Computing with Nonlinear Perceptrons

Question: What can a *nonlinear perceptron* compute?

Question: What's a *nonlinear perceptron* again?

Answer: A layered network with a nonlinear activation function.

Answer: Any computable mapping from its inputs to its outputs.

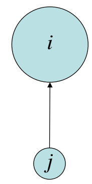
Question: WTF does *that* mean?

Answer: Here we go…

1

Computing Mappings with Layered Networks

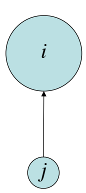
Consider a node, i, with one input, j.



2

Computing Mappings with Layered Networks

Let j take a value from $-\infty$ to $+\infty$.

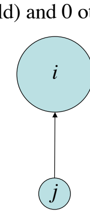


$j \in -\infty \ldots \infty$.

3

Computing Mappings with Layered Networks

Let a_i, (the activation of i), be 1 if $j > \Theta$ (where Θ is a threshold) and 0 otherwise:



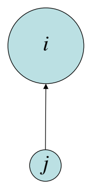
$$a_i = \begin{cases} 1 \text{ if } j > \Theta \\ 0 \text{ otherwise} \end{cases}$$

$j \in -\infty \ldots \infty$.

4

Computing Mappings with Layered Networks

Let a_i, (the activation of i), be 1 if $j > \Theta$ (where Θ is a threshold) and 0 otherwise:



$j \in -\infty \ldots \infty.$

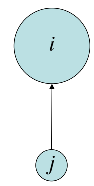
$$a_i = \begin{cases} 1 \text{ if } j > \Theta \\ 0 \text{ otherwise} \end{cases}$$

i is a *binary threshold* unit (BTU).

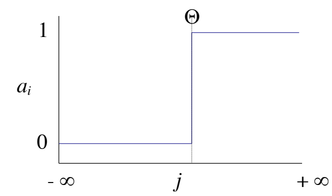
5

Computing Mappings with Layered Networks

Binary threshold unit (BTU): $a_i = \begin{cases} 1 \text{ if } j > \Theta \\ 0 \text{ otherwise} \end{cases}$



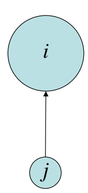
$j \in -\infty \ldots \infty.$



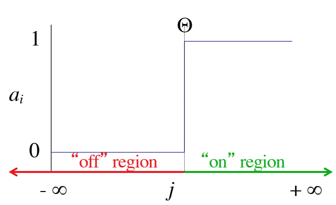
6

Computing Mappings with Layered Networks

i divides its *input space* (the line, j) into two regions, an "on" region and an "off" region, at a single point, Θ.



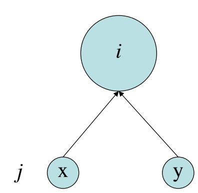
$j \in -\infty \ldots \infty.$



7

Computing Mappings with Layered Networks

What if i had two inputs (j = x and j = y)?

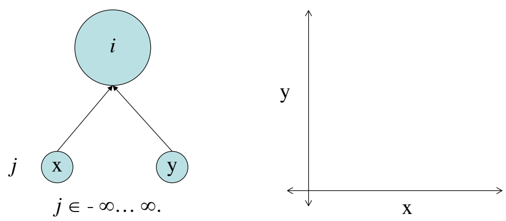


$j \in -\infty \ldots \infty.$

8

Computing Mappings with Layered Networks

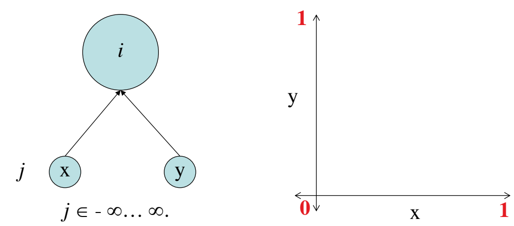
What if i had two inputs (j = x and j = y)?



Now, i's input space is not a 1-D line, but the 2-D plane, x,y.

9

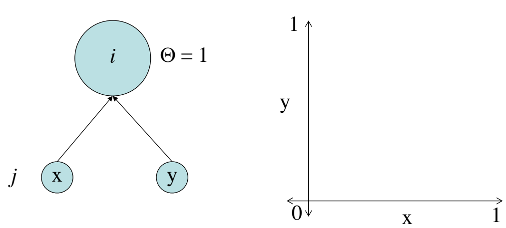
Computing Mappings with Layered Networks



10

Computing Mappings with Layered Networks

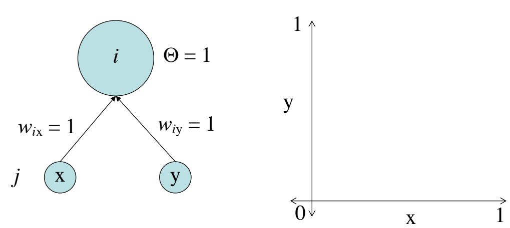
Let's assign Θ the value 1…



11

Computing Mappings with Layered Networks

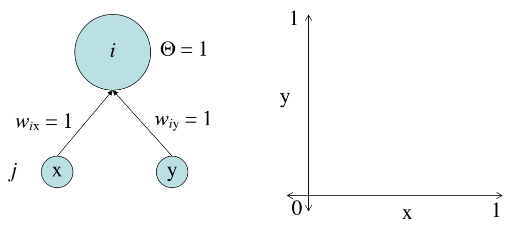
…and the connection weights from x and y each the value 1.



12

Computing Mappings with Layered Networks

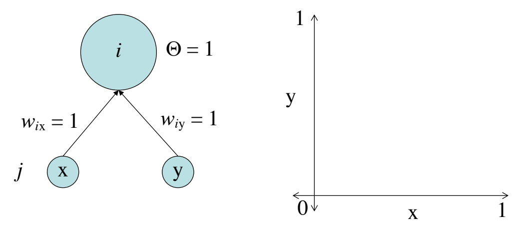
Recall the input rule: $n_i = \Sigma_j w_{ij} a_j$.



13

Computing Mappings with Layered Networks

Recall the input rule: $n_i = \Sigma_j w_{ij} a_j$.

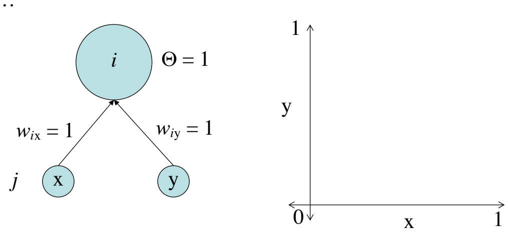


And the activation rule: $a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases}$

14

Computing Mappings with Layered Networks

And let's see what happens to *i* for various values of x and y…

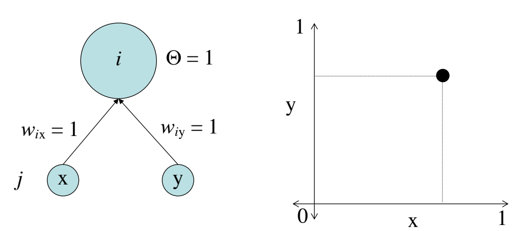


$n_i = \Sigma_j w_{ij} a_j =$ $\quad a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} =$

15

Computing Mappings with Layered Networks

…say, x = 0.75 and y = 0.75...

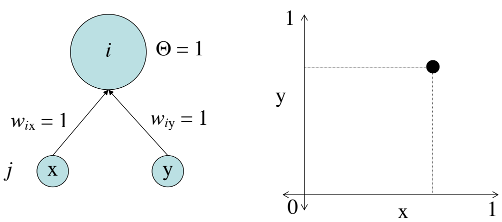


$n_i = \Sigma_j w_{ij} a_j =$ $\quad a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} =$

16

Computing Mappings with Layered Networks

…say, x = 0.75 and y = 0.75: $n_i = (1 * .75) + (1 * .75) = 1.5$.



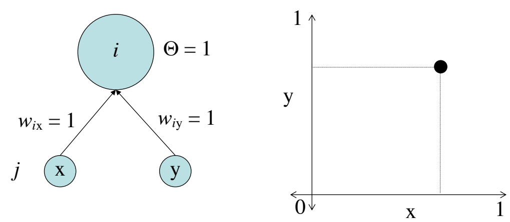
$n_i = \Sigma_j w_{ij} a_j = 1.5$

$a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} =$

17

Computing Mappings with Layered Networks

…say, x = 0.75 and y = 0.75: 1.5 is > Θ (= 1), so $a_i = 1$.



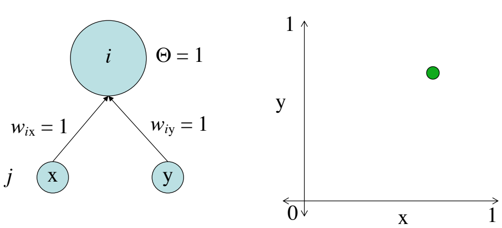
$n_i = \Sigma_j w_{ij} a_j = 1.5$

$a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} = 1$

18

Computing Mappings with Layered Networks

0.75, 0.75 is an "on" spot for i, so let's color it green



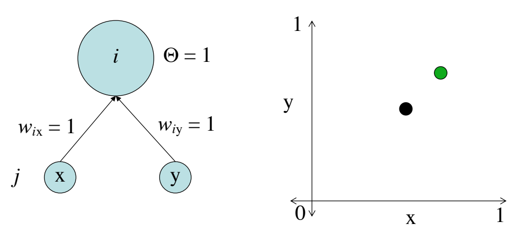
$n_i = \Sigma_j w_{ij} a_j = 1.5$

$a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} = 1$

19

Computing Mappings with Layered Networks

x = 0.5, y = 0.5?



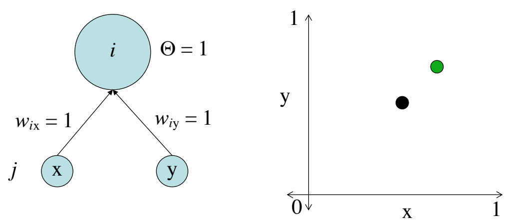
$n_i = \Sigma_j w_{ij} a_j =$

$a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} =$

20

Computing Mappings with Layered Networks

x = 0.5, y = 0.5? $n_i = (1 * .5) + (1 * .5) = 1.0$

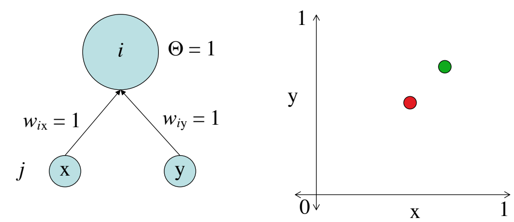


$n_i = \Sigma_j w_{ij} a_j = 1$ $a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} =$

21

Computing Mappings with Layered Networks

x = 0.5, y = 0.5? 1.0 is *not* greater than Θ (= 1.0), so $a_i = 0$.

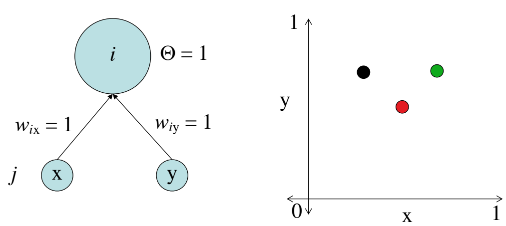


$n_i = \Sigma_j w_{ij} a_j = 1$ $a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} = 0$

22

Computing Mappings with Layered Networks

x = 0.25, y = 0.75?

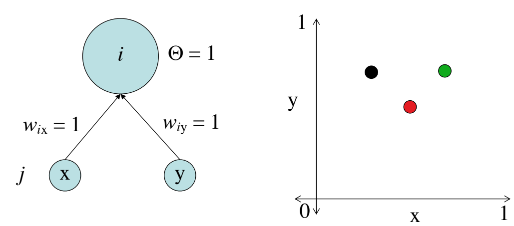


$n_i = \Sigma_j w_{ij} a_j =$ $a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} =$

23

Computing Mappings with Layered Networks

x = 0.25, y = 0.75? $n_i = 1.0$;

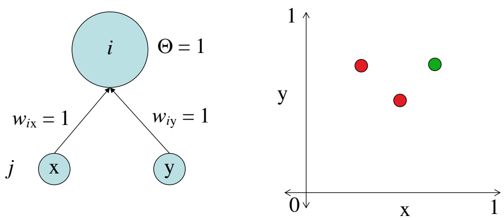


$n_i = \Sigma_j w_{ij} a_j = 1.0$ $a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} =$

24

Computing Mappings with Layered Networks

x = 0.25, y = 0.75? $n_i = 1.0$; $a_i = 0$

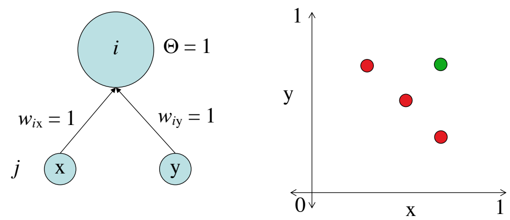


$n_i = \Sigma_j w_{ij} a_j = 1.0$ $\quad a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} = 0$

25

Computing Mappings with Layered Networks

Likewise for x = 0.75, y = 0.25: $n_i = 1.0$; $a_i = 0$

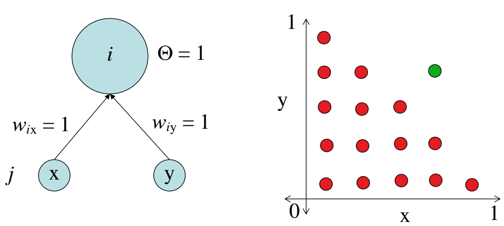


$n_i = \Sigma_j w_{ij} a_j = 1.0$ $\quad a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} = 0$

26

Computing Mappings with Layered Networks

And for all other points below them: $n_i \leq 1.0$; $a_i = 0$

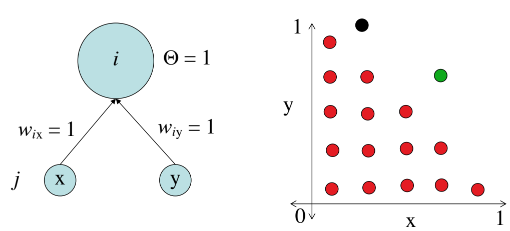


$n_i = \Sigma_j w_{ij} a_j \leq 1.0$ $\quad a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} = 0$

27

Computing Mappings with Layered Networks

What about 0.1, 1.0?

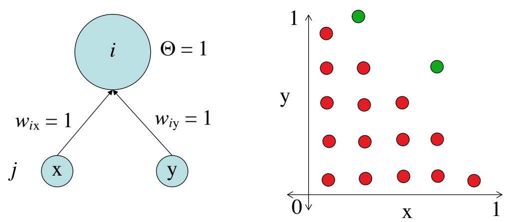


$n_i = \Sigma_j w_{ij} a_j =$ $\quad a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} =$

28

Computing Mappings with Layered Networks

What about 0.1, 1.0? $n_i = 1.1$, which is greater than Θ, so $a_i = 1$

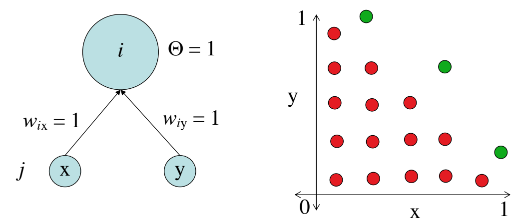


$n_i = \Sigma_j w_{ij} a_j = 1.1$ $\quad a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} = 1$

29

Computing Mappings with Layered Networks

Likewise for 1.0, 0.1: $n_i = 1.1$, so $a_i = 1$

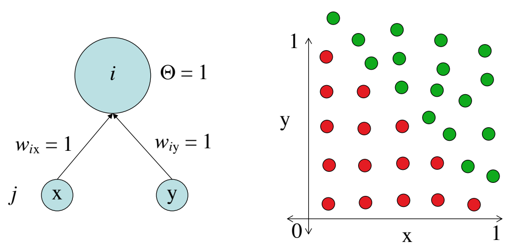


$n_i = \Sigma_j w_{ij} a_j = 1.1$ $\quad a_i = \begin{cases} 1 \text{ if } n_i > \Theta \\ 0 \text{ otherwise} \end{cases} = 1$

30

Computing Mappings with Layered Networks

And for all points above the diagonal: $n_i > 1$, so $a_i = 1$

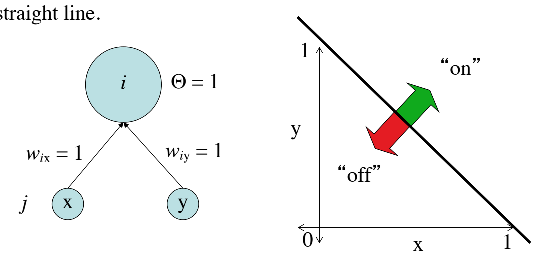


$n_i = \Sigma_j w_{ij} a_j > 1.0$ $\quad a_i = \begin{cases} 1 \text{ if } n_i \geq \Theta \\ 0 \text{ otherwise} \end{cases} = 1$

31

Computing Mappings with Layered Networks

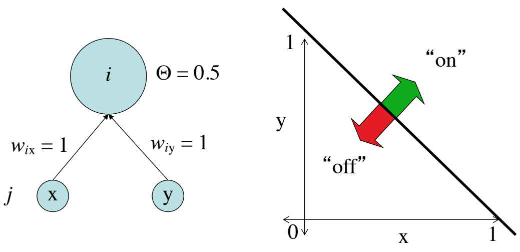
Unit *i* divides its 2-D input space into two regions with a straight line.



32

Computing Mappings with Layered Networks

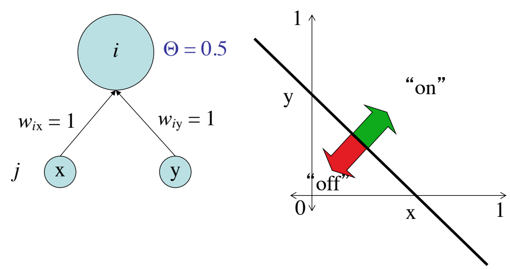
Q: What if we make $\Theta = 0.5$?



33

Computing Mappings with Layered Networks

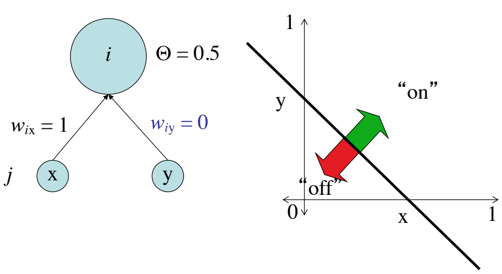
Q: What if we make $\Theta = 0.5$? A: The line moves back.



34

Computing Mappings with Layered Networks

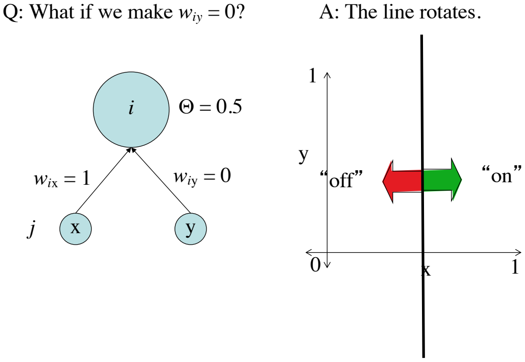
Q: What if we make $w_{iy} = 0$?



35

Computing Mappings with Layered Networks

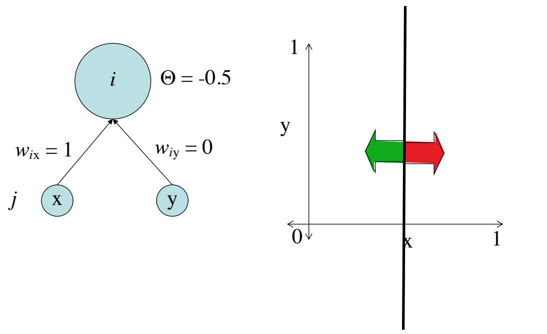
Q: What if we make $w_{iy} = 0$? A: The line rotates.



36

Computing Mappings with Layered Networks

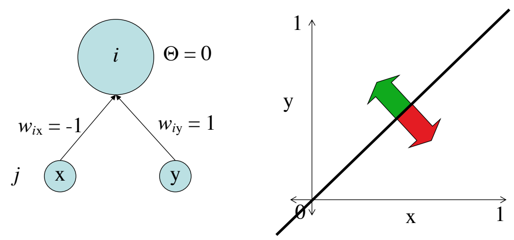
By manipulating Θ and *w*, you can make *any* line you wish…



37

Computing Mappings with Layered Networks

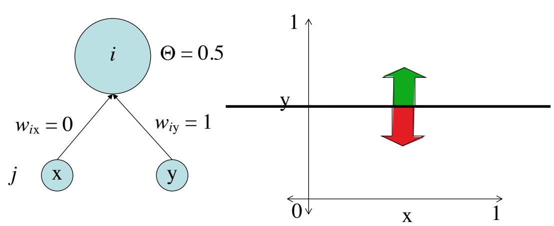
By manipulating Θ and *w*, you can make *any* line you wish…



38

Computing Mappings with Layered Networks

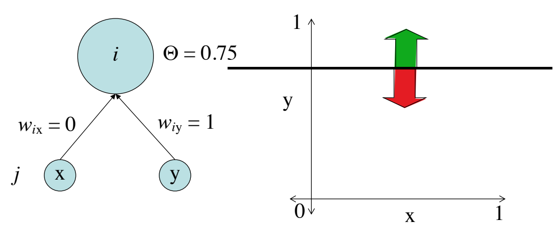
By manipulating Θ and *w*, you can make *any* line you wish…



39

Computing Mappings with Layered Networks

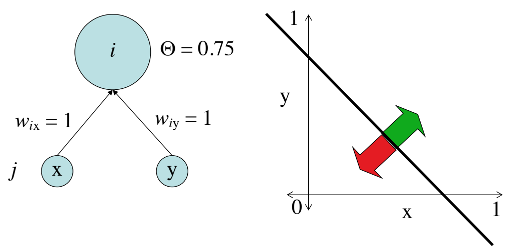
By manipulating Θ and *w*, you can make *any* line you wish…



40

Computing Mappings with Layered Networks

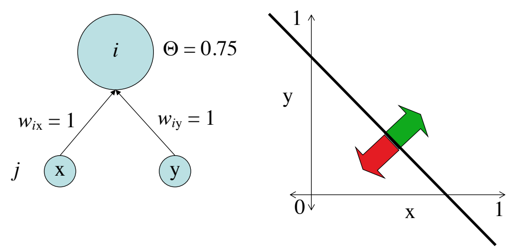
By manipulating Θ and w, you can make *any* line you wish…



41

Computing Mappings with Layered Networks

By manipulating Θ and w, you can make *any* line you wish…

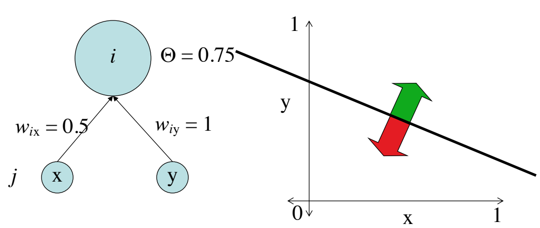


Changing w rotates the line;

42

Computing Mappings with Layered Networks

By manipulating Θ and w, you can make *any* line you wish…

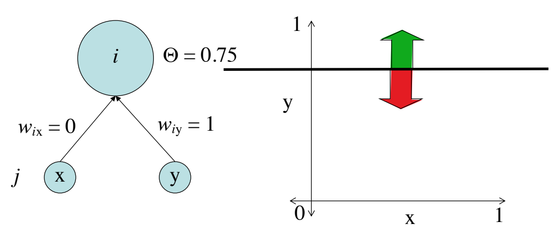


Changing w rotates the line;

43

Computing Mappings with Layered Networks

By manipulating Θ and w, you can make *any* line you wish…

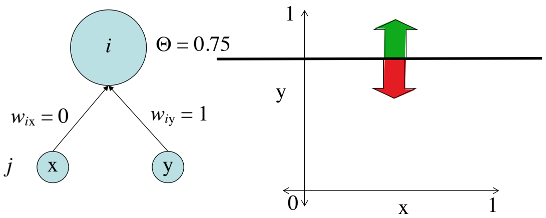


Changing w rotates the line;

44

Computing Mappings with Layered Networks

By manipulating Θ and w, you can make *any* line you wish…

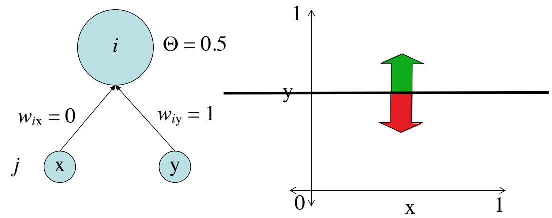


Changing w rotates the line; changing Θ slides it.

45

Computing Mappings with Layered Networks

By manipulating Θ and w, you can make *any* line you wish…

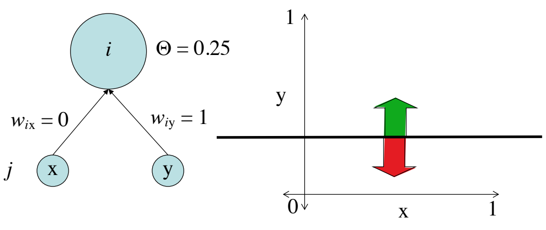


Changing w rotates the line; changing Θ slides it.

46

Computing Mappings with Layered Networks

By manipulating Θ and w, you can make *any* line you wish…

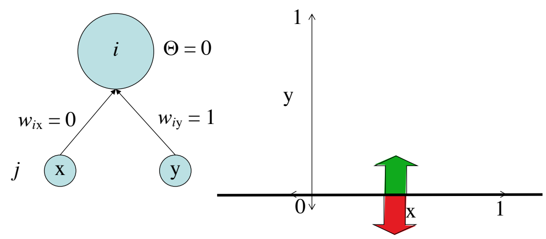


Changing w rotates the line; changing Θ slides it.

47

Computing Mappings with Layered Networks

By manipulating Θ and w, you can make *any* line you wish…

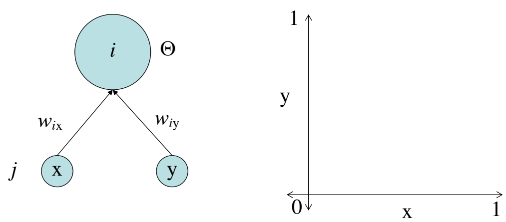


Changing w rotates the line; changing Θ slides it.

48

Computing Mappings with Layered Networks

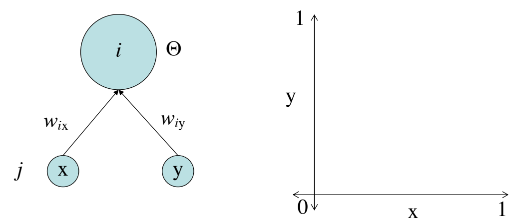
<u>The point</u>: A single BTU with a 2-D input space can divide that space into two parts with *any* straight line you wish...



49

Computing Mappings with Layered Networks

<u>The point</u>: A single BTU with a 2-D input space can divide that space into two parts with *any* straight line you wish...

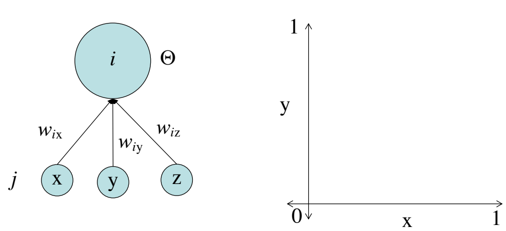


…but only with *straight* lines (more on this to come).

50

Computing Mappings with Layered Networks

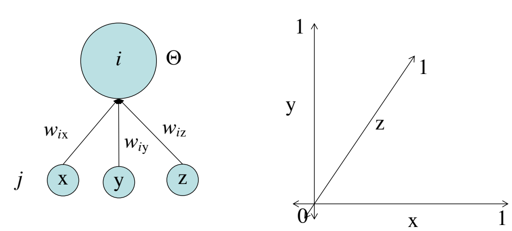
What about a BTU with **three** inputs?



51

Computing Mappings with Layered Networks

What about a BTU with **three** inputs?

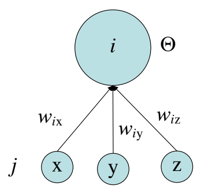


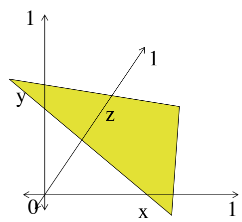
Its input space is 3-D…

52

Computing Mappings with Layered Networks

What about a BTU with **three** inputs?



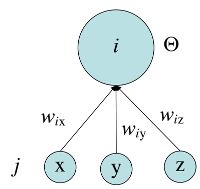


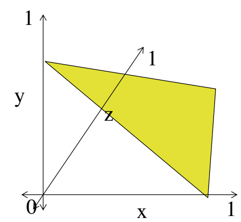
Its input space is 3-D, and it divides that space into two parts with 2-D planes.

53

Computing Mappings with Layered Networks

What about a BTU with **three** inputs?



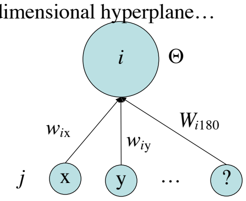


Its input space is 3-D, and it divides that space into two parts with 2-D planes. *Any* plane you want, as long as its straight.

54

Computing Mappings with Layered Networks

In general: A BTU with *N* inputs divides its input space into two parts (an “on” part and an “off” part) with an *N*-1 dimensional hyperplane…

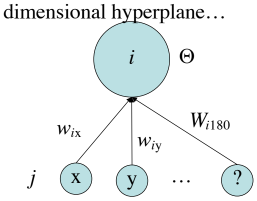


(180-D picture not available)

55

Computing Mappings with Layered Networks

In general: A BTU with *N* inputs divides its input space into two parts (an “on” part and an “off” part) with an *N*-1 dimensional hyperplane…



…*any* hyperplane you want, *as long as it’s straight*.

56

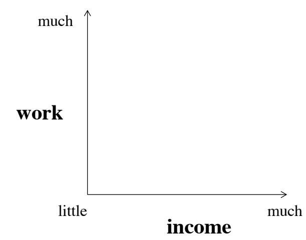
Linear Separability

Two categories are *linearly separable* iff the members of one can be separated from the members of the other by a straight hyperplane.

Iff two categories are linearly separable, then they can be distinguished by a BTU.

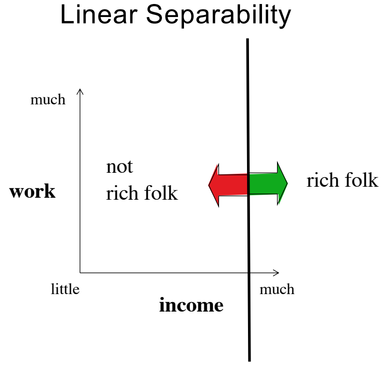
57

Linear Separability



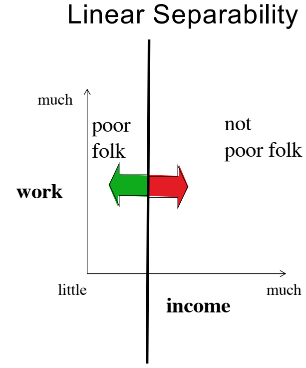
58

Linear Separability



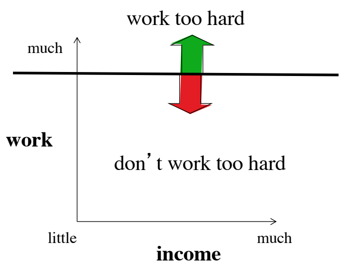
59

Linear Separability



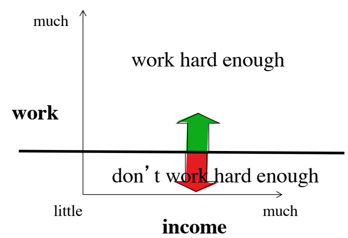
60

Linear Separability



61

Linear Separability



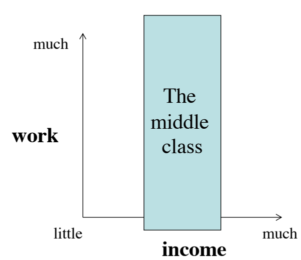
62

But What About the Middle Class?

And What About Those Who Work Just the Right Amount?

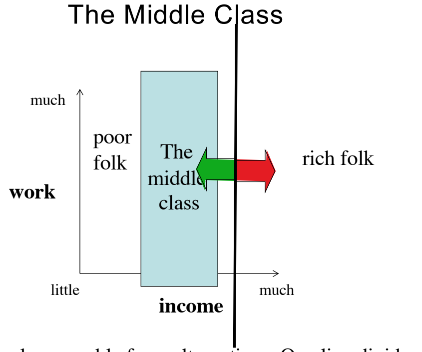
63

The Middle Class



64

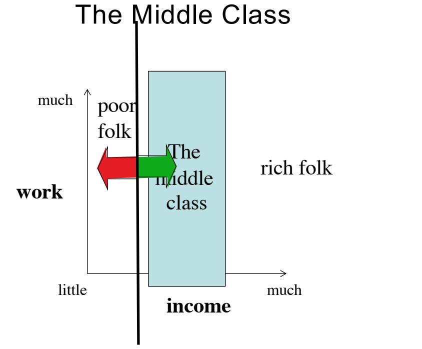
The Middle Class



Not linearly separable from alternatives: One line divides the middle class and the poor from the rich folk…

65

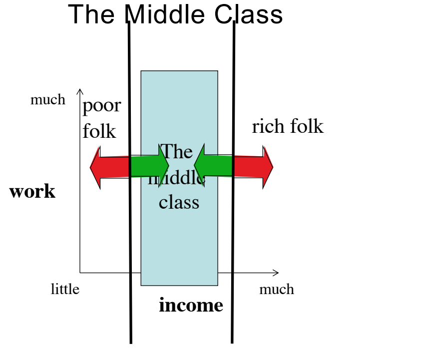
The Middle Class



…and another line divides the middle class and the rich from the poor folk.

66

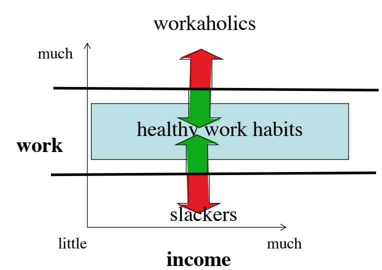
The Middle Class



The middle class needs *two* lines to classify it: Not separable from others by one line: *not linearly separable*.

67

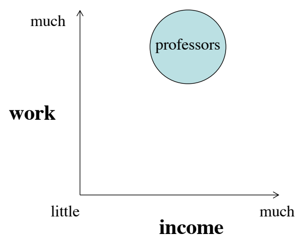
Those Who Work the Right Amount



Likewise, "Those who work the right amount" needs *two* lines to classify it: *not linearly separable*.

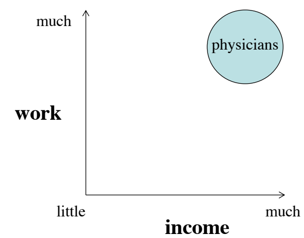
68

Other Non-linearly Separable Categories



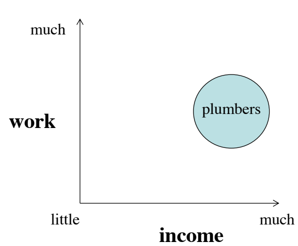
69

Other Non-linearly Separable Categories



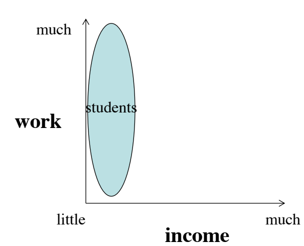
70

Other Non-linearly Separable Categories



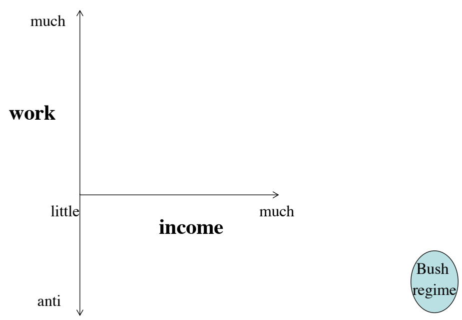
71

Other Non-linearly Separable Categories



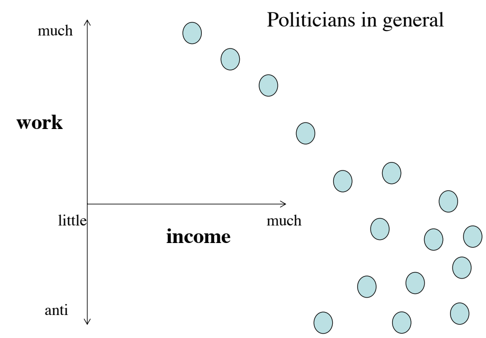
72

Other Non-linearly Separable Categories



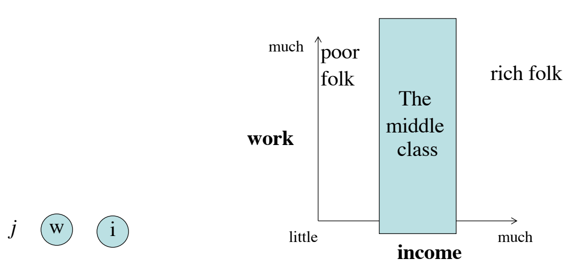
73

Other Non-linearly Separable Categories



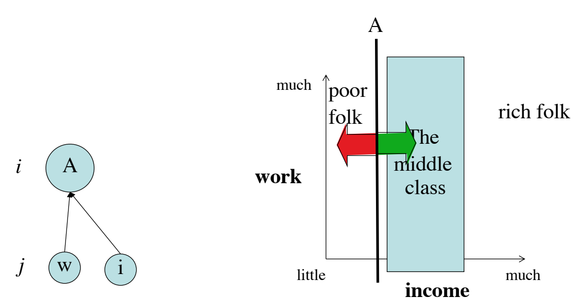
74

Dealing with Non-linearly Separable Categories



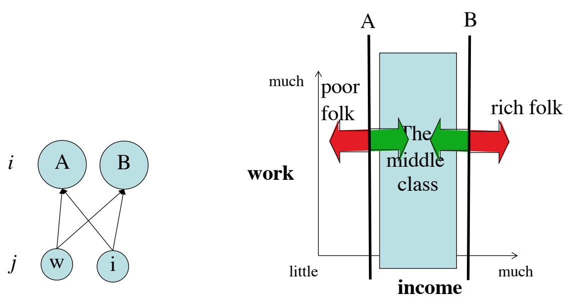
75

Dealing with Non-linearly Separable Categories



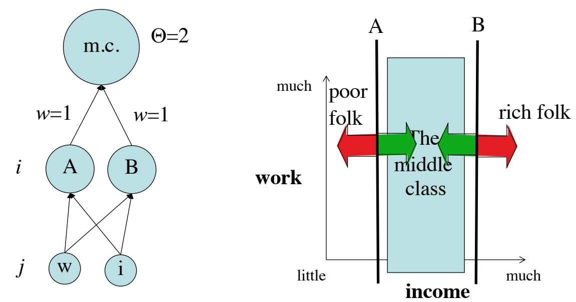
76

Dealing with Non-linearly Separable Categories



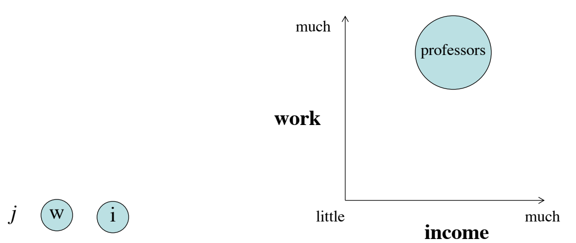
77

Dealing with Non-linearly Separable Categories



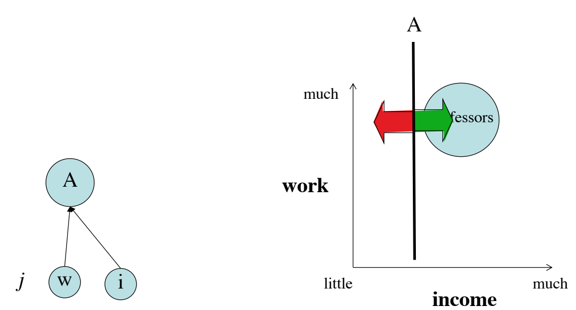
78

Dealing with Non-linearly Separable Categories



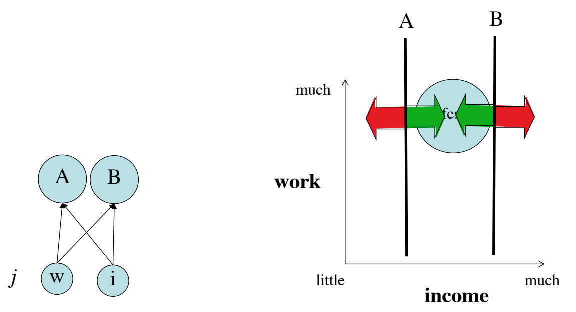
79

Dealing with Non-linearly Separable Categories



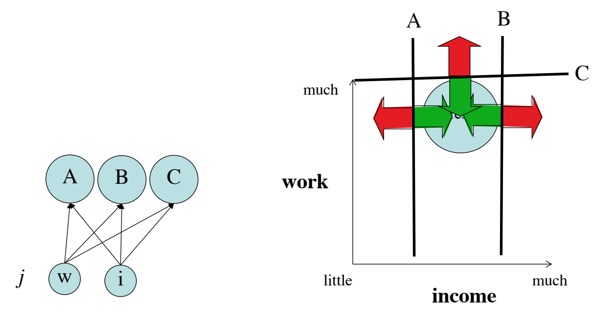
80

Dealing with Non-linearly Separable Categories



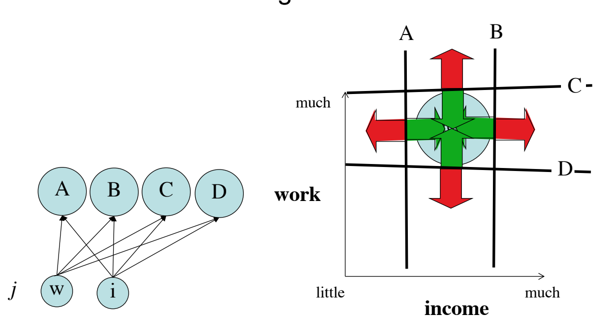
81

Dealing with Non-linearly Separable Categories



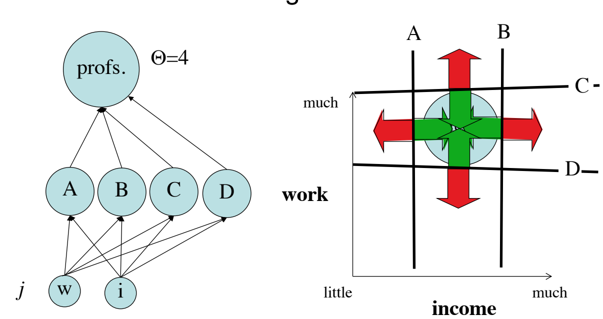
82

Dealing with Non-linearly Separable Categories



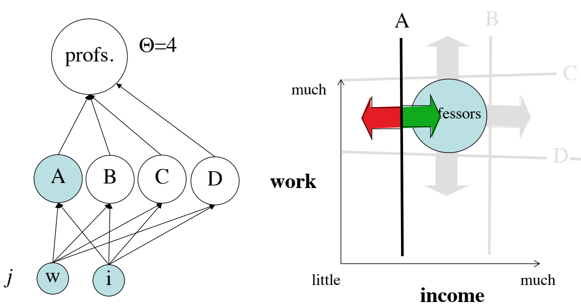
83

Dealing with Non-linearly Separable Categories



84

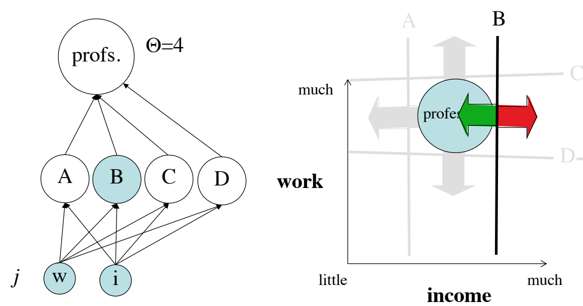
In General…



Each unit in the first layer divides the input space with a straight hyperplane:
Any straight hyperplane.

85

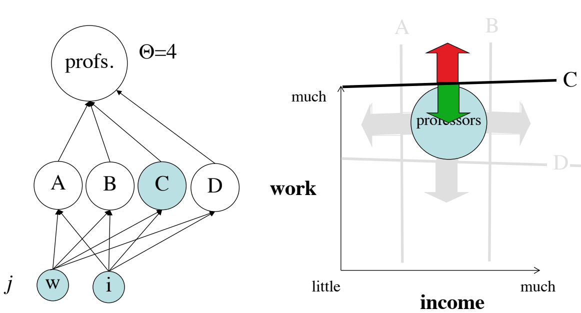
In General…



Each unit in the first layer divides the input space with a straight hyperplane:
Any straight hyperplane.

86

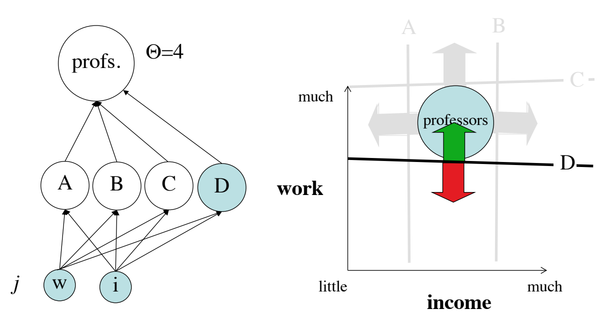
In General…



Each unit in the first layer divides the input space with a straight hyperplane:
Any straight hyperplane.

87

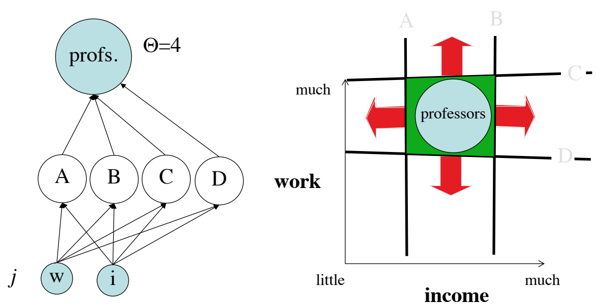
In General…



Each unit in the first layer divides the input space with a straight hyperplane:
Any straight hyperplane.

88

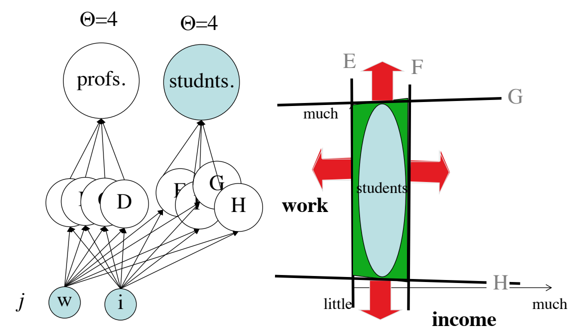
In General…



Each unit in the second layer can ***and*** those hyperplanes together to divide the input space into a convex region.

89

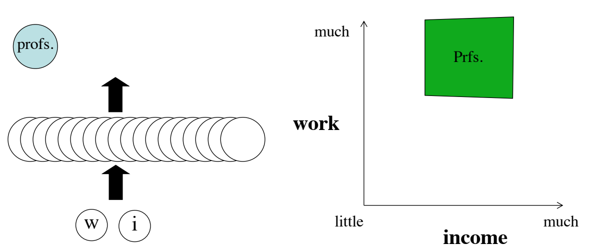
In General…



Each unit in the second layer can ***and*** those hyperplanes together to divide the input space into a convex region: ***Any*** convex region.

90

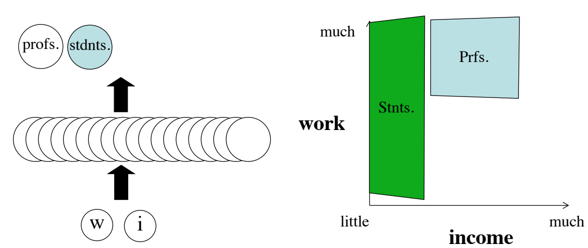
More examples



Each unit in the second layer can ***and*** those hyperplanes together to divide the input space into a convex region: ***Any*** convex region.

91

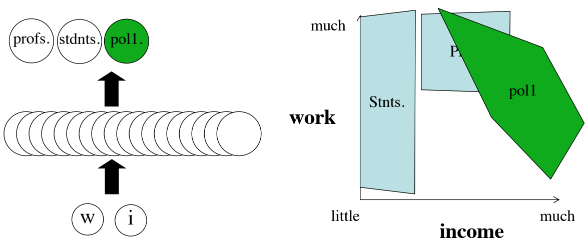
More examples



Each unit in the second layer can ***and*** those hyperplanes together to divide the input space into a convex region: ***Any*** convex region.

92

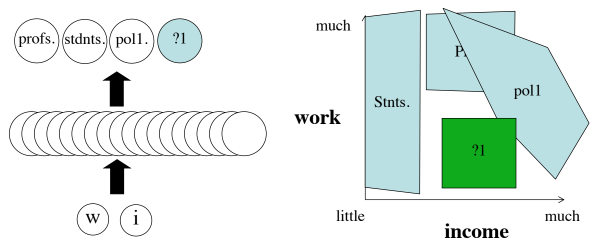
More examples



Each unit in the second layer can ***and*** those hyperplanes together to divide the input space into a convex region: ***Any*** convex region.

93

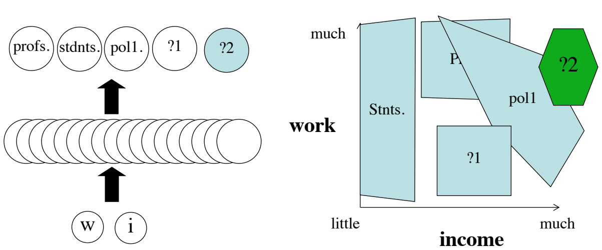
More examples



Each unit in the second layer can ***and*** those hyperplanes together to divide the input space into a convex region: ***Any*** convex region.

94

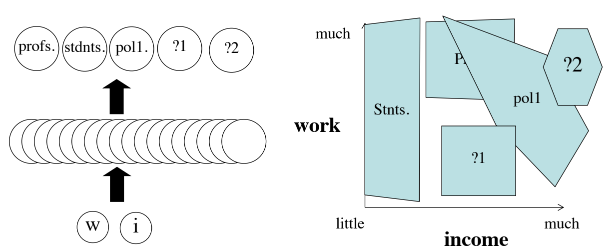
More examples



Each unit in the second layer can ***and*** those hyperplanes together to divide the input space into a convex region: ***Any*** convex region.

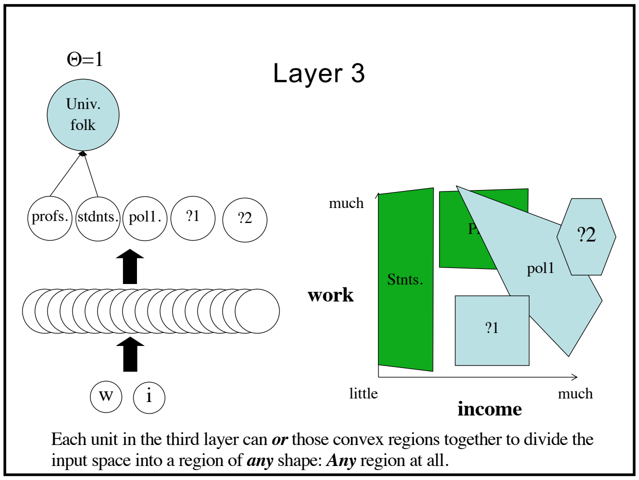
95

Layer 3

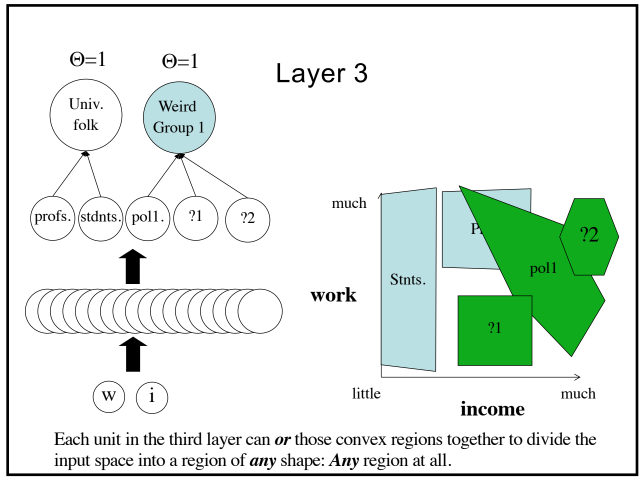


Each unit in the third layer can ***or*** those convex regions together to divide the input space into a region of ***any*** shape: ***Any*** region at all.

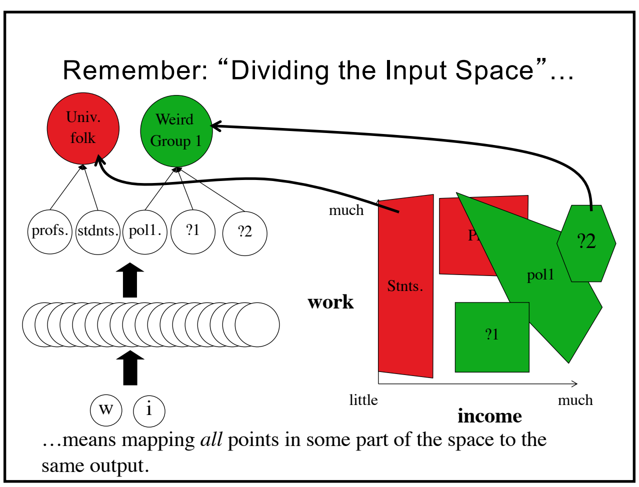
96


Layer 3
Θ=1
Univ. folk
profs.
stdnts.
pol1.
?1
?2
w
i
much
work
little
income
much
Stnts.
pol1
?1
?2
Each unit in the third layer can *or* those convex regions together to divide the input space into a region of *any* shape: *Any* region at all.

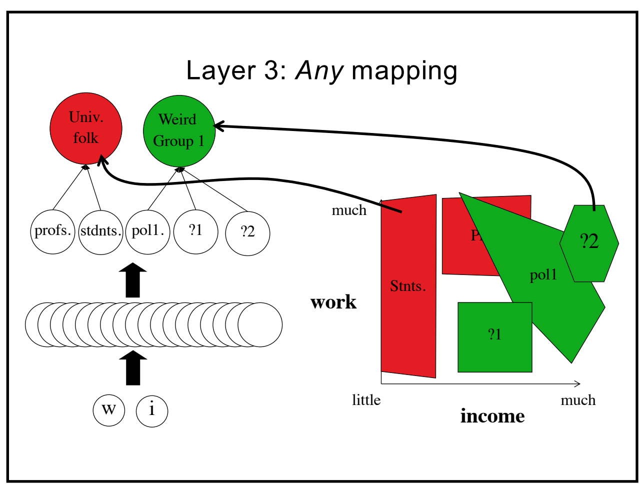
97


Layer 3
Θ=1
Θ=1
Univ. folk
Weird Group 1
profs.
stdnts.
pol1.
?1
?2
w
i
much
work
little
income
much
Stnts.
pol1
?1
?2
Each unit in the third layer can *or* those convex regions together to divide the input space into a region of *any* shape: *Any* region at all.

98

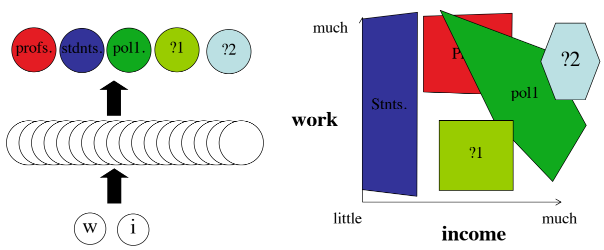

Remember: "Dividing the Input Space"…
Univ. folk
Weird Group 1
profs.
stdnts.
pol1.
?1
?2
w
i
much
work
little
income
much
Stnts.
pol1
?1
?2
…means mapping *all* points in some part of the space to the same output.

99


Layer 3: *Any* mapping
Univ. folk
Weird Group 1
profs.
stdnts.
pol1.
?1
?2
w
i
much
work
little
income
much
Stnts.
pol1
?1
?2

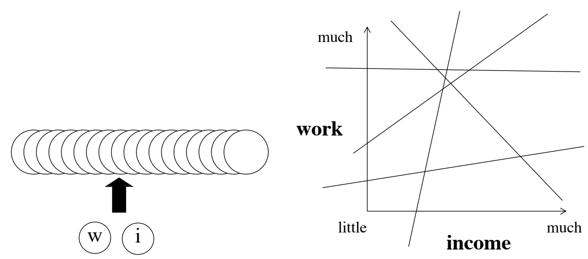
100

Layer 2: Any *convex* mapping



101

Layer 1: Any *straight* hyperplane



102

Summary and Implications

Summary: A three-layer[1] non-linear[2] perceptron[3] can compute *any* computable mapping from its inputs to its outputs.

[1]Three layers of connections above the input.
[2]Non-linear activation function (e.g., BTU).
[3]Layered network.

Implication: Does this mean it can compute symbolic functions?

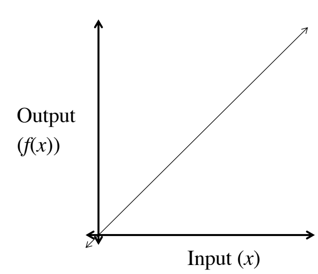
Answer: No. It must be trained on (or wired to compute) each mapping *individually*: It can compute any mapping (in principle), but you have to tell it how to compute each one.

A function, by contrast, is simultaneously applicable to *all* possible mappings in its domain.

103

Functions vs. Perceptrons

A function (the identity function): $f(x) = x$

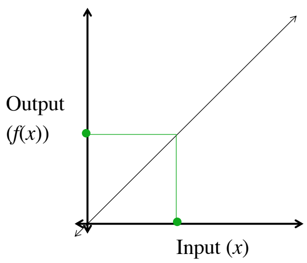


The output of a function is defined for *all* inputs in its domain.

104

Functions vs. Perceptrons

A function (the identity function): $f(x) = x$

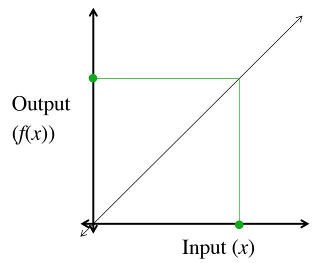


The output of a function is defined for *all* inputs in its domain.

105

Functions vs. Perceptrons

A function (the identity function): $f(x) = x$

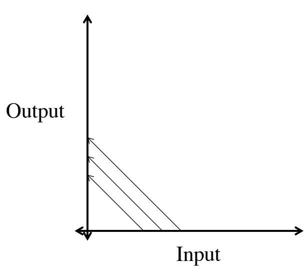


The output of a function is defined for *all* inputs in its domain.

106

Functions vs. Perceptrons

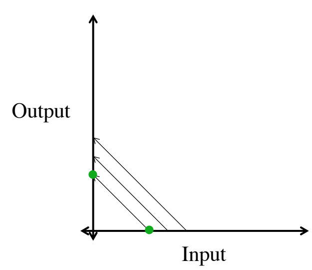
A perceptron learns to compute *specific* mappings



107

Functions vs. Perceptrons

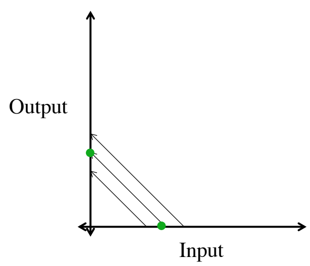
A perceptron learns to compute *specific* mappings



108

Functions vs. Perceptrons

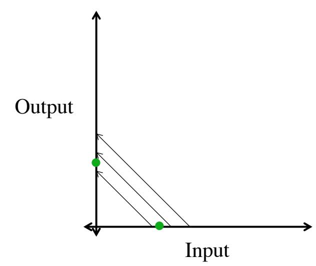
A perceptron learns to compute *specific* mappings



109

Functions vs. Perceptrons

A perceptron learns to compute *specific* mappings

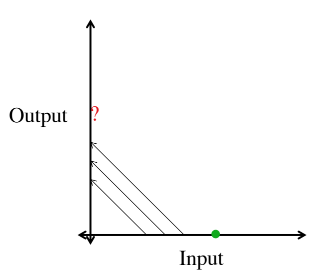


Interpolation (sometimes)

110

Functions vs. Perceptrons

A perceptron learns to compute *specific* mappings



But no extrapolation, even for inputs in the trained domain.

111