MA666: Neural Networks and Learning

Part 1
A Discrete Neuron: The Perceptron

Today

We'll begin to study neural networks:

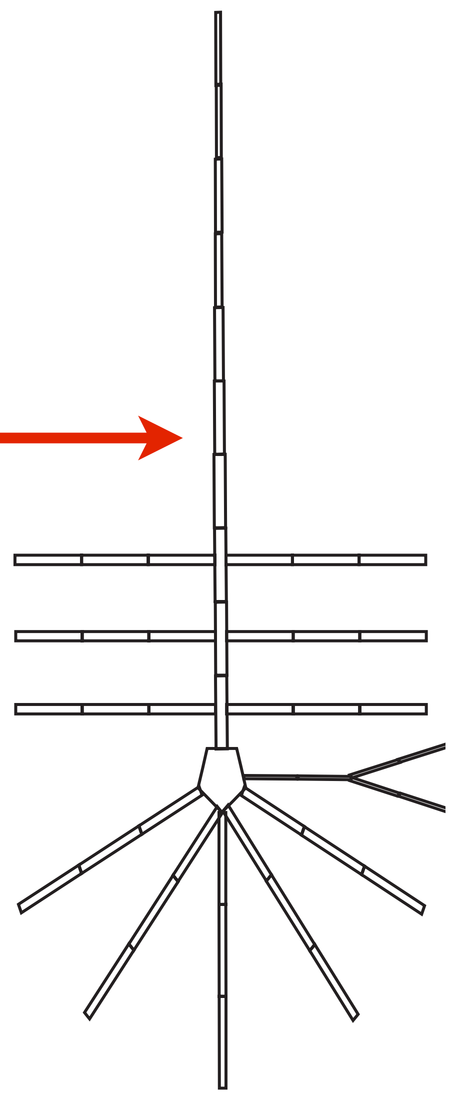
– The simplest case: The Perceptron.

Neural models

. . . can be extremely complicated:

multi-compartment models

each at least a HH model



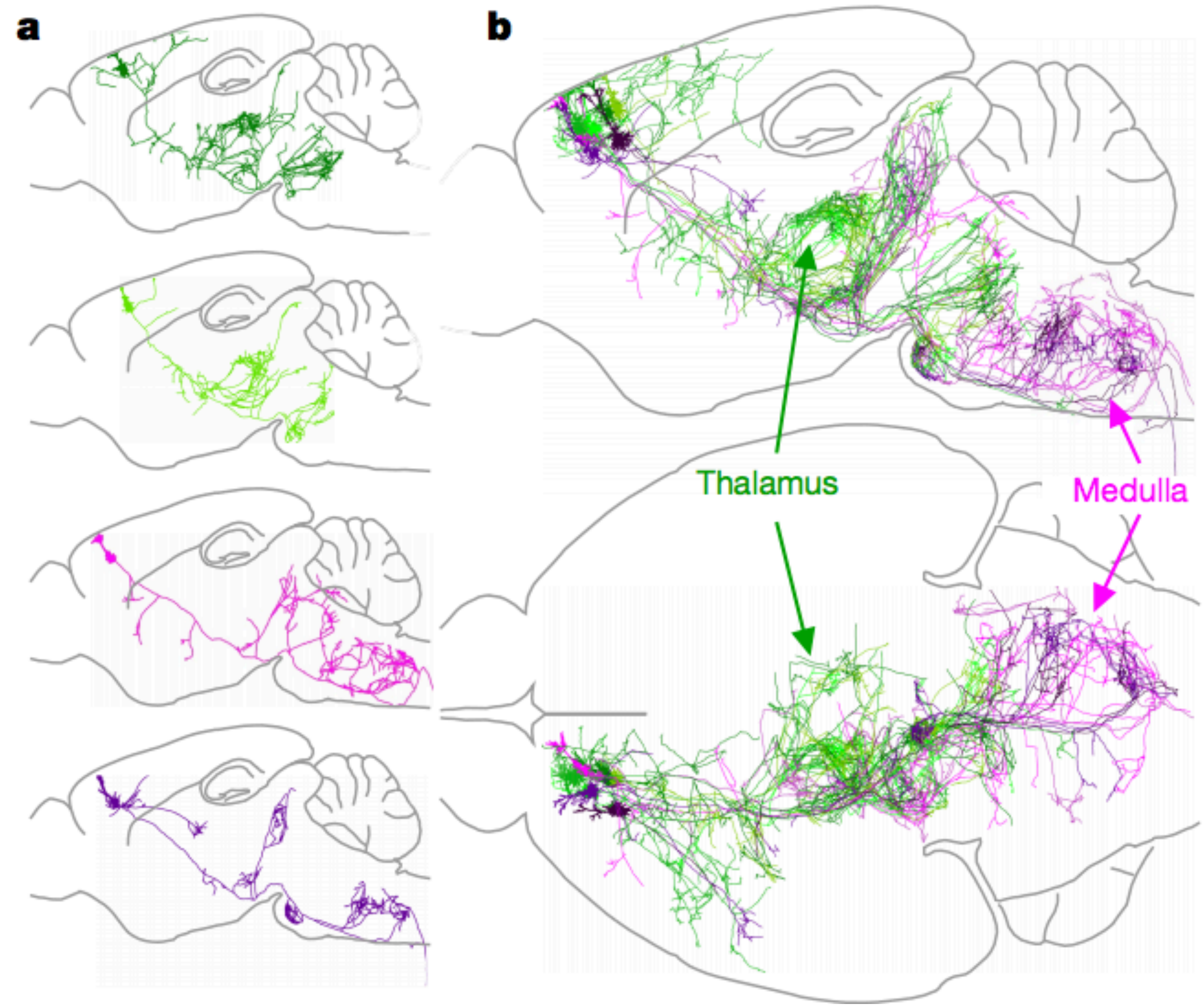
Blue brain project

A neuron, conceptually

Conceptually, a neuron:

- receives inputs
- processes those inputs
- generates an output.

In practice, it's really complicated …



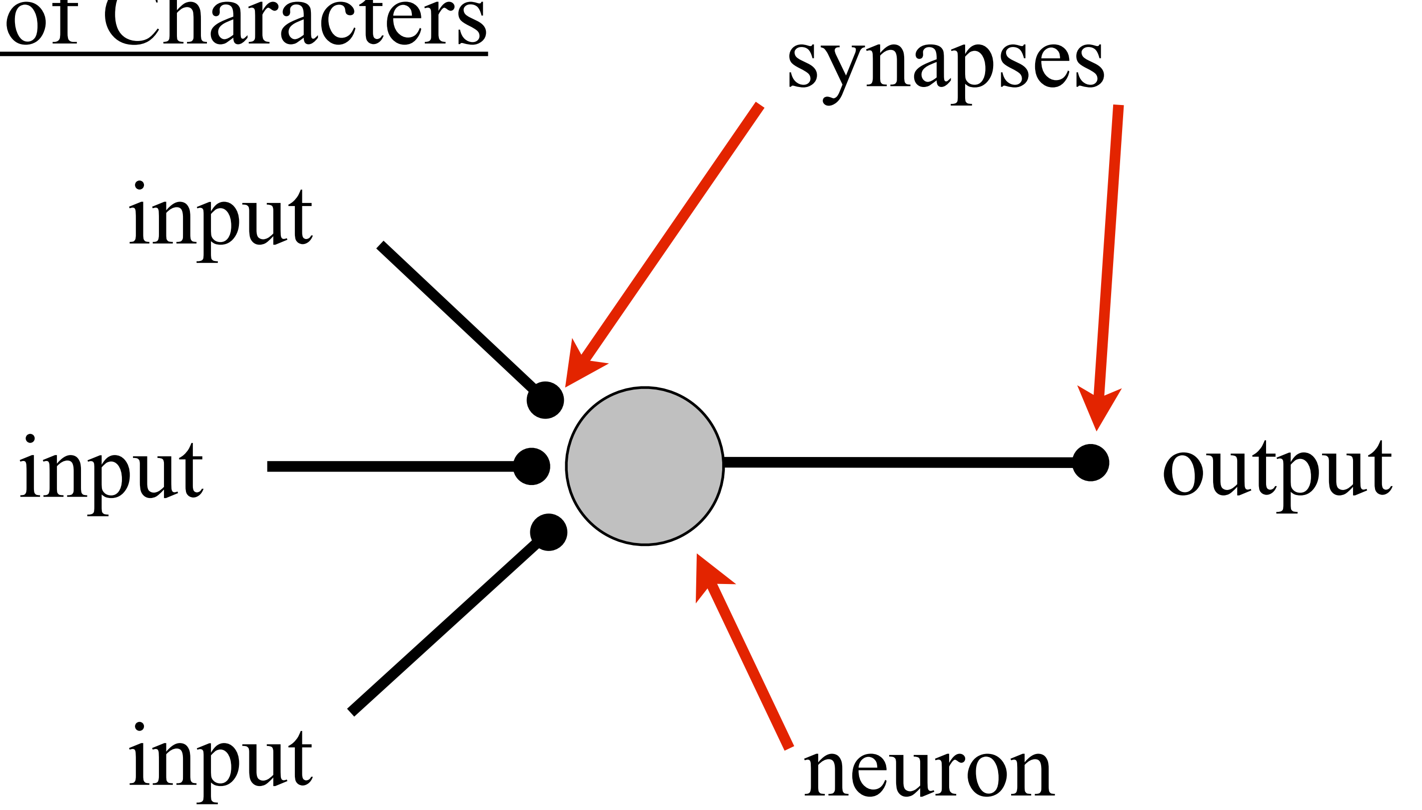
[Economo et al, Nature, 2018]

Neural network models

Here, we'll simplify.

Consider **neural networks**: collections of abstracted neurons connected to each other through weighted connections (simplified "synapses").

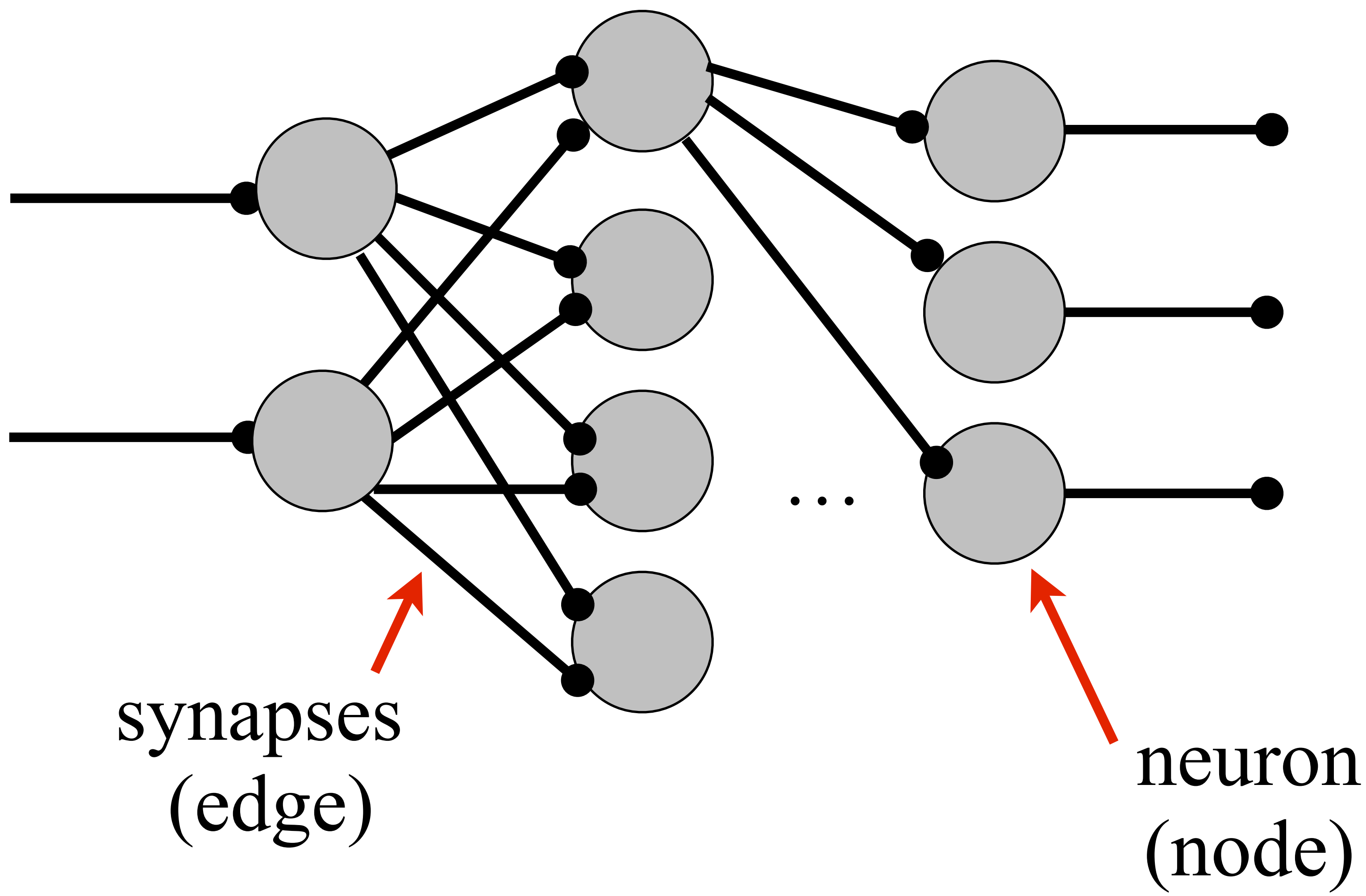
Cartoon & Cast of Characters



Q: What's been lost here?

Neural network models

Neural networks can be more complex …



Networks can adapt their behavior by adjusting edge weights.

We'll talk more about this if there's time next week …

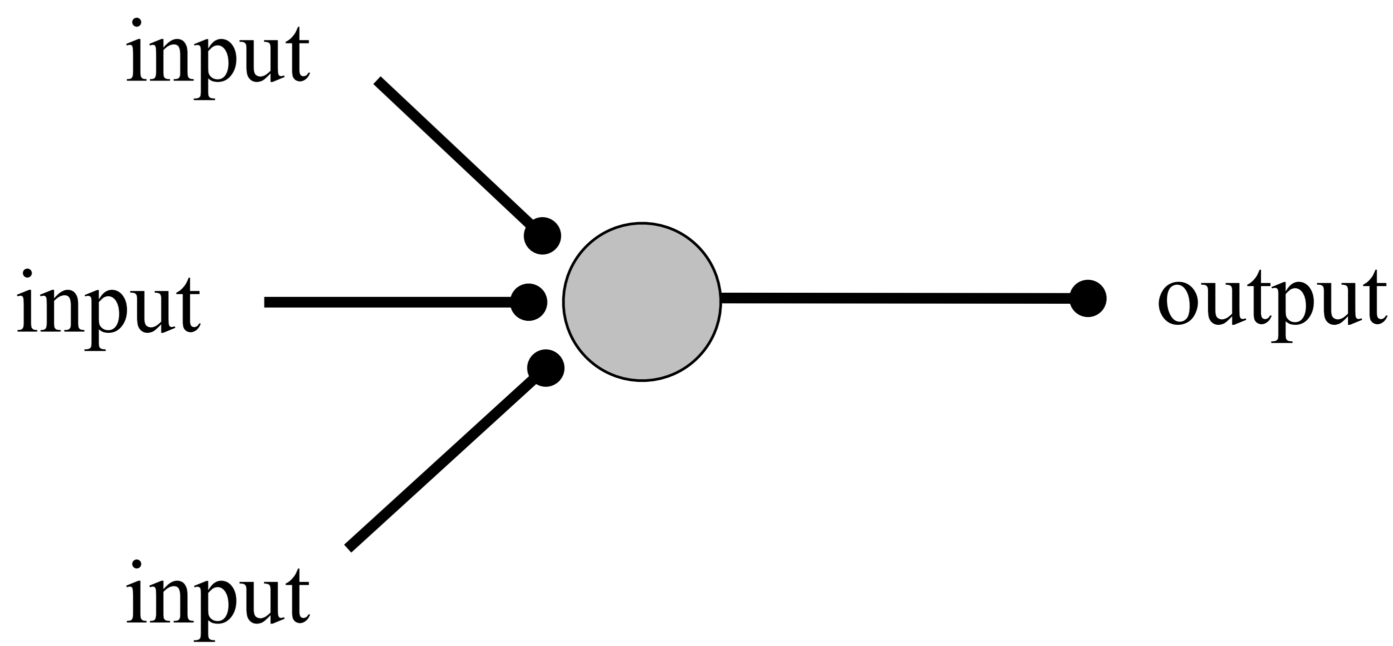
The "simplest" information processor

The Perceptron

– the simplest neural network possible: a single neuron

<u>Three elements</u>:

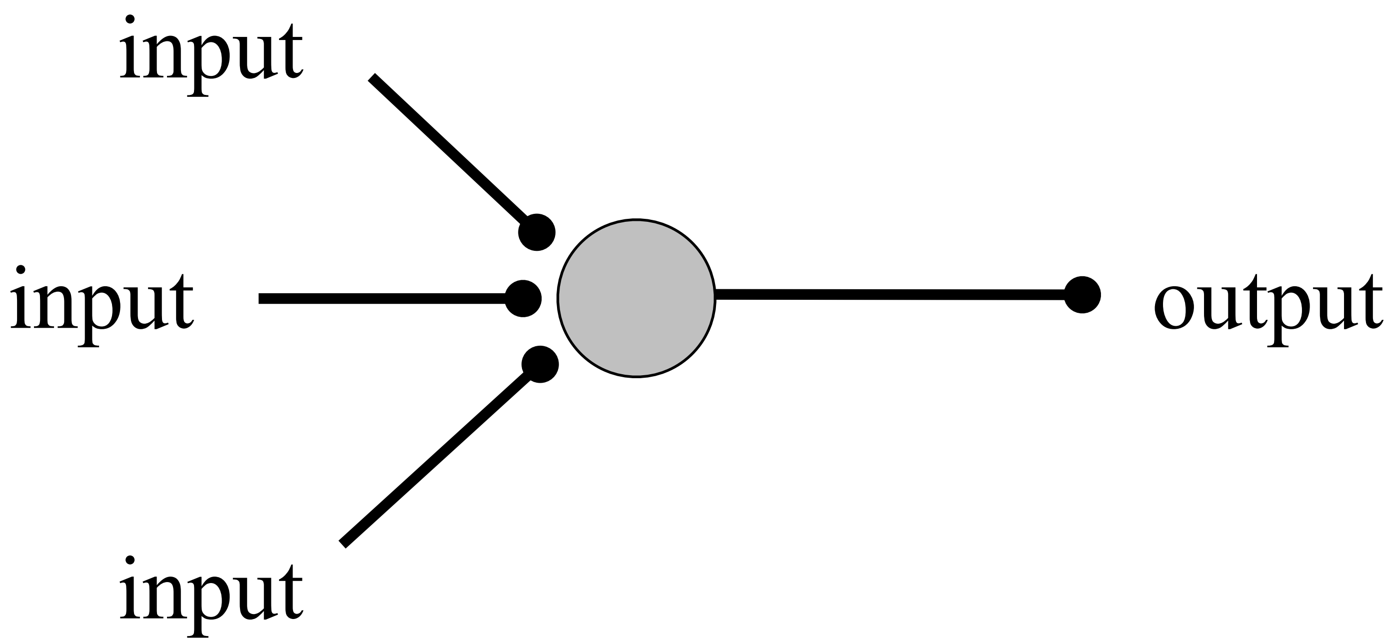
1. input(s) 2. "processor" 3. single output



Feed-forward model progresses from left to right

input comes in, gets processed, output goes out

The "simplest" information processor



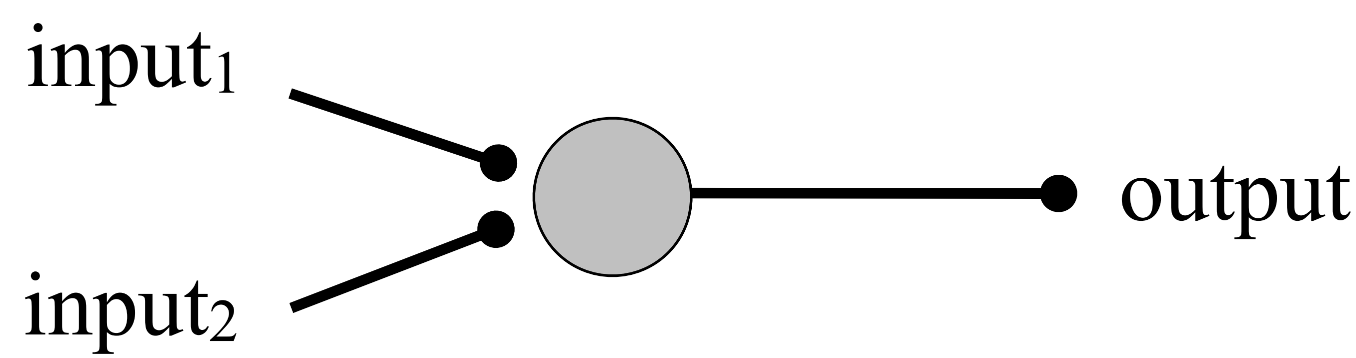
Divide information processing into 4 steps:

1. Receive inputs
2. Weight inputs
3. Sum weighted inputs
4. Generate output

Let's go through each step, in a concrete example …

4 steps of information processing (Step 1)

Step 1. Receive inputs.



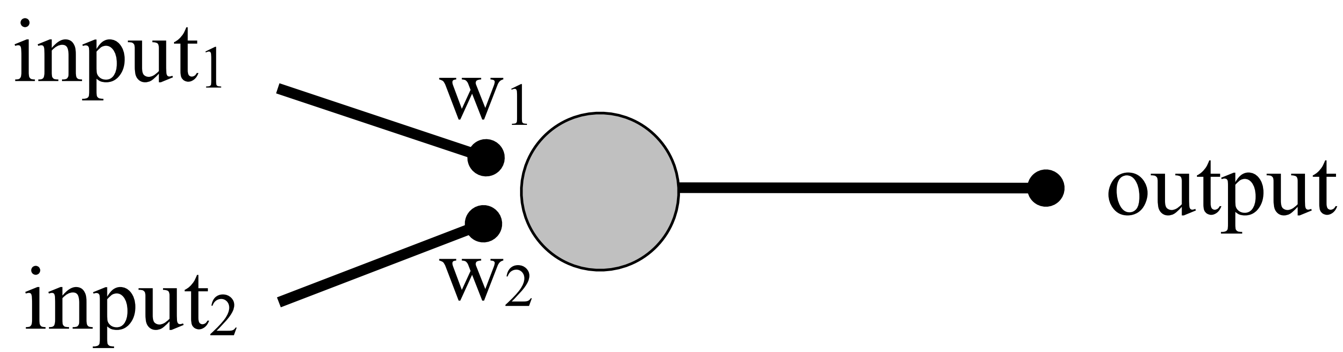
Example: a perceptron with two inputs.

Let's define: $input_1 = 12$

$input_2 = 4$

4 steps of information processing (Step 2)

Step 2. Weight inputs.



Each input sent to the neuron is **weighted**

= multiplied by some number.

Example: Let's define: $w_1 = 0.5$

$w_2 = -1$

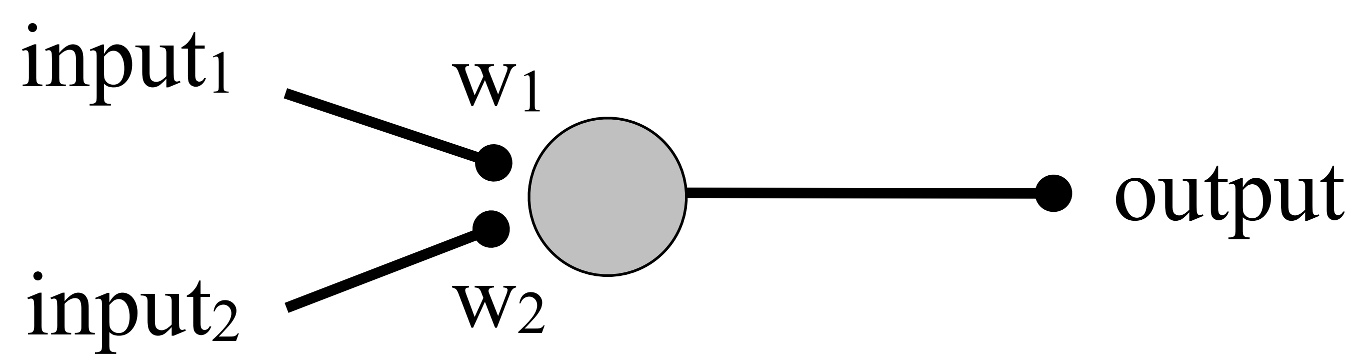
Now, "weight inputs": multiply each input by its weight.

$$input_1 * w_1 = 12 * 0.5 = 6$$

$$input_2 * w_2 = 4 * -1 = -4$$

4 steps of information processing (Step 3 & 4)

Step 3. Sum weighted inputs



$$input_1 * w_1 + input_2 * w_2 \quad = 6 + (-4) \quad = 2$$

Step 4. Generate output.

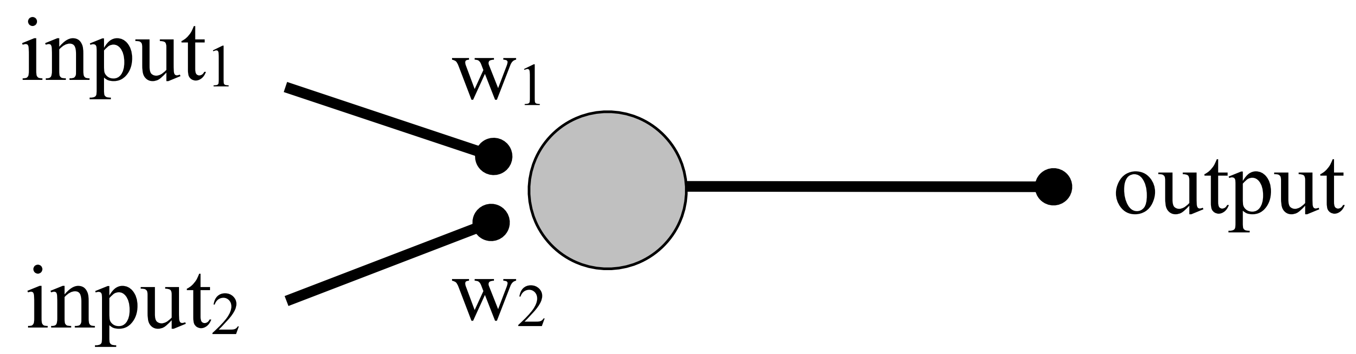
Q: How?
A: Pass the summed weighted inputs through an **activation function**

If the summed weighted input is "big enough", then "fire".

Different choices here … we'll consider different options.

The Perceptron Algorithm

Summary:



1. For every input, multiply that input by its weight.

2. Sum all of the weighted inputs

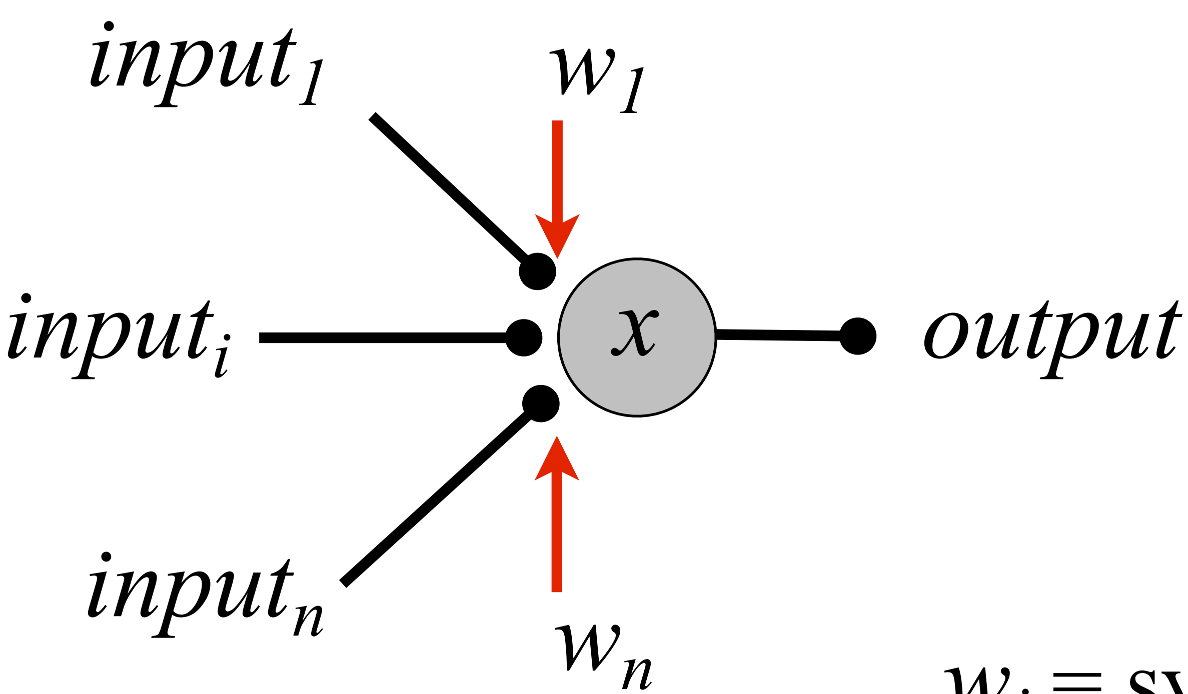
3. Compute the output of the perceptron based on that sum passed through an activation function.

(we'll discuss these later)

The "simplest" information processor: more generally

Summary: the neuron performs a **weighted addition** of its input. The sum is then run through an **activation function** to produce output which can then act as input to other neurons.

To start, let's assign variable names to each model element:



x = activity of neuron

$input_i$ = input from source i

w_i = synaptic weight from $input_i$ to neuron

The perceptron: more generally

The activity of the neuron depends on the summed, weighted inputs.

In the simplest case:

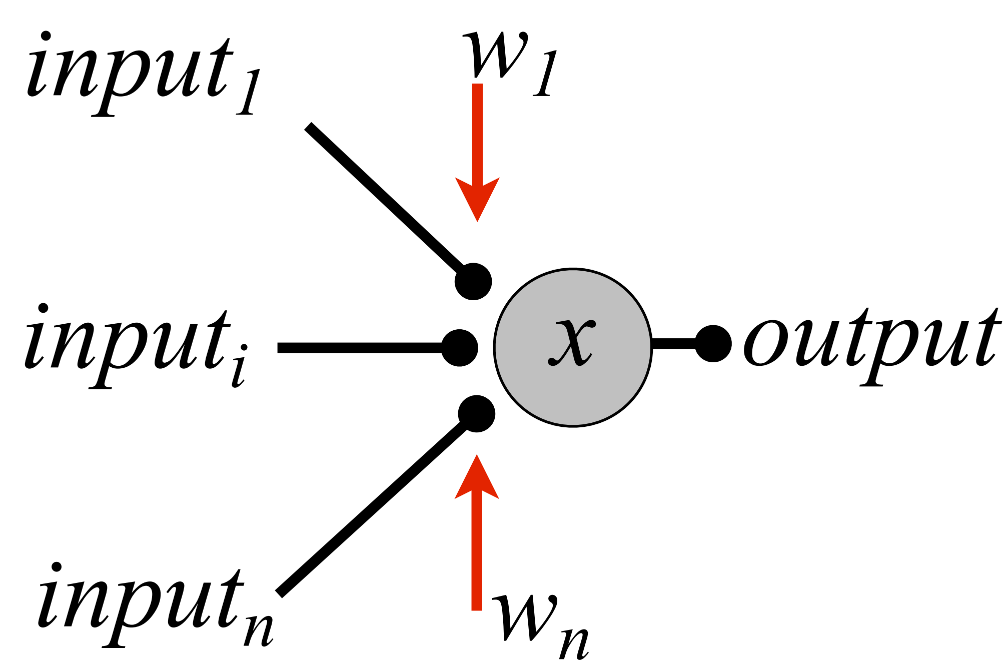
activity of neuron

$$x = \sum_i input_i \, w_i$$

“summation” over all inputs

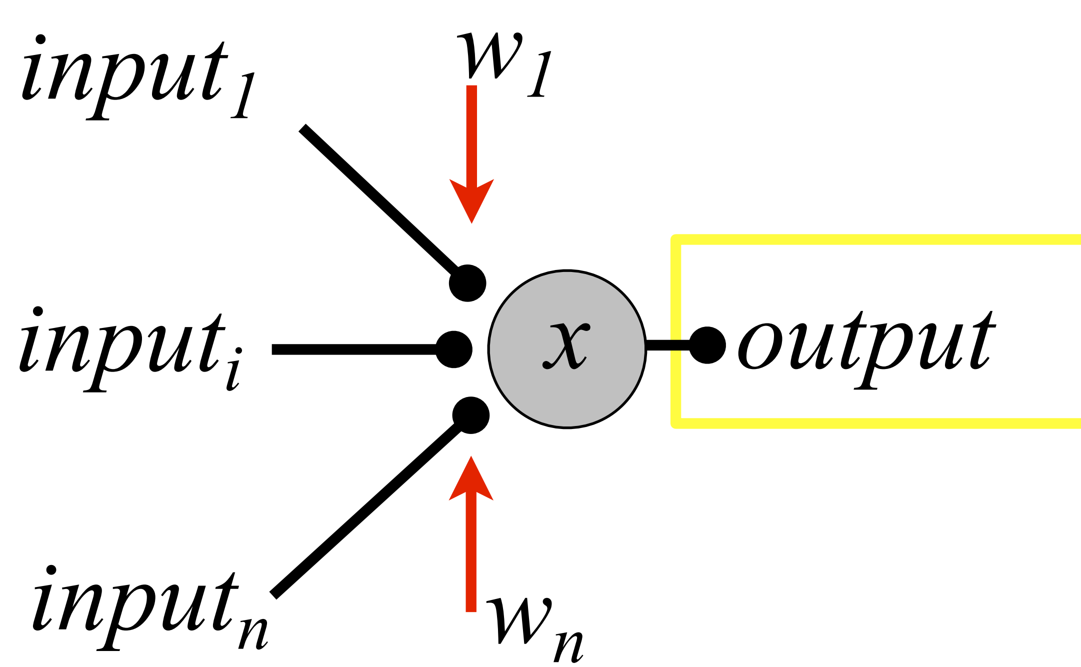
input from source i

weight from i



The perceptron: more generally

The **output** of the neuron is a function of the activity of the neuron (x):



In general,

$$\boldsymbol{output = f(x)}$$

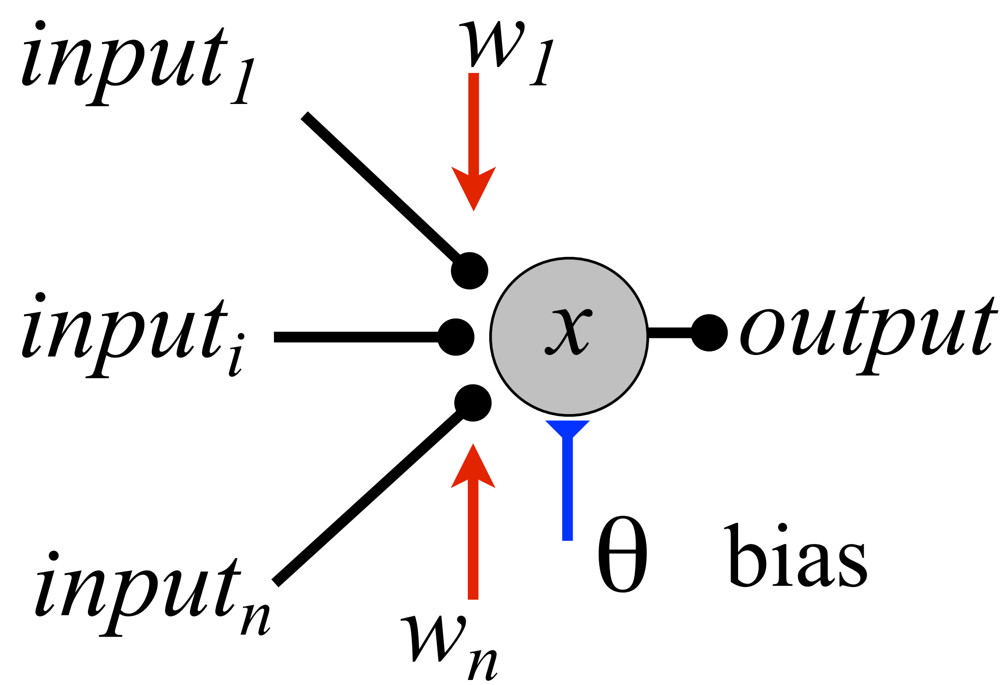
Here:

$$output = 0 \text{ for x} \leq 0$$
$$output = 1 \text{ for x} > 0$$

The activation function is **binary** (0 or 1).

Bias term

We can modify the model by adding a **bias** term:



Now the activity for the neuron becomes:

$$x = \sum_i input_i \, w_i \boxed{+\,\theta} \quad \text{new bias term}$$

Bias term

Q: What is the effect of a negative bias term θ ?

$$x = \underbrace{\boxed{\sum_i input_i \, w_i}}_{\text{Total input}} + \underbrace{\boxed{\theta}}_{\text{bias}}$$

For the neuron to generate output: x > 0 (Then *output* = 1)

To compensate for the negative bias term θ, the total input must increase to push the x above zero.

In other words: we need more input to make the neuron produce output.

The perceptron with bias term

So, the neuron model with bias:

$$x = \sum_i input_i\, w_i \underbrace{\boxed{+\,\theta}}_{\text{bias}}$$

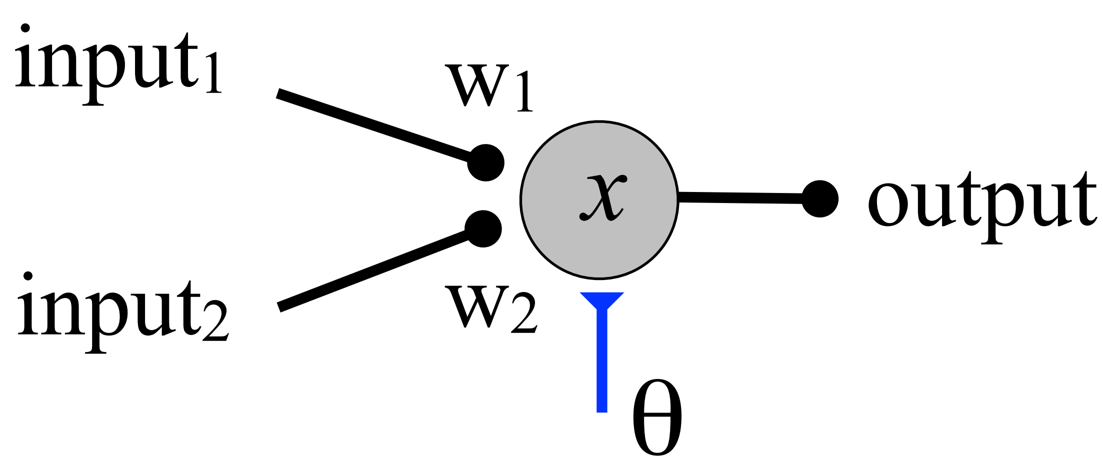
and

binary activation function

$output = 0$ for x $\leq$ 0

$output = 1$ for x > 0

Ex.



Inputs to the neuron:

$input_1 = 1 \quad input_2 = 0$

Synaptic weights:

$w_1 = 0.5 \quad w_2 = -0.5$

Bias: $\theta = -1$

Q: What is *output* ?

$$x = input_1\, w_1 + input_2\, w_2 + \theta_j$$

$$x = 1{*}0.5 + 0{*}(-0.5) - 1 = -0.5$$

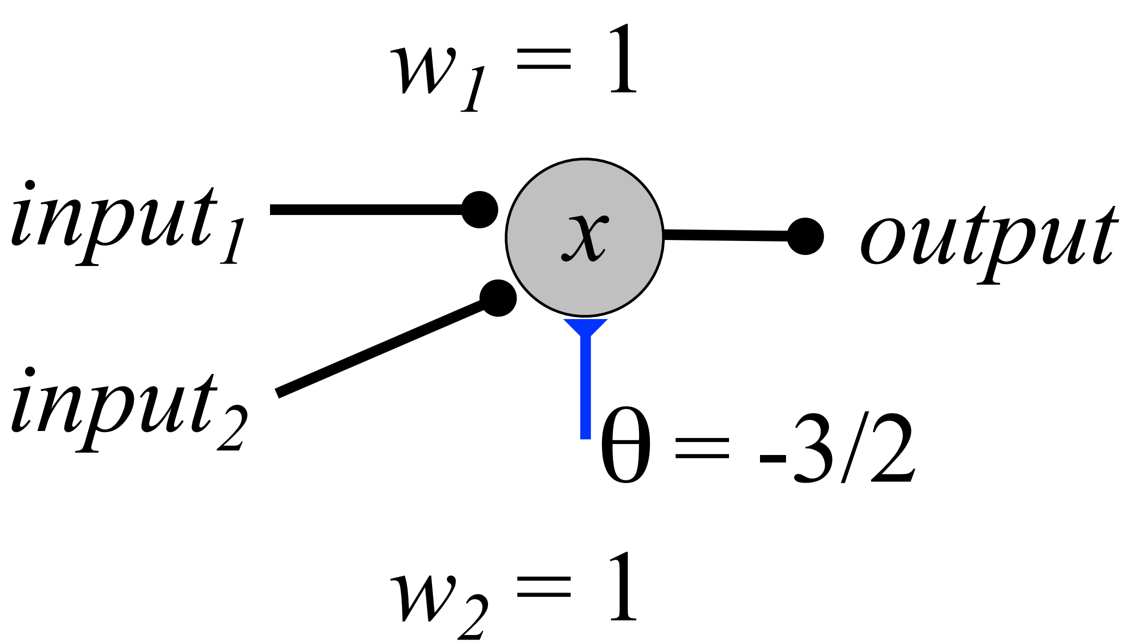
$x < 0$ so $\boxed{output = 0}$

The perceptron: application

The neuron model can perform logical operations:

Q: What logical operations can we perform?

Consider:



Note: *output* = 1 if
both $input_1$ and $input_2$ provided.

A: ?

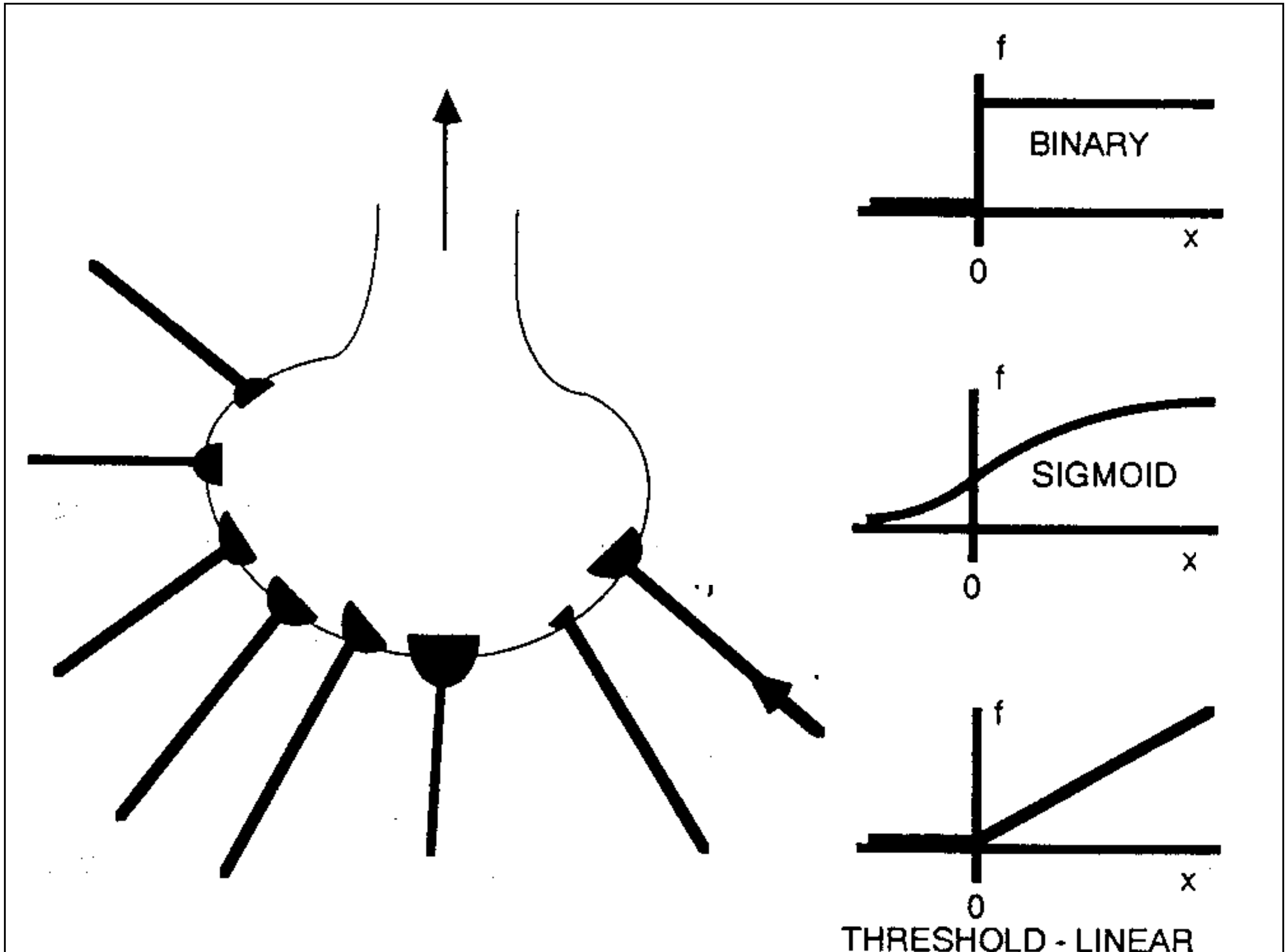
Make a table:

Input		Output
$input_1$	$input_2$	$output$
0	0	
1	0	
0	1	
1	1	

More complicated neural models

Single neuron models can become <u>more complicated</u>:

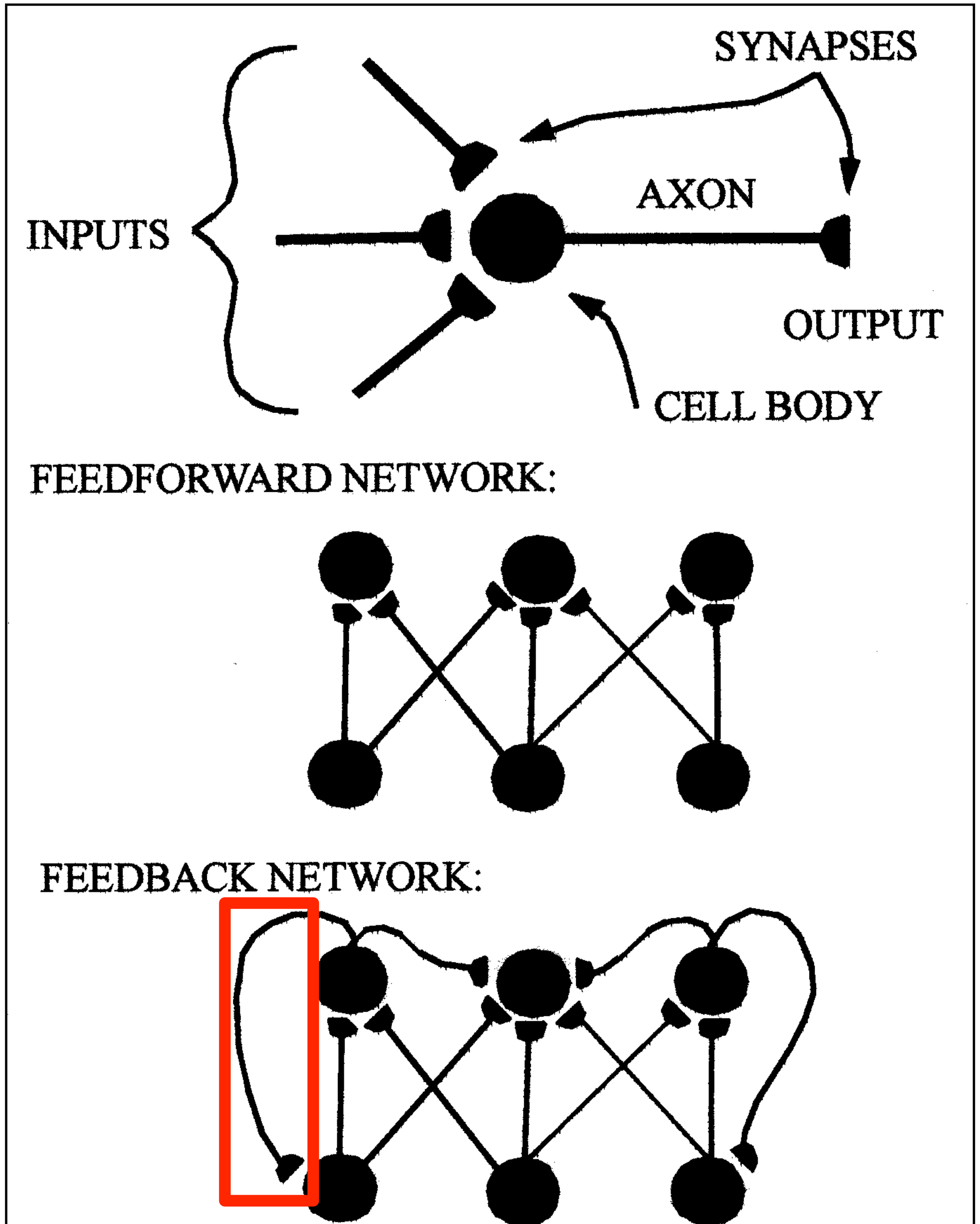
Many inputs:



Different activation functions

More complicated neural network models

Neural network models can become <u>much more complicated</u>:



"inputs"

Connect in layers

Q: Which are the "inputs?

Feedback between layers

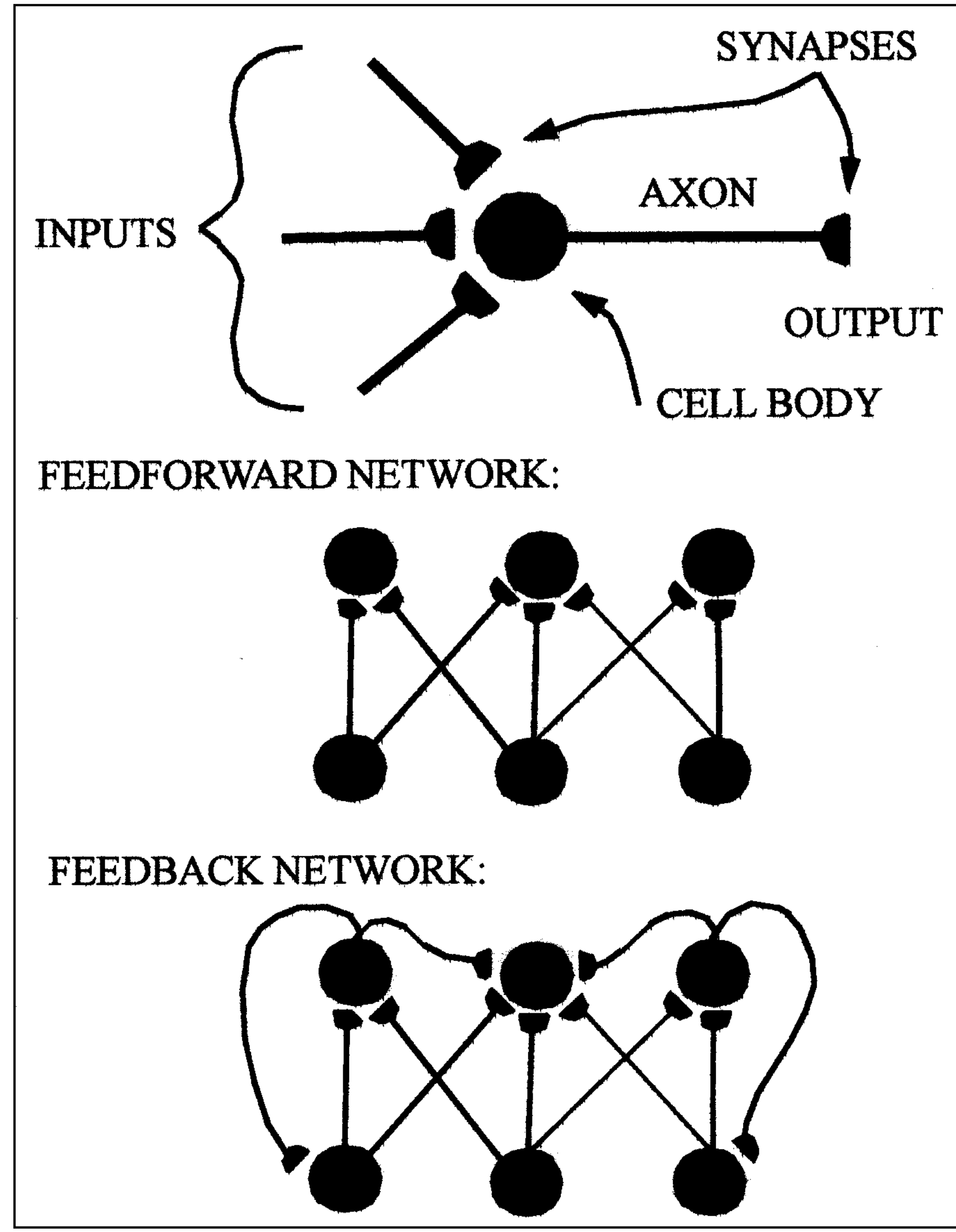
Neural network models

Summary:

- A neural network is a collection of abstracted neurons connected to each other through weighted connections ("synapses").

- The computations performed by these interconnected neurons are represented by mathematical equations or computer algorithms.

- **Learning:** The strengths of the weights are also represented by mathematical equations or computer algorithms.



MA666: Neural Networks and Learning

Part 2
Teaching the Perceptron

Now

We'll continue to study neural networks:

– The simplest case: the Perceptron.

– Simple pattern recognition

Challenge

Consider these data:

input 1	input 2	output = {0, 1}
0.9062	−0.6623	1.0000
0.8555	−0.8467	1.0000
1.9104	−0.5956	0
0.7769	−2.3029	0
2.5611	−1.2519	0
0.8517	−0.2829	1.0000
1.1616	−1.9551	0
1.7382	−0.8326	0
2.1395	−0.8733	0
1.0997	−0.4400	1.0000
3.1965	0.1410	0
1.8313	−1.0591	0
1.3909	−1.6422	0
0.1271	−1.6632	0
0.4838	−0.8297	1.0000
1.1555	−0.2390	1.0000

New data

input 1	input 2	
1.4134	−1.8730	?
1.6706	−0.7096	?
0.3063	−1.4071	?
1.3779	−1.8003	?
0.8425	−1.3501	.
1.0038	−0.1407	.
3.2511	−0.7492	.
−0.7264	0.3050	
0.1882	1.4591	
2.3571	−1.7109	

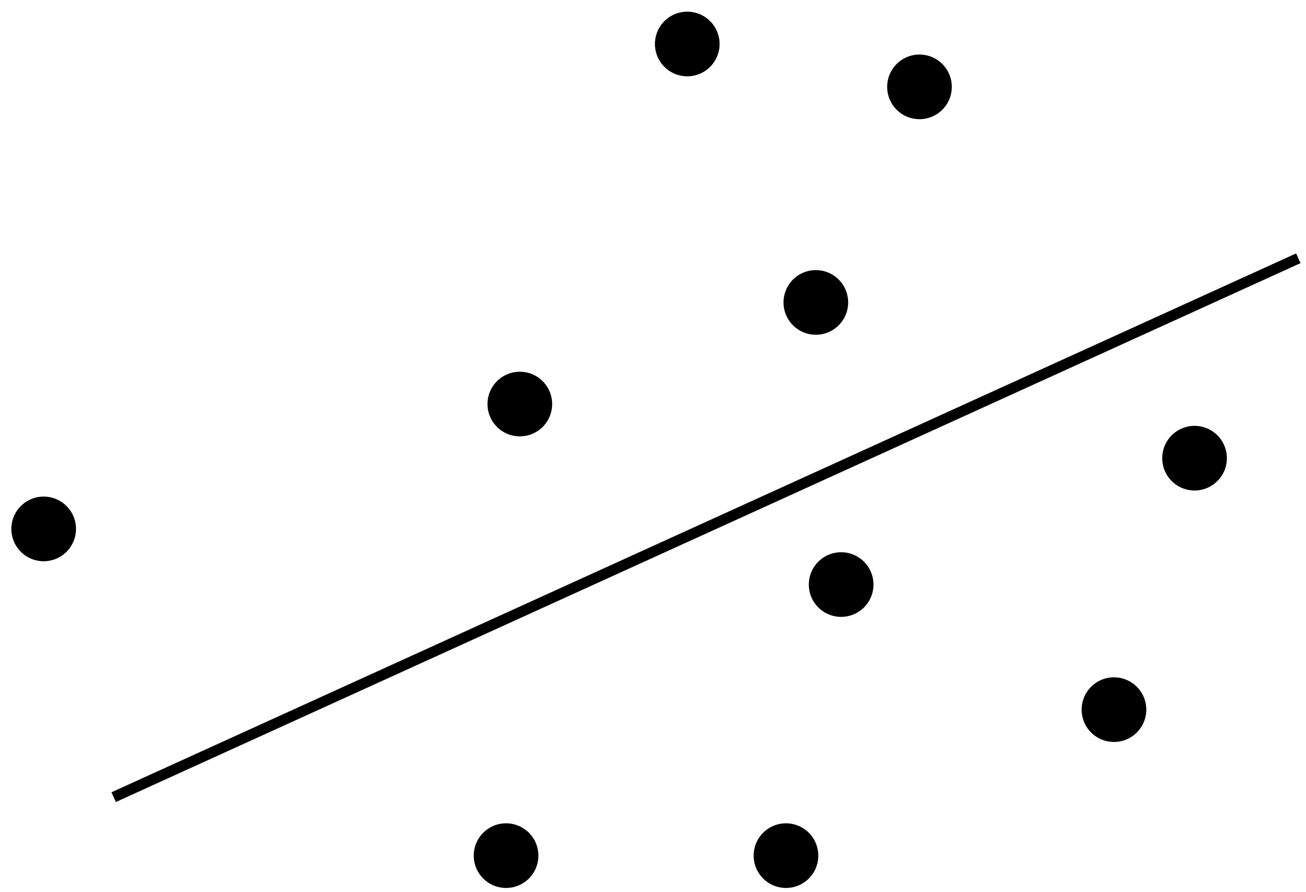
Perceptron: a classifier

Let's examine a perceptron in action …

Specifically, let's use a perceptron to **classify** some data.

Perceptron: a classifier

Consider a line:



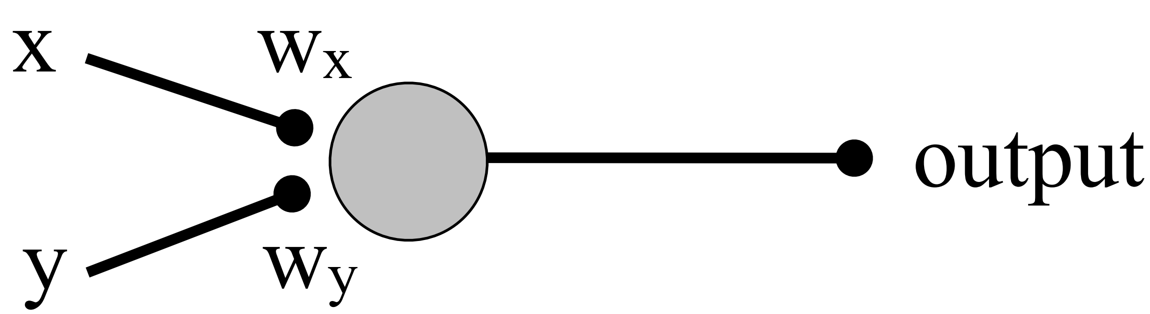
Note: Each point specified by (x,y) coordinate.

In this space, points are either “above” or “below” the line.

Q: Can we train a perceptron to recognize whether a point is above or below the line?

Perceptron: a classifier

Consider the perceptron:



Two inputs: the (x, y) coordinate of a point.

Use a <u>binary</u> activation function: output = $\{0, 1\}$

interpret as "below the line"

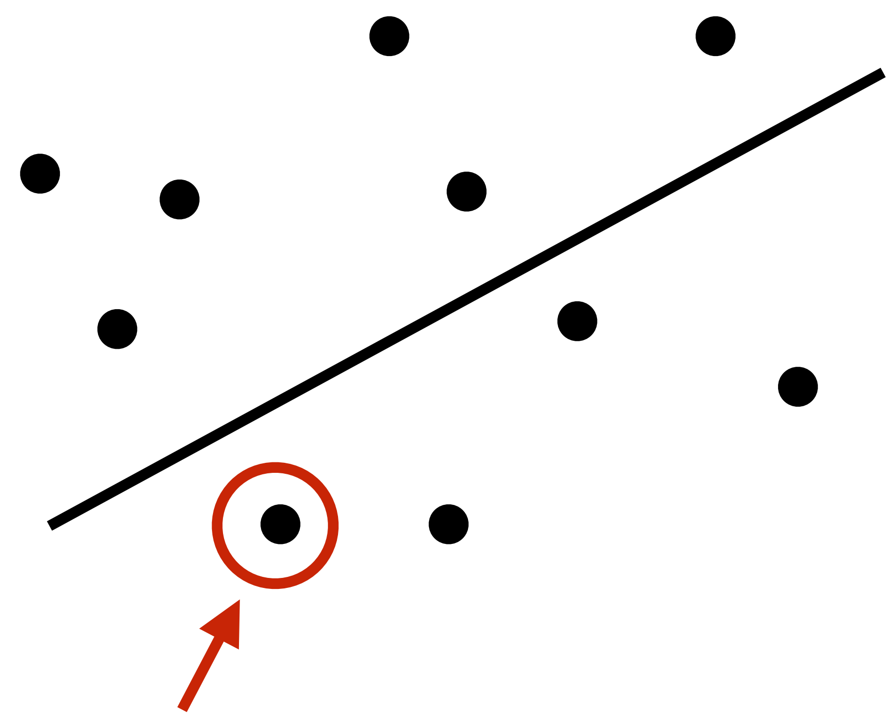
interpret as "above the line"

Weights: w_x, w_y

We'll need to specify those …

Perceptron classifier #1

We'd like to classify a point as either above or below this line:



Let's consider a point (-2, -3).

Q: What weights? To start let's choose: w_x=1, w_y=1

Q: What is the output?

binary activation function

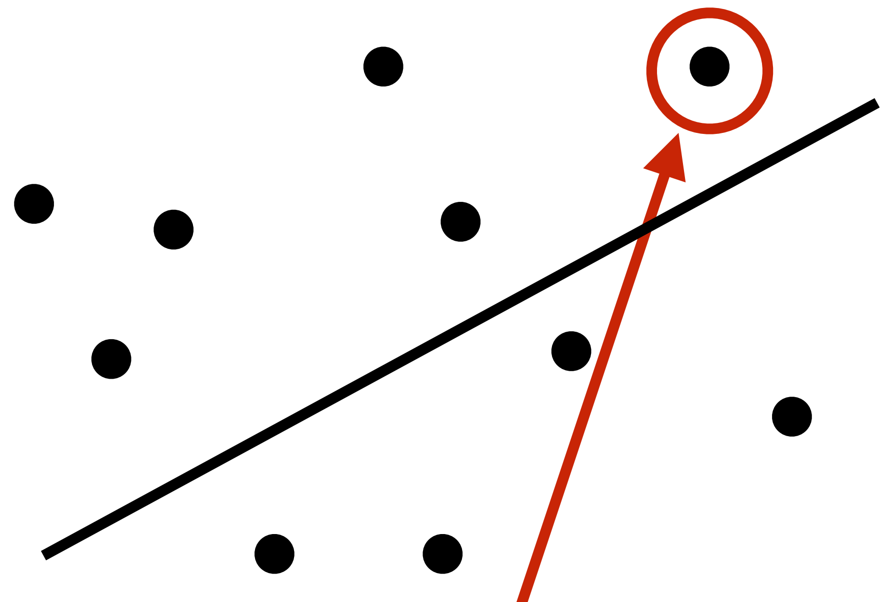
$x * w_x + y * w_y \quad = -2 * 1 + -3 * 1 = -5 < 0$ so, output = 0

Perceptron <u>succeeds</u>!

interpret as "below the line"

Perceptron classifier #1

We'd like to classify a point as either above or below this line:



Let's consider another point (2, 2).

Keep weights fixed at w_x=1, w_y=1

Q: What is the output?

binary activation function

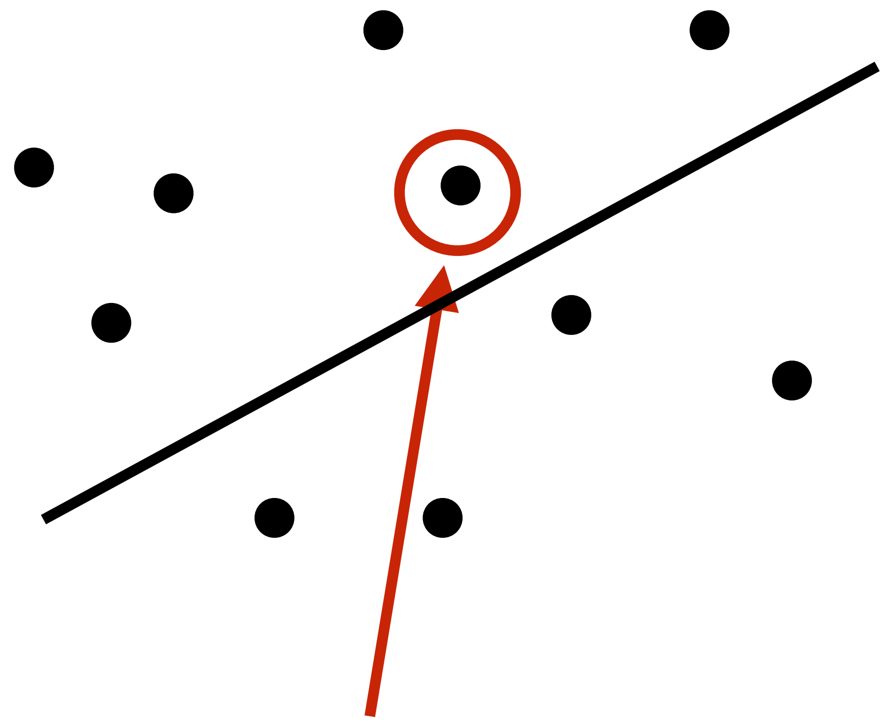
$x * w_x + y * w_y \quad = 2 * 1 + 2 * 1 = 4 \quad > 0$ so, output = 1

Perceptron <u>succeeds</u>!

interpret as "above the line"

Perceptron classifier #1

We'd like to classify a point as either above or below this line:



Let's consider another point (0, -1).

Keep weights fixed at $w_x=1$, $w_y=1$

Q: What is the output?

binary activation function

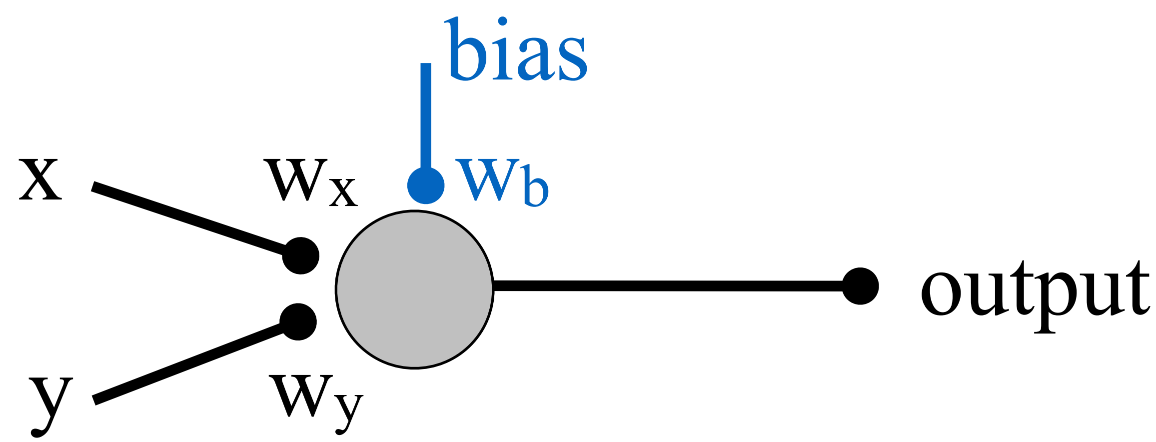
$x * w_x + y * w_y \quad = 0 * 1 + (-1) * 1 = -1 < 0$ so, output = 0

interpret as "below the line"

Perceptron <u>fails</u>!

Perceptron classifier #2

To correct this error, add another input: **bias**



We'll set *bias* = 1, and multiply it by a weight (w_b)

Let's reconsider the troublesome point (0, -1). Then, the output:

$$x * w_x + y * w_y + \text{bias} * w_b = 0 * 1 + (-1) * 1 + 1 * w_b = -1 + w_b$$

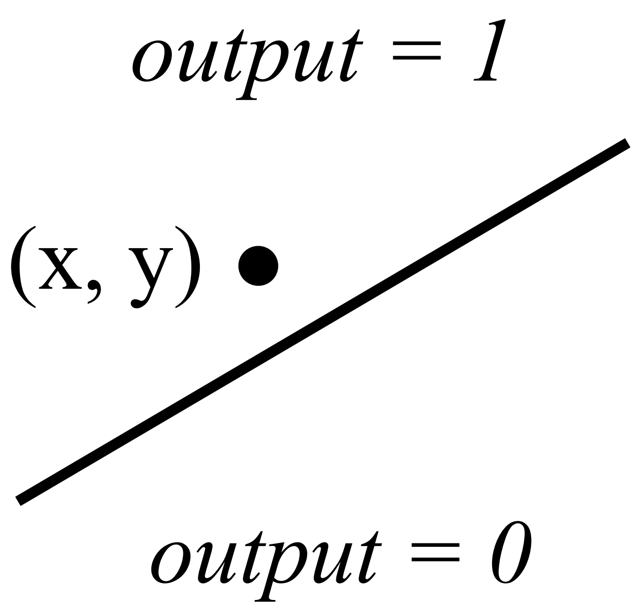
So, if $w_b > 1$	then output = 1	interpret as "above the line"
Note, if $w_b < 1$	then output = 0	interpret as "below the line"

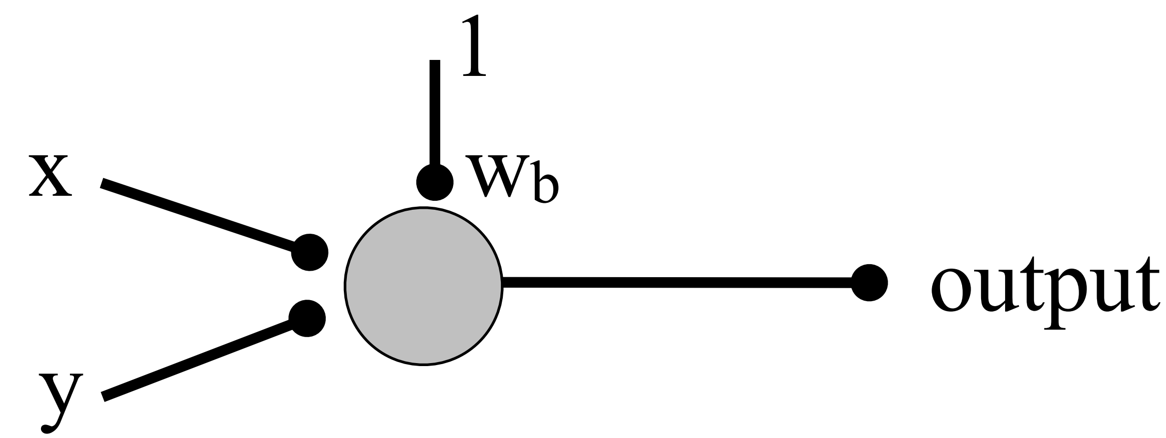
- The bias acts to "bias" the perceptron's output.

Use weights to set perceptron's knowledge: (0,-1) above or below line?

Perceptron classifier #2: Summary

Summary of perceptron classifier:





For any point (x,y) ask the perceptron:

Is the point above (output 1) or below (output 0) the line?

Q: Will the perceptron get classification right?

A: If we're lucky, then maybe … but we need to train it!

Perceptron training

To train our perceptron, we'll use **supervised learning**.

- We'll provide our perceptron with inputs & correct answer.
- The perceptron will compare its guess with the correct answer.
  - If the perceptron makes an <u>incorrect</u> guess, then it can <u>learn</u> from it's mistake

adjust its weights

Let's do it ….

Perceptron training

Perceptron training in <u>5 steps</u>:

1. Provide perceptron with inputs and known answer.
2. Ask perceptron to guess an answer.
3. Compute the error: does perceptron get answer right or wrong?
4. Adjust all weights according to the error. **Learning!**
5. Return to Step 1 and repeat.

Note: We know how to do <u>Step 2</u>, consider other steps …

forward propagation

Perceptron training: Step 3

Consider <u>Step 3</u>. *Compute the error*

Q: What is the perceptron's error?

Let's define it:

Difference between desired answer and perceptron's guess.

Error = Desired output - Perceptron output

In our case: $\{0, 1\}$ $\{0, 1\}$

Remember, the output has only 2 possible states.

Perceptron training: Step 3

Let's make a table of possible error values:

Desired output	Perceptron output	Error	
0	0	0	ok!
0	1	-1	:(
1	0	1	:(
1	1	0	ok!

Note: the error is 0 when perceptron guesses the correct output

the error is +1 or -1 when perceptron guesses the wrong output

Next step: use the error to adjust the weights …

Perceptron training: Step 3

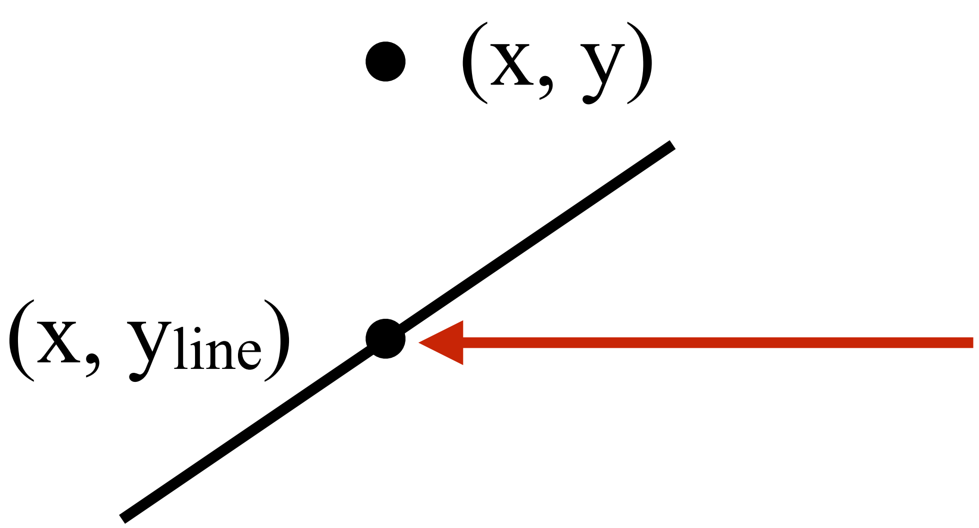
Q: How do we know if a point is above or below the line?

Remember the formula for a line:

$\mathbf{y_{line} = m*x + b}$

m = slope of line
b = intercept of line

Given a point:



Compute: $y_{line} = m*x + b$

A: Compare y_{line} versus y.

If $y > y_{line}$ then y is above the line

Perceptron training: Step 4

Consider Step 4. *Adjust all weights according to the error.*

The error determines how weights should be adjusted.

Let's define the change in weight:

$$\triangle \text{weight} = \text{Error} * \text{Input}$$

Then, to update the weight:

$$\begin{aligned}\text{New weight} &= \text{weight} + \triangle \text{weight} \\ &= \text{weight} + \text{Error} * \text{Input}\end{aligned}$$

Note: The error determines how the weight should be adjusted

big error — big change in weight

Perceptron training: Step 4

So, for our perceptron to learn:

– adjust the weights according to the error.

We'll also include a **learning constant**:

Compute this for Step 4:

$$\boxed{\text{New weight} = \text{weight} + \text{Error} * \text{Input} * \text{Learning Constant}}$$

When learning constant is big: weights change more drastically.

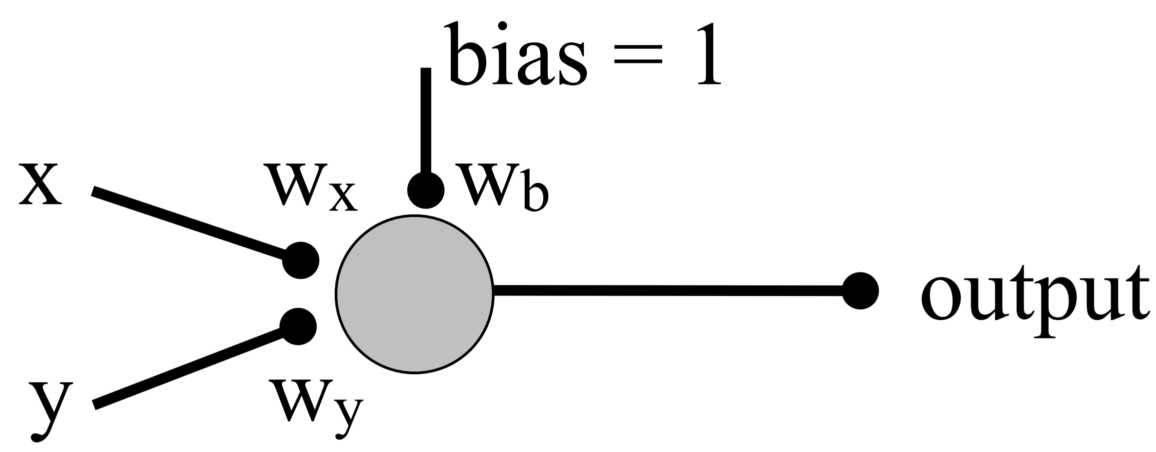
- Get to a solution more quickly.

When learning constant is small: weights change more slowly.

- Small adjustments improve accuracy

Perceptron training: by-hand

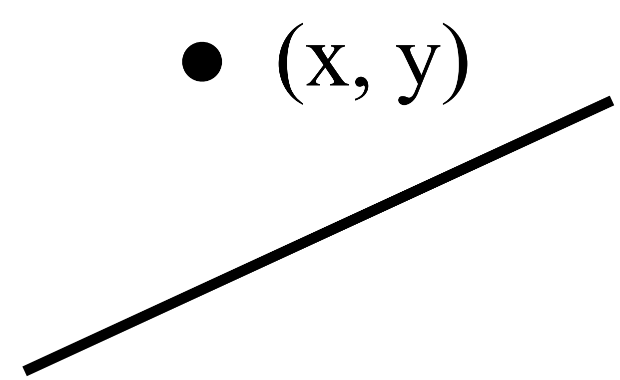
Let's train the perceptron …



Initialize:

All weights = 0.5

Learning constant = 0.01



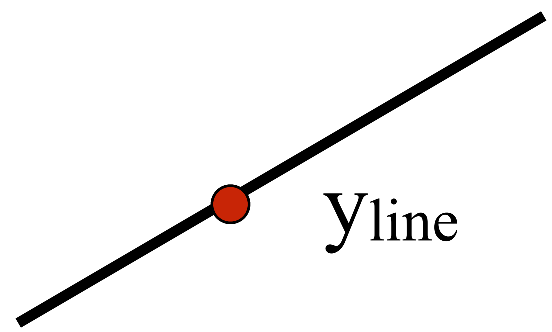
Define line: $y = 2x + 1$

This is the relationship we want our perceptron to learn …

Perceptron training: by-hand

<u>Step 1</u>: *Provide perceptron with inputs and known answer.*

• choose (0.7, 3)
x y



y_{line}

line: $y = 2x + 1$

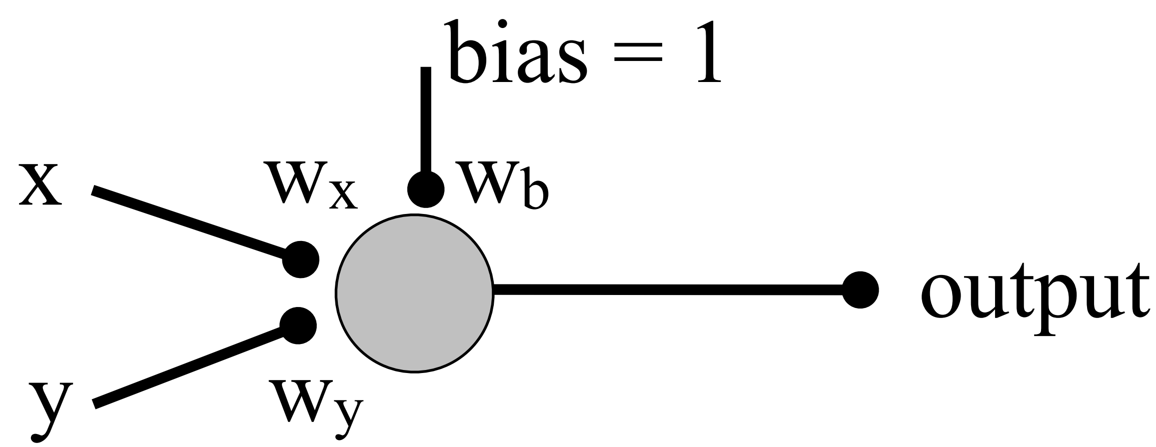
line @ x=0.7: $y_{line} = 2*0.7 + 1 = 2.4$

So, $y > y_{line}$

So, y is <u>above</u> the line. (this is the known answer)

Perceptron training: by-hand

<u>Step 2</u>. *Ask perceptron to guess an answer.*



Compute weighted summed inputs:

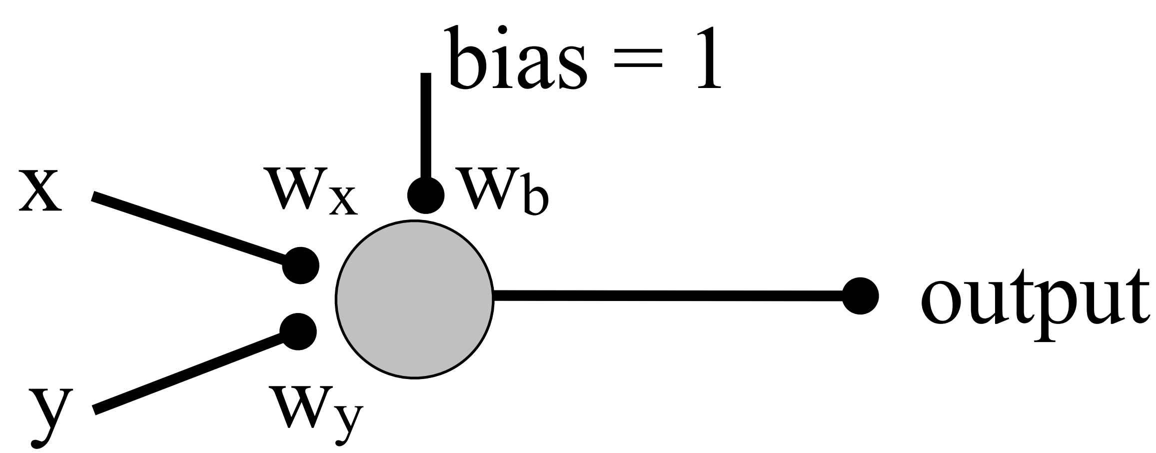
$$w_x\,x + w_y\,y + w_b\,\text{bias} = 0.5 * \underset{x}{0.7} + 0.5 * \underset{y}{3} + 0.5 * \underset{\text{bias}}{1} = 2.35$$

So, $w_x\,x + w_y\,y + w_b\,\text{bias} > 0$

So, $\boxed{\text{output} = 1}$

Perceptron training: by-hand

Step 3. *Compute the error.*



Perceptron output = 1 (Perceptron: "point is above the line")

Desired output = 1 (Us: the point is above the line.)

Error = Desired output - Perceptron output

= 1 - 1

= 0 No error, perceptron guess is correct.

Perceptron training: by-hand

Step 4. *Adjust all weights according to the error.*

$$\boxed{\text{New weight} = \text{weight} + \text{Error} * \text{Input} * \text{Learning Constant}}$$

w_x : 0.5 + 0 * 0.7 * 0.01 = 0.5

w_y : 0.5 + 0 * 3 * 0.01 = 0.5

w_b : 0.5 + 0 * 1 * 0.01 = 0.5

No change in weights!

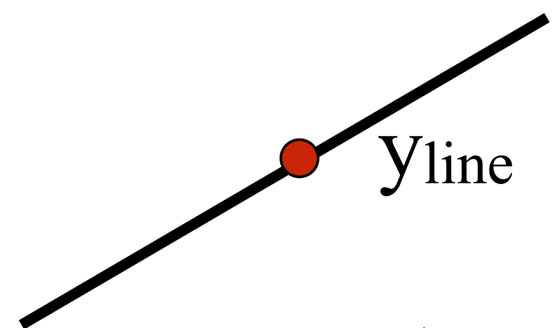
Q: Our Perceptron is already "smart enough"?

Step 5. *Return to Step 1 and repeat …*

Perceptron training: by-hand

Step 1: *Provide perceptron with inputs and known answer.*

line: $y = 2x + 1$



Choose another point: • (1, 0)

x y

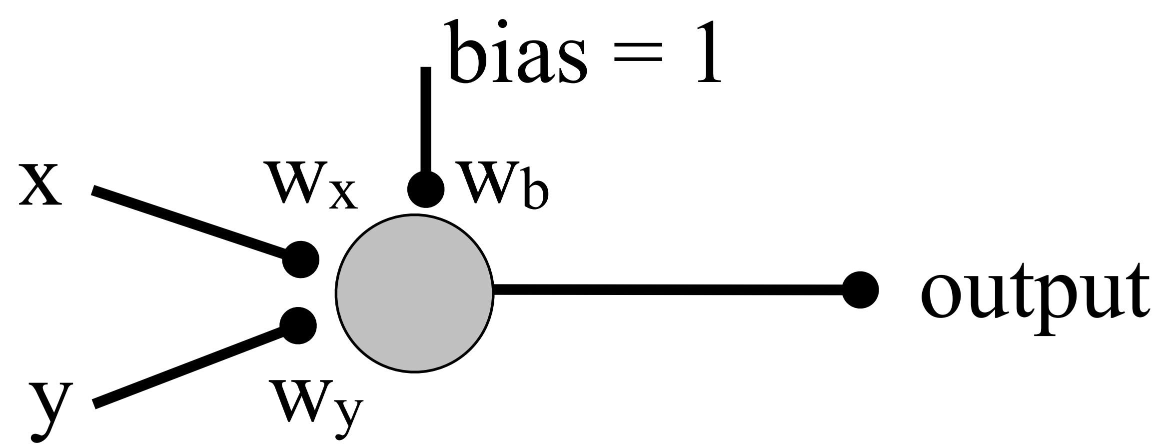
line @ x=1: $2*1 + 1 = 3 = y_{line}$

So, $y < y_{line}$

So, y is <u>below</u> the line. (this is the known answer)

Perceptron training: by-hand

<u>Step 2</u>. *Ask perceptron to guess an answer.*



Compute weighted summed inputs:

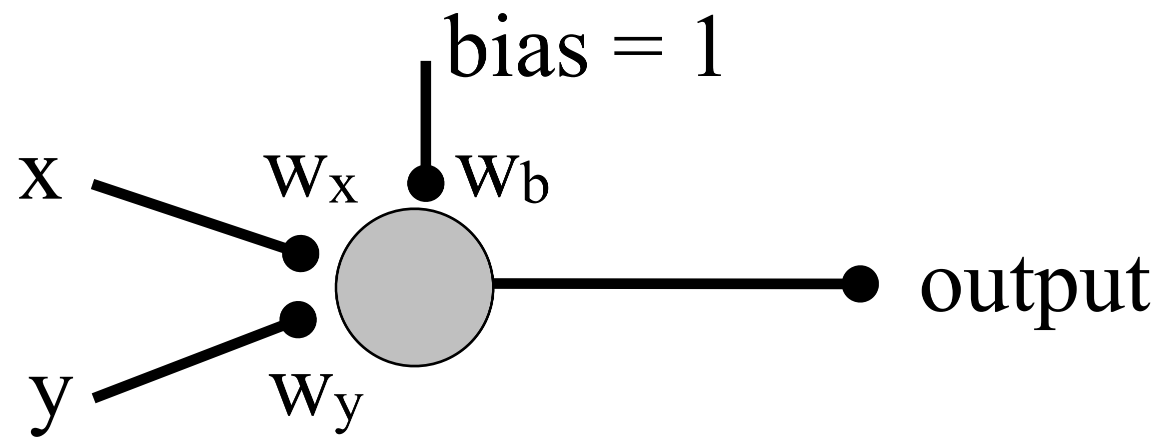
$$w_x x + w_y y + w_b \text{bias} = 0.5 * \underset{x}{1} + 0.5 * \underset{y}{0} + 0.5 * \underset{\text{bias}}{1} = 1$$

So, $w_x x + w_y y + w_b \text{bias} > 0$

So, $\boxed{\text{output} = 1}$

Perceptron training: by-hand

Step 3. *Compute the error.*



Perceptron output = 1 (Perceptron: “point is above the line”)

Desired output = 0 (Us: the point is below the line.)

Error = Desired output - Perceptron output

= 0 - 1

= -1 Error, the perceptron guess is wrong.

Perceptron training: by-hand

<u>Step 4</u>. *Adjust all weights according to the error.*

New weight = weight + Error * Input * Learning Constant

w_x :	0.5	+	-1	* 1	* 0.01	=	0.49
w_y :	0.5	+	-1	* 0	* 0.01	=	0.5
w_b :	0.5	+	-1	* 1	* 0.01	=	0.49

We've changed the weights!

Q: Our Perceptron is already "smart enough"?

A: No, our Perceptron is "getting smarter"

Perceptron training: by-hand

<u>Step 5</u>. *Return to Step 1 and repeat …*

In fact, repeat the entire process 1000 times (or more).

Each time:

- Choose a random (x,y).
- Determine if it's above or below 2x + 1.
- Ask the perceptron.
- Adjust the weights.

Q: Could you do this by hand?

Q: <u>Would</u> you do this by hand?

Python

Computer lab: implement a learning perceptron.