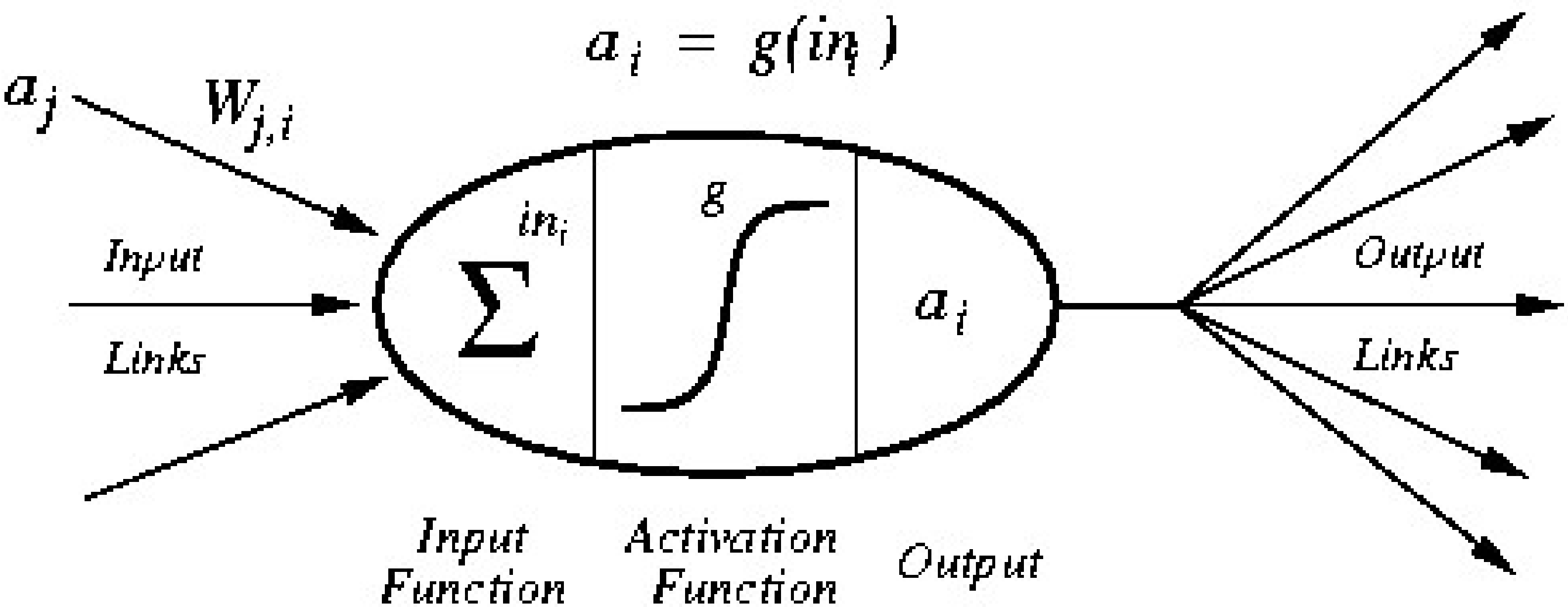
- The only efficient learning element at that time was for single-layered networks.

- Today, used as a synonym for a single-layer, feed-forward network.

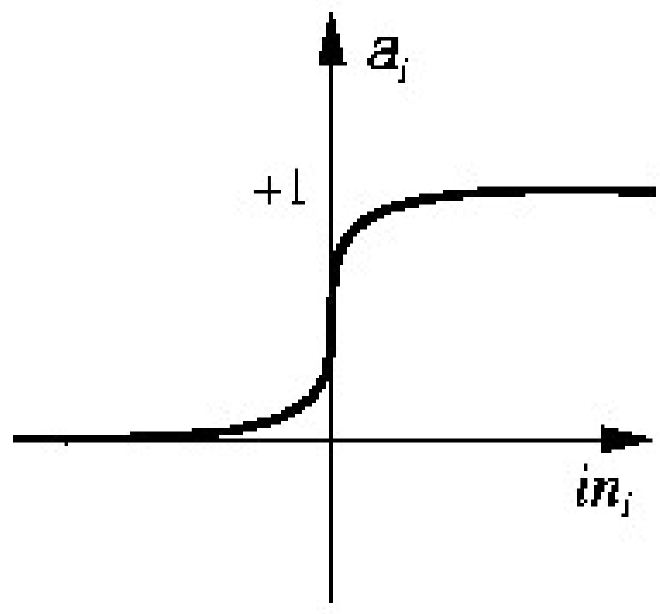
Fig. 19.8. Perceptrons



Perceptrons



Sigmoid Perceptron



(c) Sigmoid function

Perceptron learning rule

- Teacher specifies the **desired output** for a given input
- Network calculates what it thinks the output should be
- Network changes its weights in proportion to the error between the desired & calculated results
- $\Delta w_{i,j} = \alpha * [teacher_i - output_i] * input_j$
  - where:
  - α is the learning rate;
  - $teacher_i - output_i$ is the error term;
  - and $input_j$ is the input activation
- $w_{i,j} = w_{i,j} + \Delta w_{i,j}$

Delta rule

Adjusting perceptron weights

- $\Delta w_{i,j} = \alpha$ * [$teacher_i$ - $output_i$] * $input_j$
- $miss_i$ is ($teacher_i$ - $output_i$)
- Adjust each $w_{i,j}$ based on $input_j$ and $miss_i$
- Incremental learning.

Node biases

- A node's output is a weighted function of its inputs

- What is a bias?
- How can we learn the bias value?

- Answer: treat them like just another weight

Training biases (Θ)

- A node's output:
  - 1 if $w_1x_1 + w_2x_2 + \ldots + w_nx_n >= \Theta$ bias
  - 0 otherwise
- Rewrite
  - $w_1x_1 + w_2x_2 + \ldots + w_nx_n - \Theta >= 0$
  - $w_1x_1 + w_2x_2 + \ldots + w_nx_n + \Theta(-1) >= 0$
- Hence, the bias is just another weight.
- Just add one more input unit to the network topology

Perceptron Convergence Theorem

- If a set of <input, output> pairs are learnable (representable), the delta rule will find the necessary weights
  - in a finite number of steps
  - independent of initial weights
- However, a single layer perceptron can only learn linearly separable concepts
  - it works **iff** gradient descent works

Linear separability

- Consider a perceptron
- Its output is
  – 1, if $W_1X_1 + W_2X_2 > \Theta$
  – 0, otherwise
- In terms of feature space
  – hence, it can only classify examples if a line (hyperplane more generally) can separate the positive examples from the negative examples

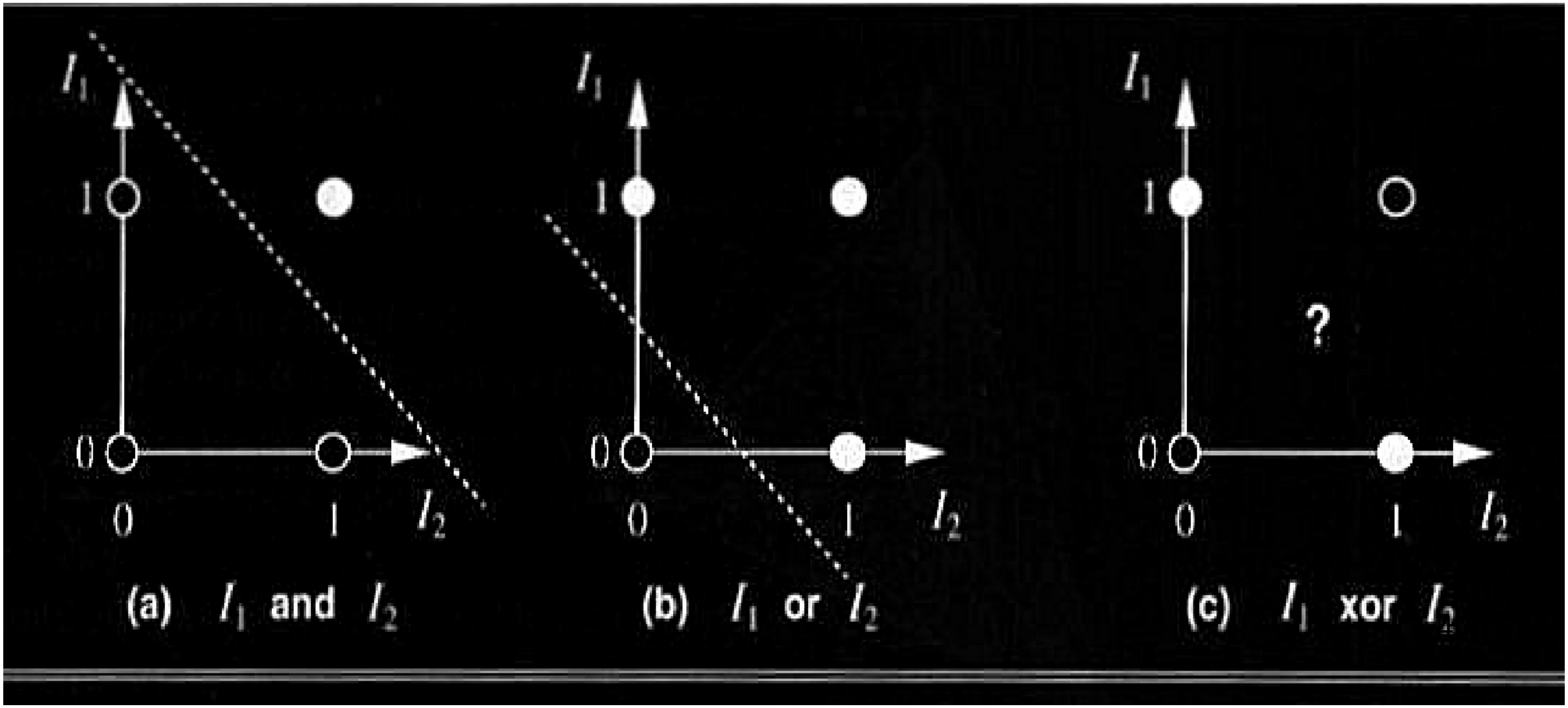
What can Perceptrons Represent ?

- Some complex Boolean function can be represented.

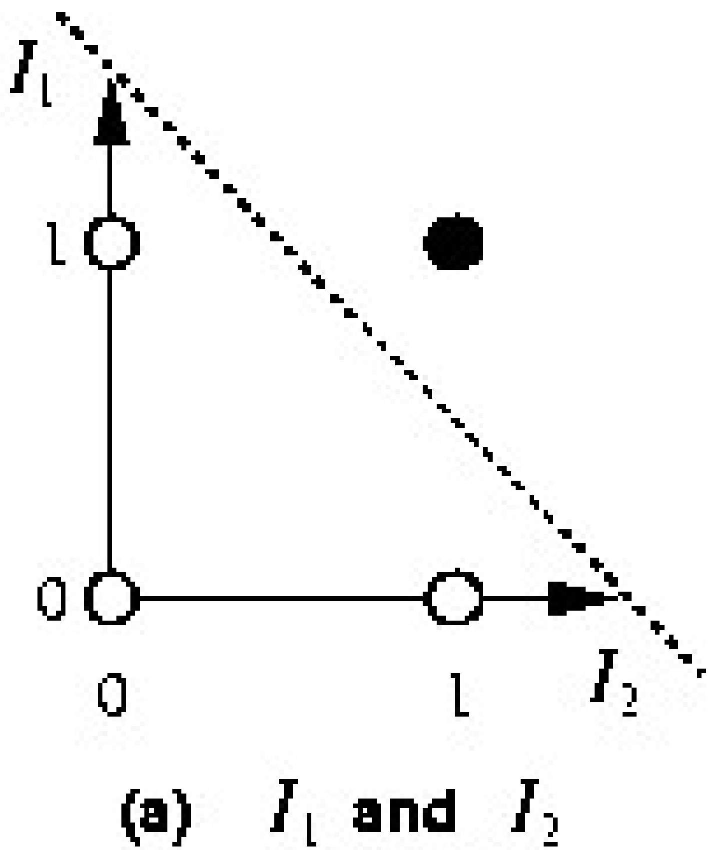
- Perceptrons are limited in the Boolean functions they can represent.

The Separability Problem and EXOR trouble

Figure 5:3. Linear Separability in Perceptrons

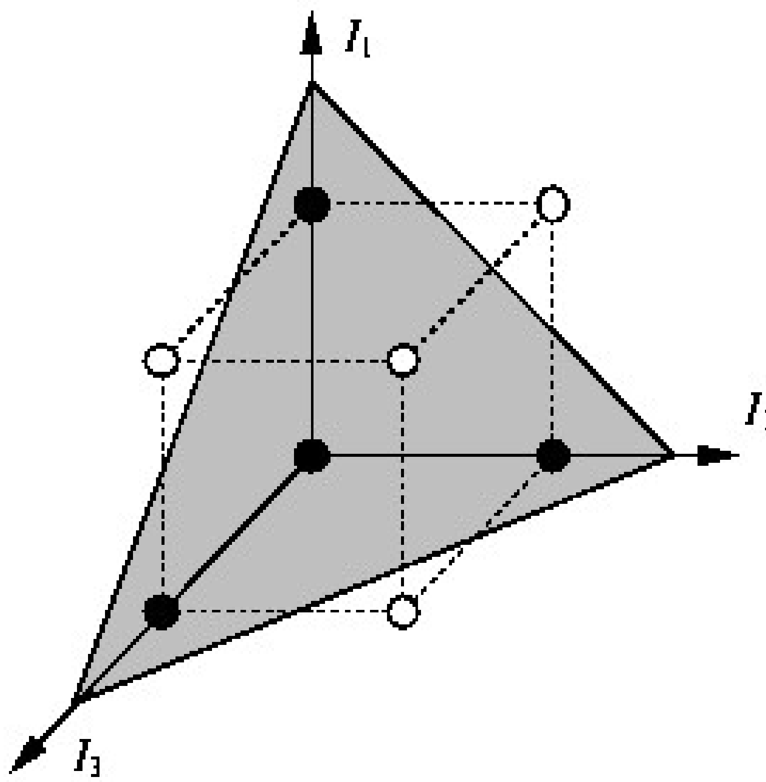


AND and OR linear Separators



Separation in n-1 dimensions

majority



(a) Separating plane

Example of
3Dimensional space

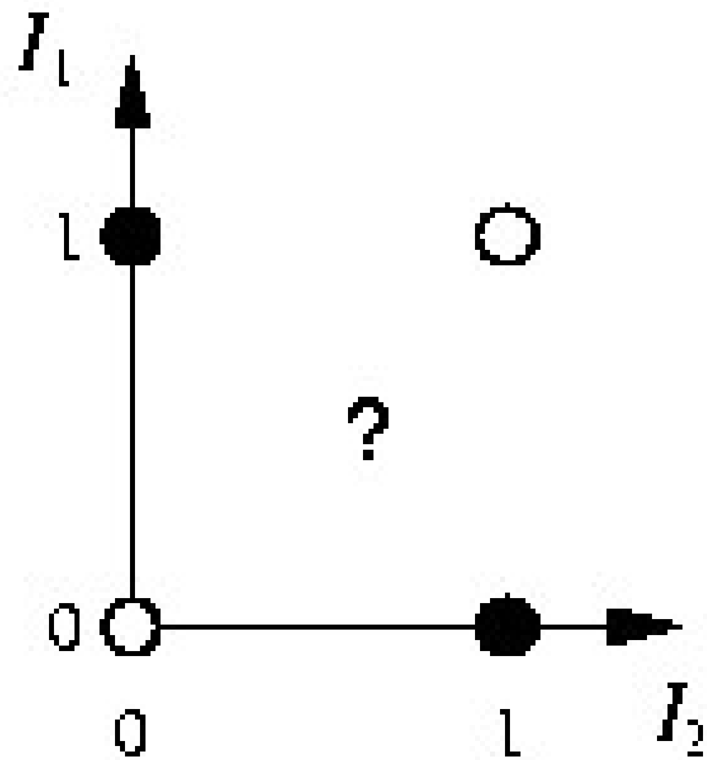
Perceptrons & XOR

- XOR function

Input1	Input2	Output
0	0	0
0	1	1
1	0	1
1	1	0

  – no way to draw a line to separate the positive from negative examples

How do we compute XOR?



(c) I_1 xor I_2

Learning Linearly Separable Functions (1)

What can these functions learn ?

Bad news:

- There are not many linearly separable functions.

Good news:

- There is a perceptron algorithm that will learn any linearly separable function, given enough training examples.

Learning Linearly Separable Functions (2)

- Initial network has a randomly assigned weights.

- Learning is done by making small adjustments in the weights to reduce the difference between the observed and predicted values.

- Main difference from the logical algorithms is the need to repeat the update phase several times in order to achieve convergence.

-Updating process is divided into epochs.

-Each epoch **updates all the weights** of the process.

Figure 5-4: The Generic Neural Network Learning Method: adjust the weights until predicted output values O and true values T agree

e are examples from set examples

```
function NEURAL-NETWORK-LEARNING(examples) returns network

  network ← a network with randomly assigned weights
  repeat
     for each e in examples do
        O ← NEURAL-NETWORK-OUTPUT(network, e)
        T ← the observed output values from e
        update the weights in network based on e, O, and T
     end
  until all examples correctly predicted or stopping criterion is reached
  return network
```

Two types of networks were compared for the restaurant problem

Examples of Feed-Forward Learning

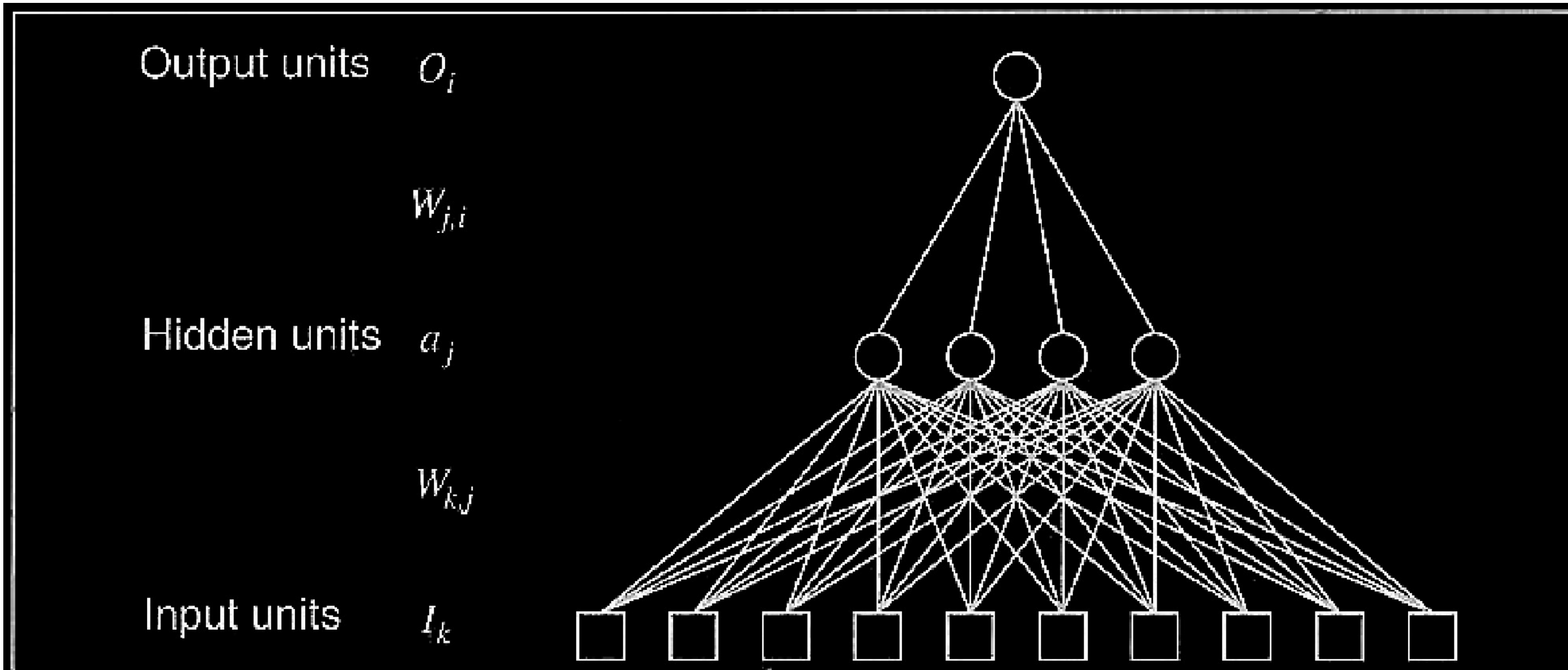
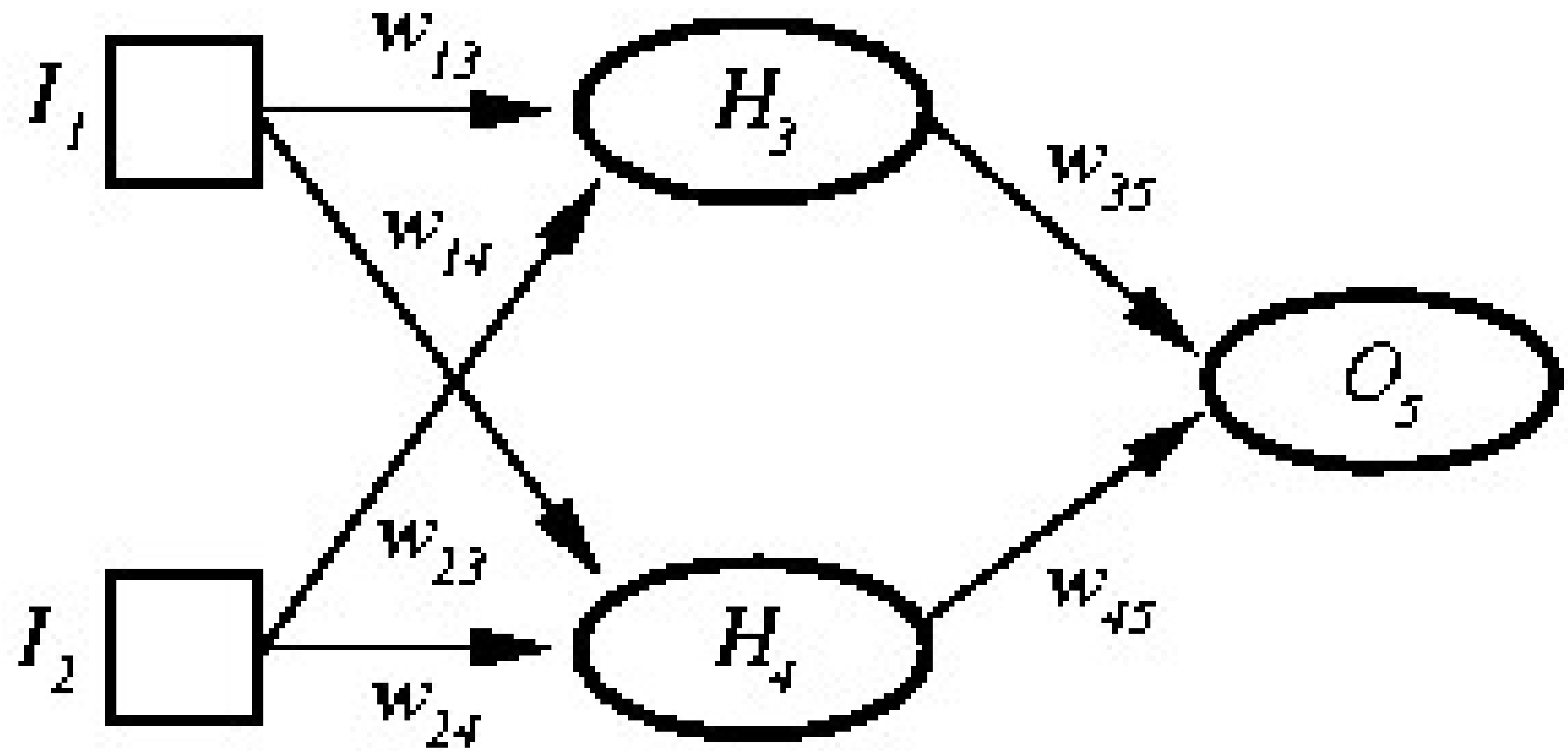


Figure 19.13 A two-layer feed-forward network for the restaurant problem.

Multi-Layer Neural Nets

Feed Forward Networks

2-layer Feed Forward example



Need for **Hidden Units**

- If there is one layer of enough hidden units, the input can be recoded (perhaps just memorized; example)

- This recoding allows any mapping to be represented

- Problem: How can the weights of the hidden units be trained?

N-layer FeedForward Network

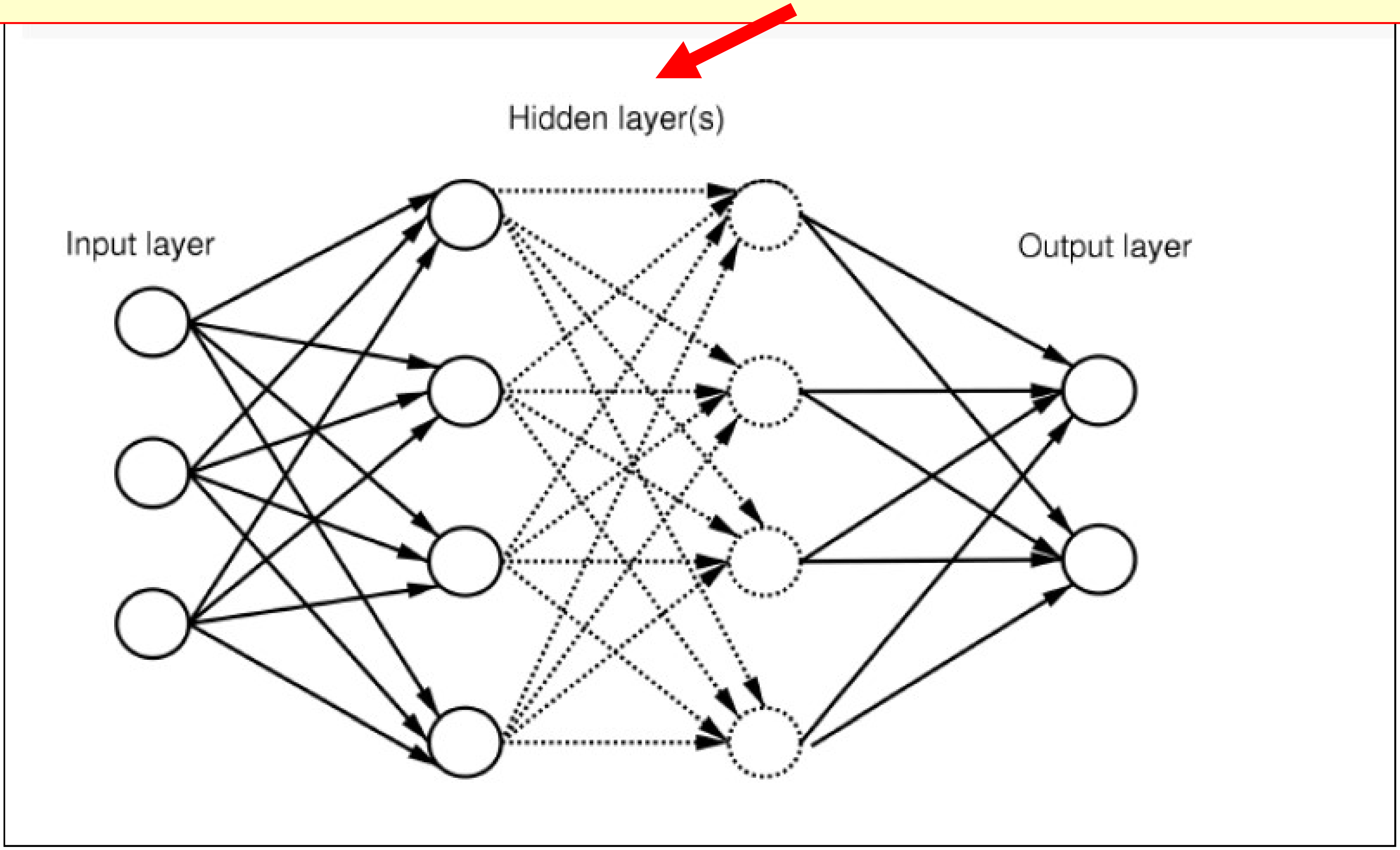
- Layer 0 is input nodes
- Layers 1 to N-1 are hidden nodes
- Layer N is output nodes
- All nodes at any layer k are connected to all nodes at layer k+1
- There are no cycles

Linear Threshold Units

2 Layer FF net with LTUs

- 1 output layer + 1 hidden layer
  - Therefore, 2 stages to "assign reward"
- Can compute functions with convex regions

Feed-forward NN with hidden layer



Evaluation of a ***Feedforward NN*** using software is easy

Set bias input neuron

```
void feedforward(float N_in[NIN],float N_hid[NHID],float N_out[NOUT])
{ int i,j;
  N_in[NIN-1] = 1.0; // set bias input neuron
  for (i=0; i<NHID-1; i++) // calculate activation of hidden neurons
  { N_hid[i] = 0.0;
    for (j=0; j<NIN; j++)
      N_hid[i] += N_in[j] * w_in[j][i];
    N_hid[i] = sigmoid(N_hid[i]);
  }
  N_hid[NHID-1] = 1.0; // set bias hidden neuron
  for (i=0; i<NOUT; i++) // calculate output neurons
  { N_out[i] = 0.0;
    for (j=0; j<NHID; j++)
      N_out[i] += N_hid[j] * w_out[j][i];
    N_out[i] = sigmoid(N_out[i]);
  }
}
```

Calculate activation of hidden neurons

Calculate output neurons

Take from hidden neurons and multiply by weights

Backpropagation Networks

Introduction to Backpropagation

- In 1969 a method for learning in multi-layer network, Backpropagation, was invented by Bryson and Ho.

- The Backpropagation algorithm is a sensible approach for dividing the contribution of each weight.

- Works basically the same as perceptrons

Backpropagation Learning Principles: Hidden Layers and Gradients

There are two **differences for the updating rule :**

1) The activation of the hidden unit is used instead of the input value.

2) The rule contains a term for the gradient of the activation function.

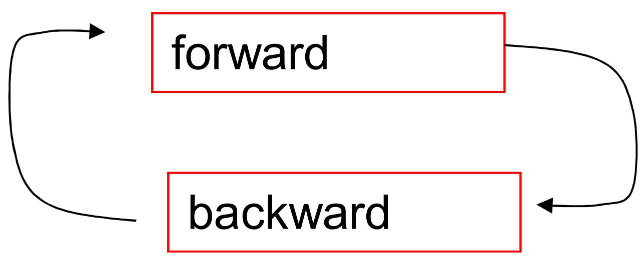
Backpropagation Network training

- 1. Initialize network with random weights
- 2. For all training cases (called examples):
  - **a.** Present training inputs to network and calculate output
  - **b.** For all layers (starting with output layer, back to input layer):
    - i. Compare network output with correct output (error function)
    - ii. Adapt weights in current layer

This is what you want

Backpropagation Learning Details

- Method for learning weights in feed-forward (FF) nets
- Can’t use Perceptron Learning Rule
  - no teacher values are possible for hidden units
- Use gradient descent to minimize the error
  - Propagate deltas to adjust for errors
    backward from outputs
    to hidden layers
    to inputs



Backpropagation Algorithm – Main Idea – error in hidden layers

The ideas of the algorithm can be summarized as follows :

1. Computes the **error term for the output units** using the observed error.

2. From output layer, repeat
   - propagating the error term back to the previous layer and
   - updating the weights between the two layers

   until the earliest hidden layer is reached.

Backpropagation Algorithm

- Initialize weights (typically random!)
- Keep doing epochs
  - For each example **e** in training set do
    - **forward pass** to compute
      - O = neural-net-output(network,e)
      - miss = (T-O) at each output unit
    - **backward pass** to calculate deltas to weights
    - update all weights
  - end
- until tuning set error stops improving

Forward pass explained earlier

Backward pass explained in next slide

Backward Pass

- Compute deltas to weights
  - from hidden layer
  - to output layer

- Without changing any weights (yet), compute the actual contributions
  - within the hidden layer(s)
  - and compute deltas

Updating hidden-to-output

- We have teacher supplied desired values

- $\text{delta}_{wji} = \alpha * a_j * (T_i - Oi) * g'(in_i)$
  $= \alpha * a_j * (T_i - O_i) * O_i * (1 - O_i)$

  – for sigmoid the derivative is, $g'(x) = g(x) * (1 - g(x))$

derivative

alpha

Here we have general formula with derivative, next we use for sigmoid

miss

Updating interior weights

- Layer k units provide values to all layer k+1 units
  - "miss" is ***sum of misses*** from all units on k+1
  - $\text{miss}_j = \Sigma\, [\, a_i(1-a_i)\,(T_i - a_i)\, w_{ji}\,]$
  - weights coming into this unit are adjusted based on their contribution

$\text{delta}_{kj} = \alpha * I_k * a_j * (1 - a_j) * \text{miss}_j$

For layer k+1

Compute deltas

How do we pick α?

Small for slow, conservative learning

How Many Hidden Layers?

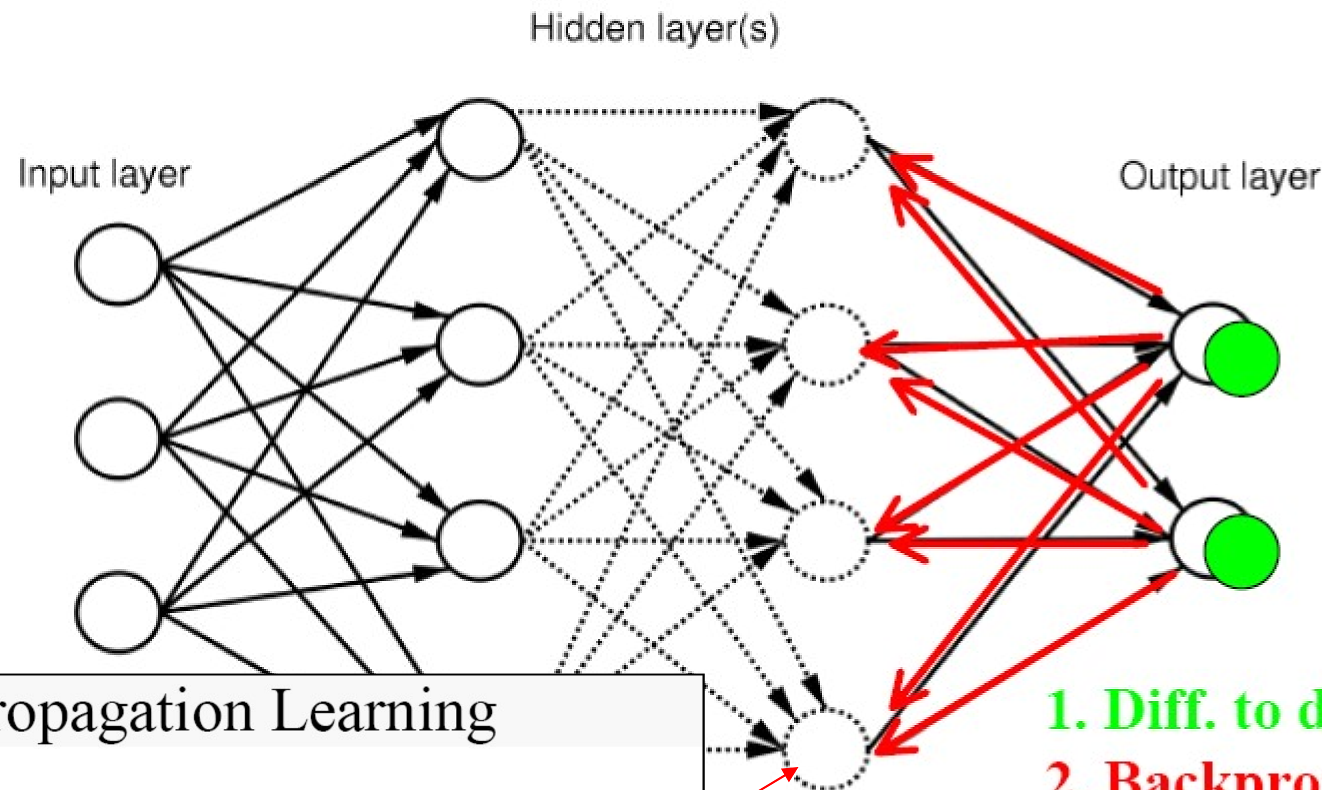
- Usually just one (i.e., a 2-layer net)

- How many hidden units in the layer?
  - Too few ==> can't learn
  - Too many ==> poor generalization

How big a training set?

- Determine your target error rate, e
- Success rate is $1- e$
- Typical training set approx. n/e, where n is the number of weights in the net
- Example:
  - $e = 0.1$, $n = 80$ weights
  - training set size 800

Backpropagation Learning



1. Diff. to desired values

2. Backprop output layer

9

Backpropagation Learning

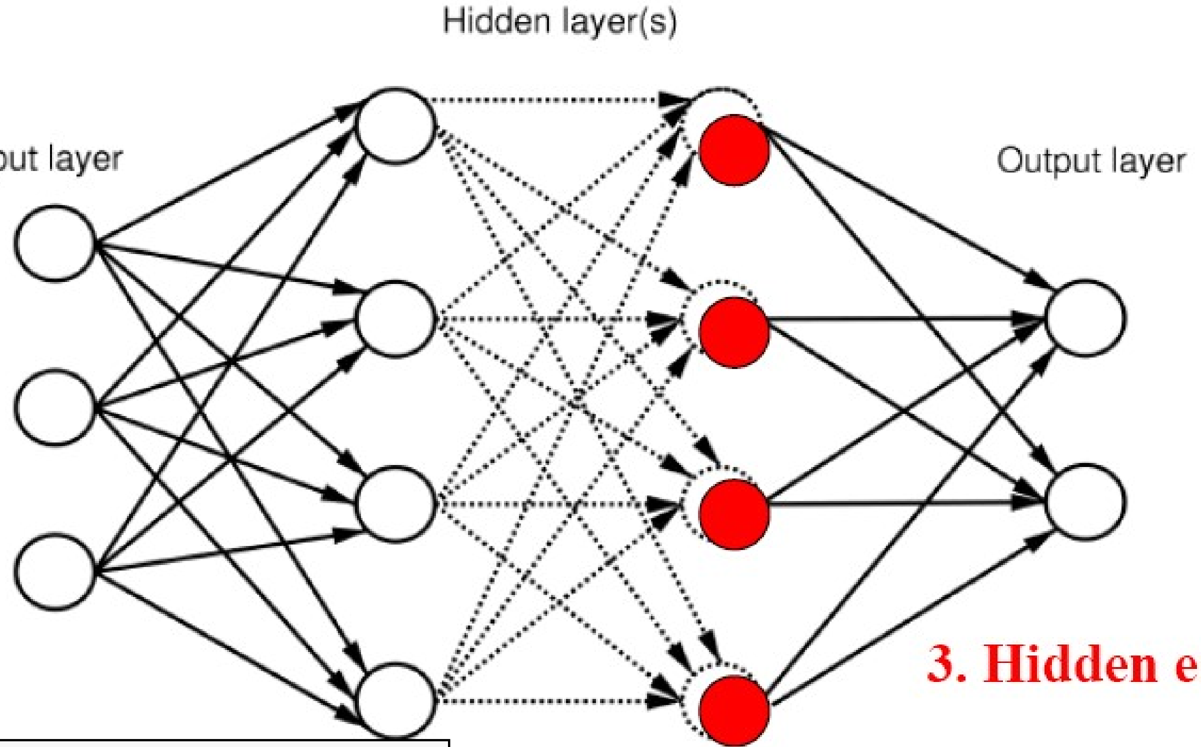
$$E_{\text{out } i} = d_{\text{out } i} - \text{out}_i$$

$$E_{\text{total}} = \sum_{i=0}^{\text{num}(n_{\text{out}})} E^2_{\text{out } i}$$

$$E_{\text{hid } i} = \sum_{k=1}^{\text{num}(n_{\text{out}})} E_{\text{out } k} \cdot w_{\text{out } i,k}$$

$$\text{diff}_{\text{hid } i} = E_{\text{hid } i} \cdot (1 - o(n_{\text{hid } i})) \cdot o(n_{\text{hid } i})$$

Backpropagation Learning



10

Backpropagation Learning

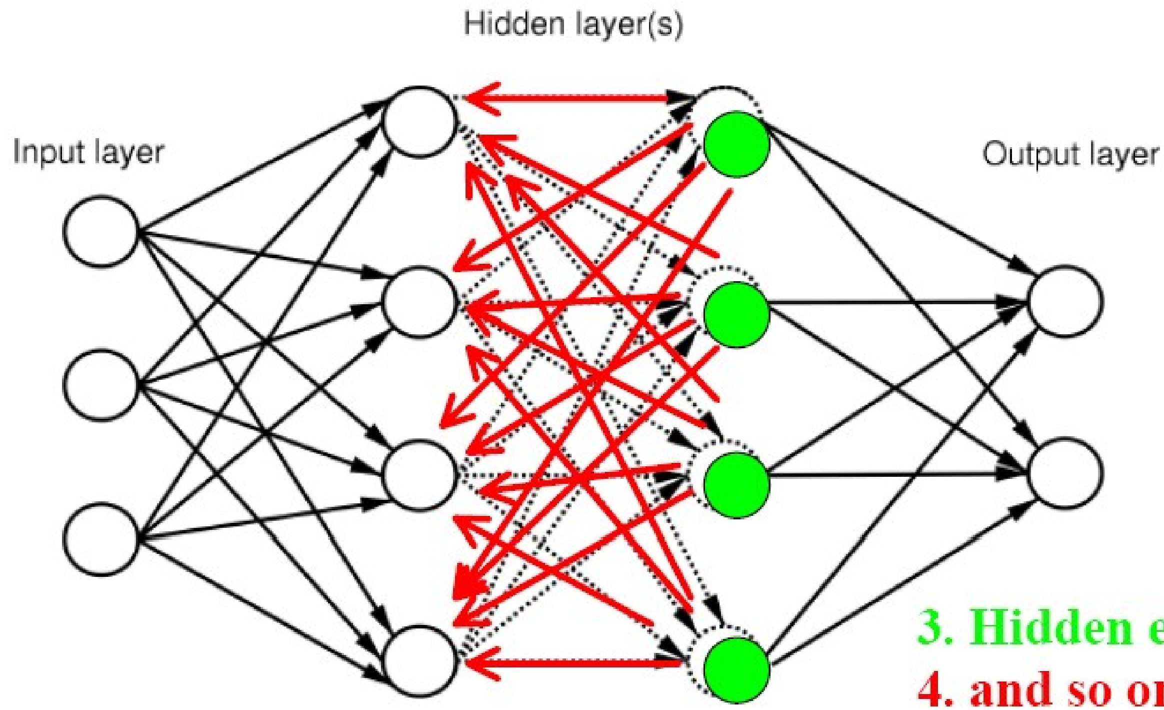
$E_{out\ i} = d_{out\ i} - out_i$

$$E_{total} = \sum_{i=0}^{num(n_{out})} E^2_{out\ i}$$

$$E_{hid\ i} = \sum_{k=1}^{num(n_{out})} E_{out\ k} \cdot w_{out\ i,k}$$

$diff_{hid\ i} = E_{hid\ i} \cdot (1 - o(n_{hid\ i})) \cdot o(n_{hid\ i})$

Backpropagation Learning



3. Hidden error values

4. and so on …

Backpropagation Learning

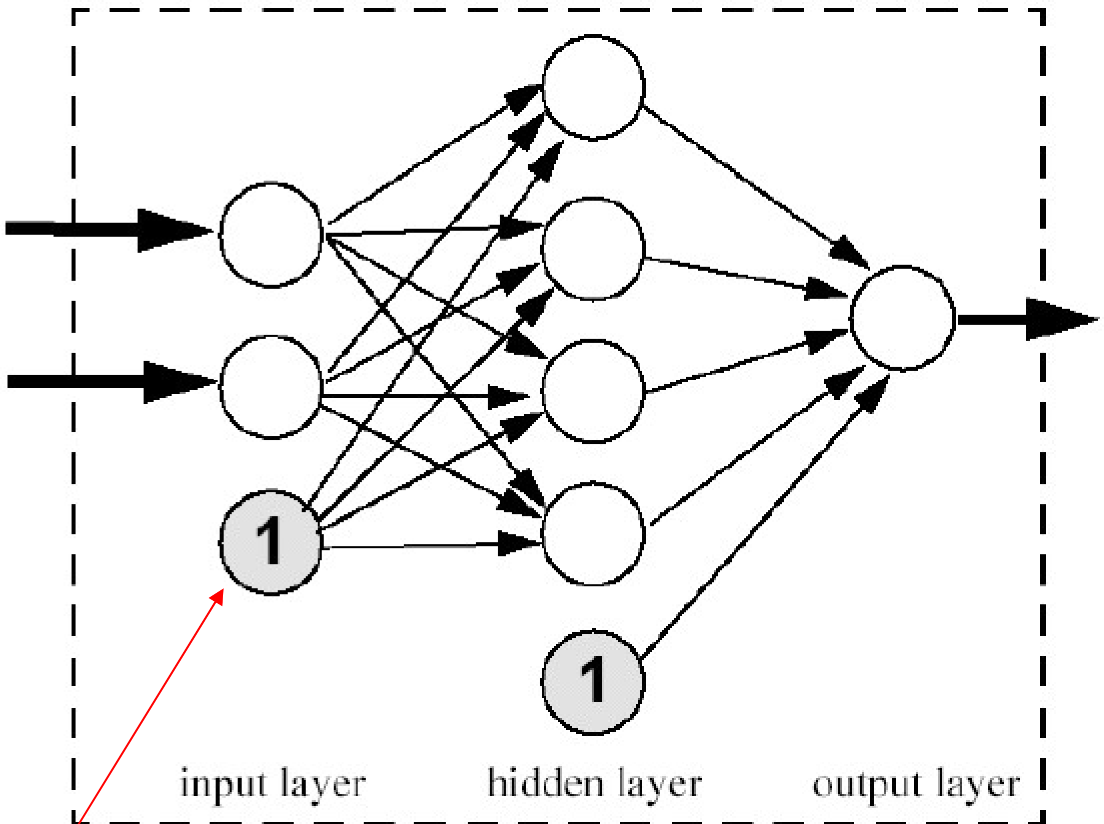
$$E_{out\ i} = d_{out\ i} - out_i$$

$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\ i}^2$$

$$E_{hid\ i} = \sum_{k=1}^{num(n_{out})} E_{out\ k} \cdot w_{out\ i,k}$$

$$diff_{hid\ i} = E_{hid\ i} \cdot (1 - o(n_{hid\ i})) \cdot o(n_{hid\ i})$$

Bias Neurons in Backpropagation Learning



Bias neurons

bias neuron in input layer

The general Backpropagation Algorithm for updating weights in a multilayer network

Repeat until convergent

Here we use alpha, the learning rate

Go through all examples

Run network to calculate its output for this example

function BACK-PROP-UPDATE(*network*, *examples*, α) **returns** a network with modified weights
  inputs: *network*, a multilayer network
    examples, a set of input/output pairs
    α, the learning rate

  repeat
    for each e **in** *examples* **do**
      / Compute the output for this example */*
      $\mathbf{O} \leftarrow$ RUN-NETWORK(*network*, $\mathbf{I}^e$)
      / Compute the error and Δ for units in the output layer */*
      $\mathbf{Err}^e \leftarrow \mathbf{T}^e - \mathbf{O}$
      / Update the weights leading to the output layer */*
      $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times Err_i^e \times g'(in_i)$
      for each subsequent layer **in** *network* **do**
        / Compute the error at each node */*
        $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i} \Delta_i$
        / Update the weights leading into the layer */*
        $W_{k,j} \leftarrow W_{k,j} + \alpha \times I_k \times \Delta_j$
      end
    end
  until *network* has converged
  return *network*

Compute the error in output

Update weights to output layer

Compute error in each hidden layer

Update weights in each hidden layer

Return learned network

Examples and Applications of ANN

Neural Network in Practice

NNs are used for classification and function approximation or mapping problems which are:

- Tolerant of some imprecision.
- Have lots of training data available.
- Hard and fast rules cannot easily be applied.

NETalk (1987)

- Mapping character strings into phonemes so they can be pronounced by a computer
- Neural network trained how to pronounce each letter in a word in a sentence, given the three letters before and three letters after it in a window
- Output was the correct phoneme
- Results
  - 95% accuracy on the training data
  - 78% accuracy on the ***test set***

Other Examples

- Speech Recognition (Waibel, 1989)
- Character Recognition (LeCun et al., 1989)
- Face Recognition (Mitchell)

Feed-forward vs. Interactive Nets

- Feed-forward
  - activation propagates in one direction
  - We usually focus on this

- Interactive
  - activation propagates forward & backwards
  - propagation continues until equilibrium is reached in the network
  - We do not discuss these networks here, complex training. May be unstable.

Ways of learning with an ANN

- Add nodes & connections
- Subtract nodes & connections
- Modify connection weights
  - current focus
  - can simulate first two

- I/O pairs:
  - given the inputs, what should the output be? ["typical" learning problem]

More Neural Network Applications

- May provide a model for massive parallel computation.

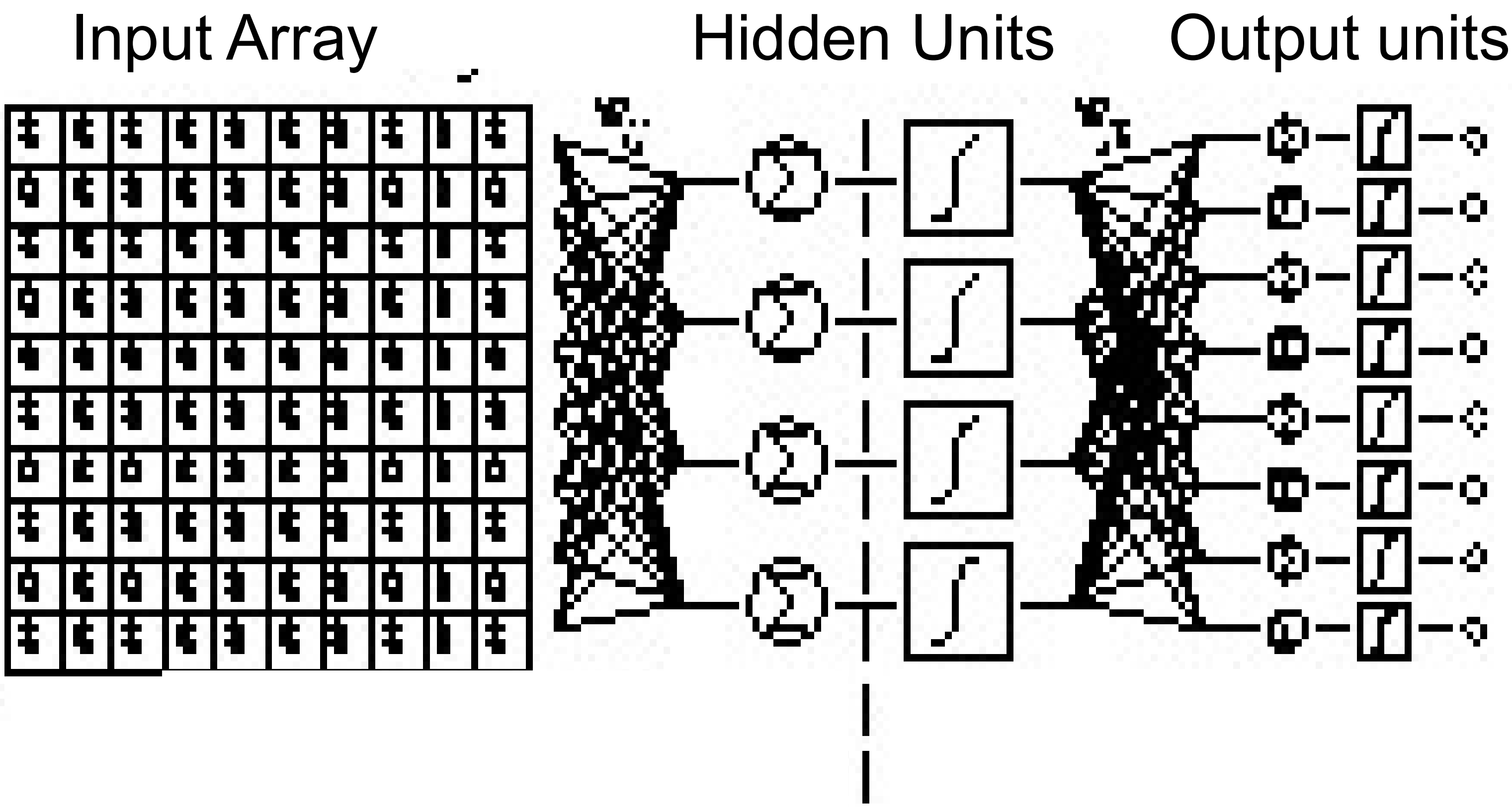
- More successful approach of “parallelizing” traditional serial algorithms.

- Can compute any computable function.

- Can do everything a normal digital computer can do.

- Can do even more under some impractical assumptions.

Neural Network Approaches



Summary

- Neural network is a computational model that simulate some properties of the human brain.

- The connections and nature of units determine the behavior of a neural network.

- Perceptrons are feed-forward networks that can only represent linearly separable functions.

Summary

- Given enough units, any function can be represented by Multi-layer feed-forward networks.

- Backpropagation learning works on multi-layer feed-forward networks.

- Neural Networks are widely used in developing artificial learning systems.

References

- Russel, S. and P. Norvig (1995). Artificial Intelligence - A Modern Approach. Upper Saddle River, NJ, Prentice Hall.

- Sarle, W.S., ed. (1997), *Neural Network FAQ, part 1 of 7: Introduction,* periodic posting to the Usenet newsgroup comp.ai.neural-nets,
URL: ftp://ftp.sas.com/pub/neural/FAQ.html