

# **Class 9: Learning II**

Selmer Bringsjord

June 15, 1999

- Logistics
  - email Selim for grades
  - too many late things: makes it rather hard for us!
  - Project 3: you should know the score exactly now
  - Final: yes, again, there'll be a practice final

- The Past/The Future
  - Minsky and Pappert kill connectionism
    - \* they focus on perceptrons
    - \* what about multi-layer nets?
  - approx. 1980 connectionism roars back
  - McCarthy: “two horses in the race”
    - \* (the McCarthy-Minsky split)
  - calls for hybrid approaches (Har-nad, Pollock, Bringsjord)
  - e.g., ALVINN

- **Artificial** Neural Nets

- Transfer function of a “neuron” computed in 3-step process: *fig*

- 1. compute the weighted input

$$in_i = \sum_{j=1}^n w_{ij}x_j$$

- 2. convert net input to activation level via some activation function *g*:

$$g(in_i)$$

- 3. send activation value as output signal  $a_i$ , commonly via

$$a_i = \begin{cases} 1 & \text{if } in_i > T \\ 0 & \text{otherwise} \end{cases}$$

- Standardizing Thresholds to 0
  - verify the equality:

$$step_t\left(\sum_{j=1}^n w_{ij}x_j\right) = step_{a_0}\left(\sum_{j=0}^n w_{ij}x_j\right)$$

where  $a_0$  is fixed at -1 and whose weight is the former threshold  $t$

- verify equivalence through AND  
( $-1 \times 1.5 = -1.5 \dots$ )

- Logic Gates
  - AND OR NOT *fig write*
  - recall Boolean functions via truth-tables
  - $\forall$  Boolean *f exists* a corresponding formula  $\alpha$  *write*
  - any Boolean *f* can be done w/ just OR and NOT, e.g.
    - \* DeMorgan's Laws *write*

- Feed-Forward vs. Recurrent Nets
  - Recurrent
    - \* Hopfield Nets
      - bidirectional connections w/ symmetric weights
      - $g$  is sign function
      - activation levels +1 or -1
    - \* Boltzmann Machines (MONA-LISA<sub>browser</sub>)
      - symmetric weights
      - hidden layers
      - stochastic activation function

- Feed-Forward vs. Recurrent Nets
  - Feed-Forward *fig*
    - \* Single-Layer vs. Multi-Layer
      - the former: Perceptrons
      - the latter: can do all Boolean functions

- Perceptrons *figs*
  - can't do all Boolean functions
  - can do majority function
    - \* majority function in truth-tables  
*write*
    - \* majority function in perceptron
      - $w_j = 1; t = \frac{n}{2}$  – (diagram)
  - can only do linearly separable functions (e.g., not XOR<sub>*write*</sub>)

- Perceptrons and Linearly Separable Functions – Visually Put
  - easy to visualize with 2 inputs *fig*
  - still no problem with 3 inputs *fig*
  - the restaurant problem — can it be visualized?

- Learning Algorithms
  - CBH<sub>fig</sub>
  - weight update rule for perceptrons:

$$Err = T - O$$

$$w_j \leftarrow w_j + \alpha \times I_j \times Err$$

where  $\alpha$  is a constant called the **learning rate**

- Learning Algorithms (con.)
  - perceptrons vs. decision trees on majority  $f$  and restaurant problem *fig*
  - See back prop for restaurant problem: better
  - weight updating in back prop:

$$Err_i = T_i - O_i$$

$$w_{j,i} \leftarrow w_{j,i} + \alpha \times a_j \times Err_i \times g'(in_i)$$

- comparison on restaurant problem: no magic *fig*

- “Learning Something From Nothing”  
– Zoombinis!...

$$H \wedge D \models C$$

1	$\forall x(Q(x) \Leftarrow C(x))$	?
2	$C(a)$	Description
3	$Q(a)$	1, 2

- EBL
  - the caveman example
  - structure:

$$H \wedge D \models C$$

$$B \models H$$

- RBL

- the language example

- structure:

$$H \wedge D \models C$$

$$B \wedge D \models H$$

- verify in Hyperproof<sub>Hyperproof</sub>:

$$\left| \begin{array}{l} 1 \mid \forall x, y, n, l ((N(x, n) \wedge N(y, n) \wedge L(y, l)) \Rightarrow L(y, l)) \\ 2 \mid N(f, b) \wedge L(f, p) \\ 3 \mid \forall x (N(x, b) \Rightarrow L(x, p)) \end{array} \right| \begin{array}{l} B \\ D \\ = H; 1, 2 \end{array} \right|$$

- KBIL powered by ILP

- the equation this time is

$$B \wedge H \wedge D \models C$$

- and we attempt to solve for  $H$

- the family example *fig*

- $Q$  is *Grandparent*

- $D$  is ...

- $H$  is ...

- $B$  is ...

- Finally: New Predicates from Nowhere
  - Inverse Resolution to find  $H$  *figs*

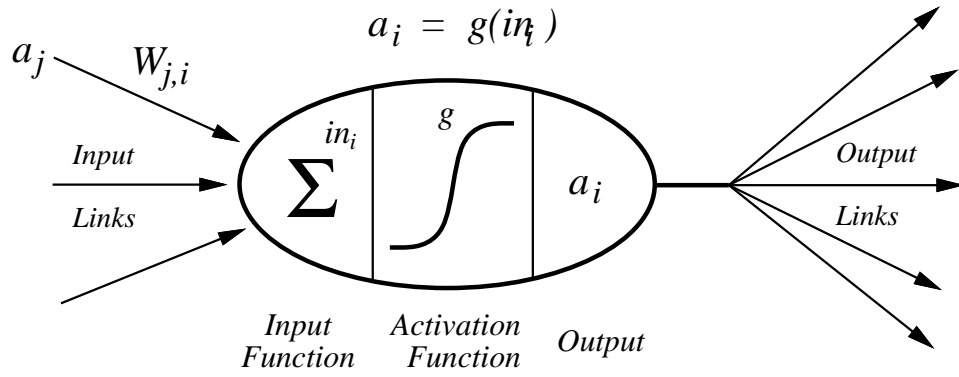


Figure 1:

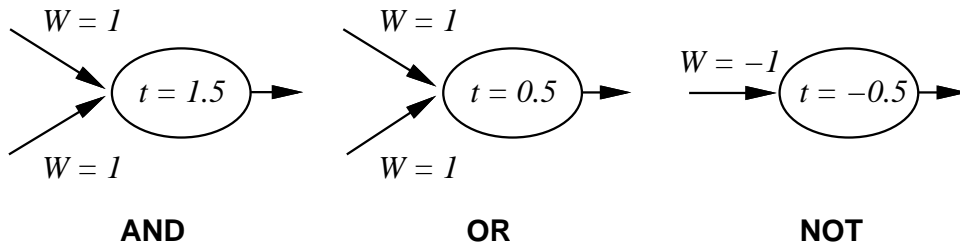


Figure 2:

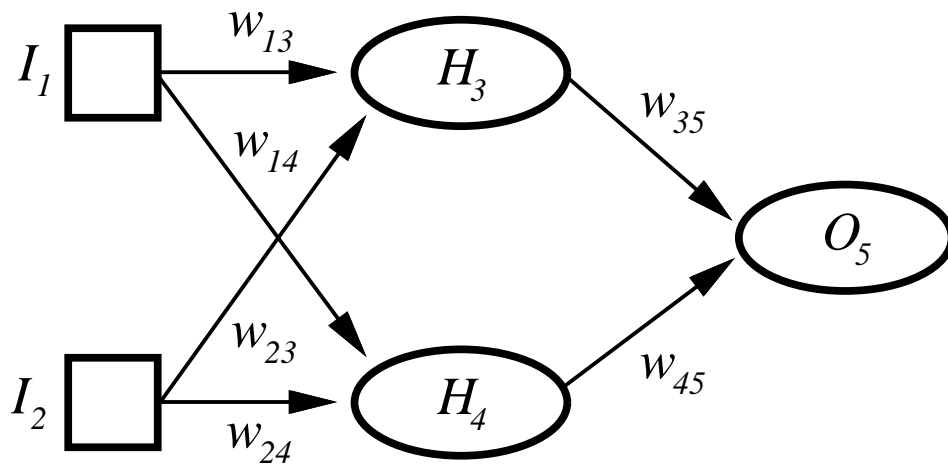


Figure 3:

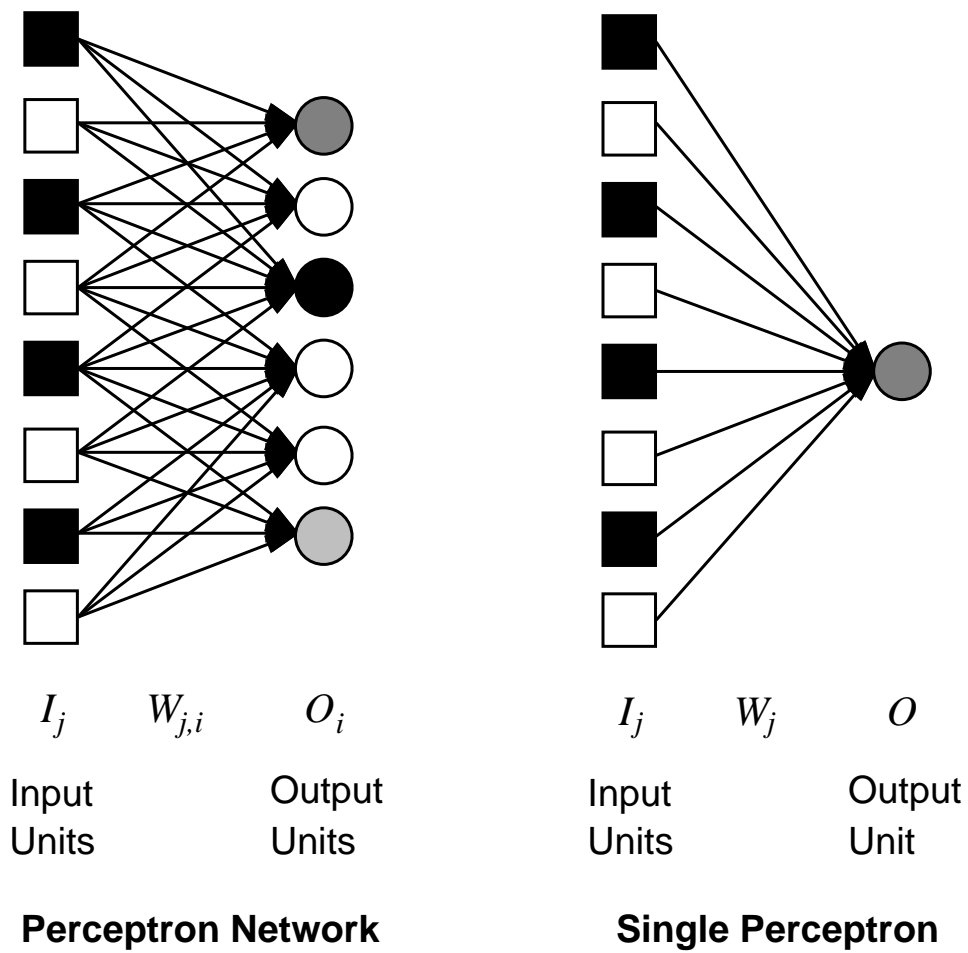


Figure 4:

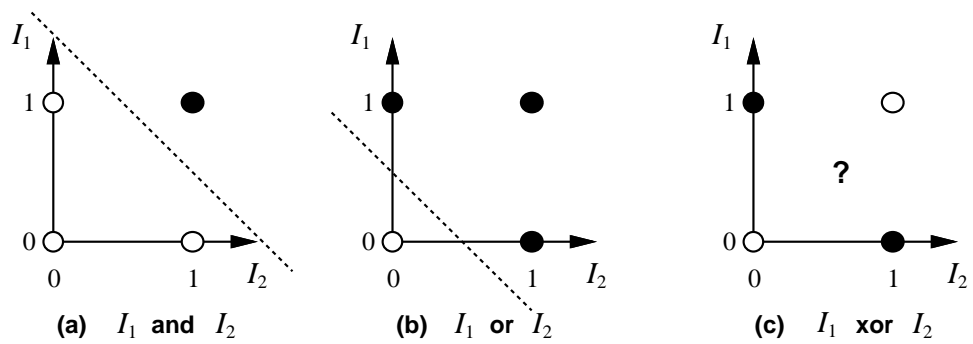


Figure 5:

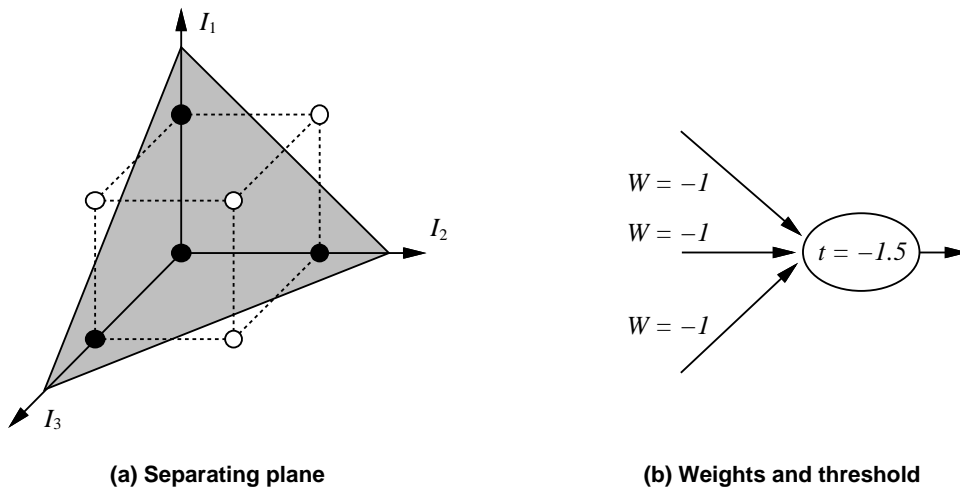


Figure 6:

```

function NEURAL-NETWORK-LEARNING(examples) returns network
  network  $\leftarrow$  a network with randomly assigned weights
  repeat
    for each e in examples do
      O  $\leftarrow$  NEURAL-NETWORK-OUTPUT(network, e)
      T  $\leftarrow$  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

```

Figure 7:

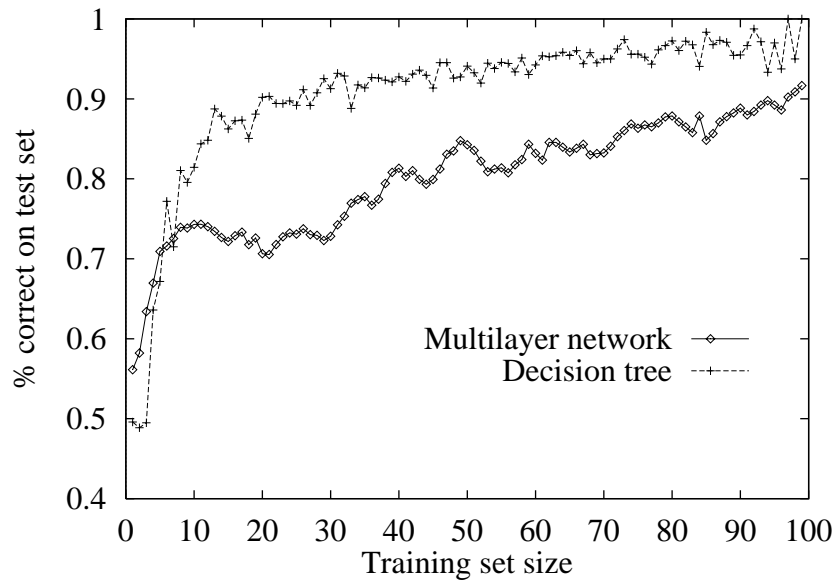


Figure 8:

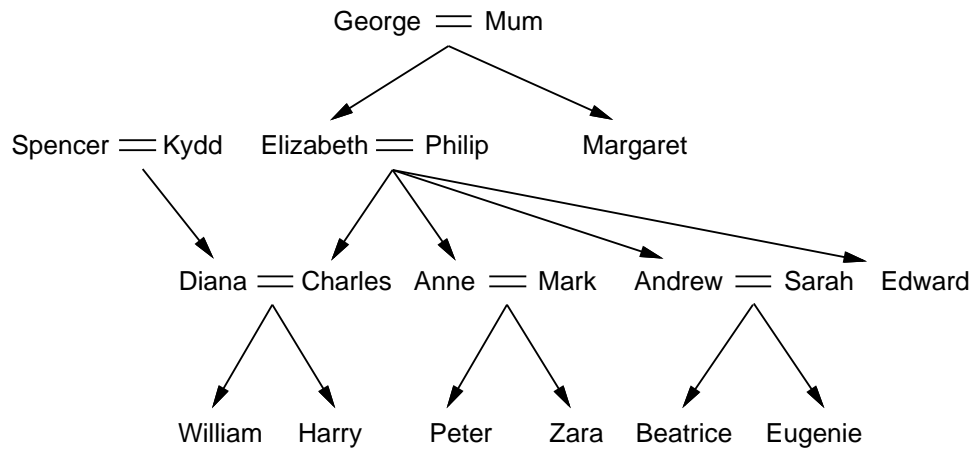


Figure 9:

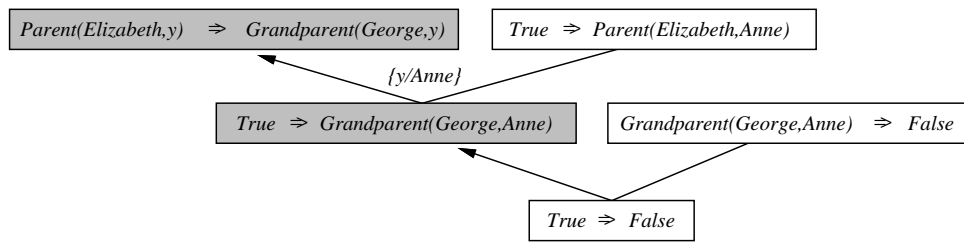


Figure 10:

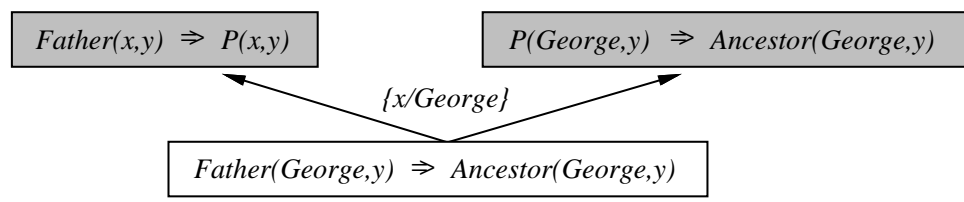


Figure 11: