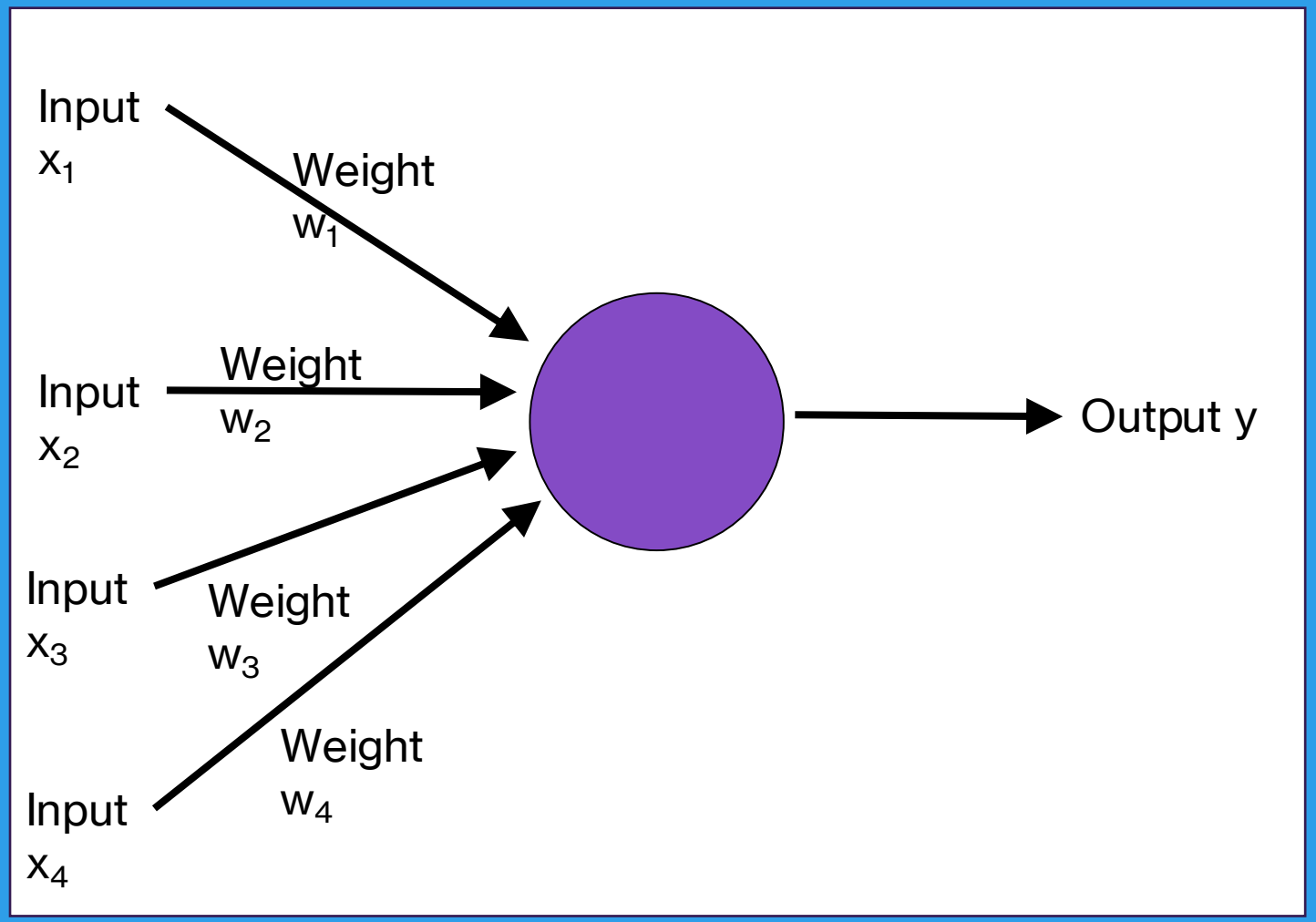
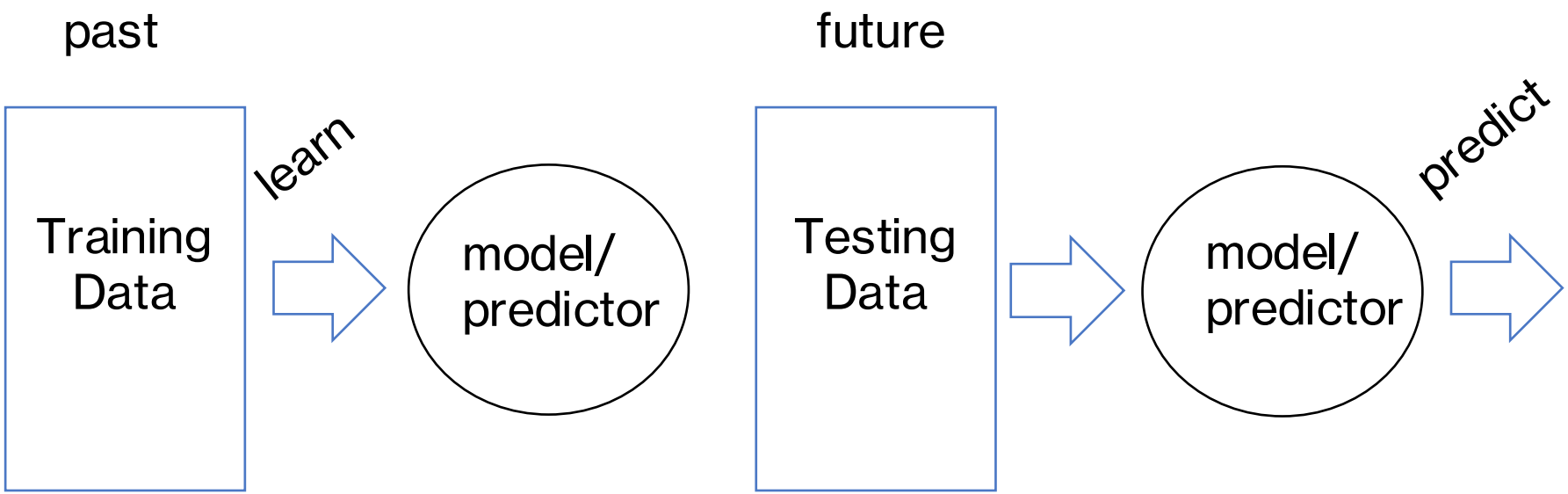
Machine Learning



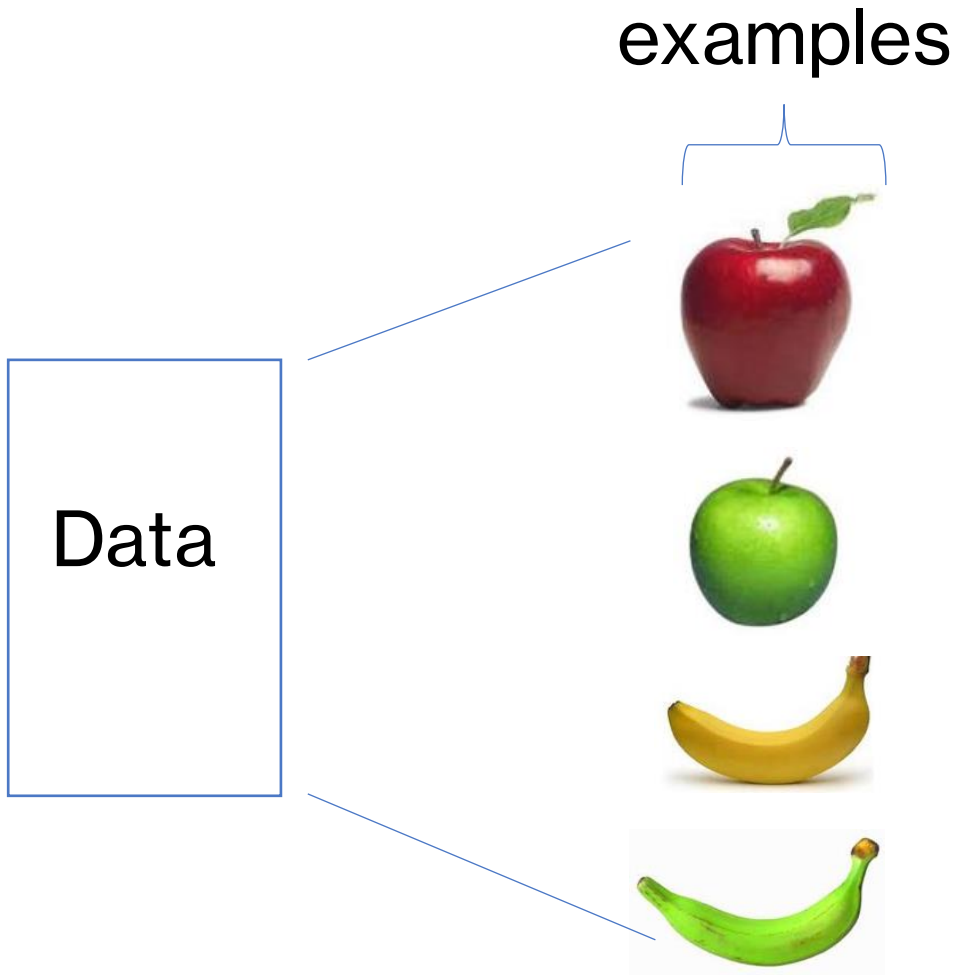
Intro to Machine Learning and Perceptron

CS51 – Spring 2026

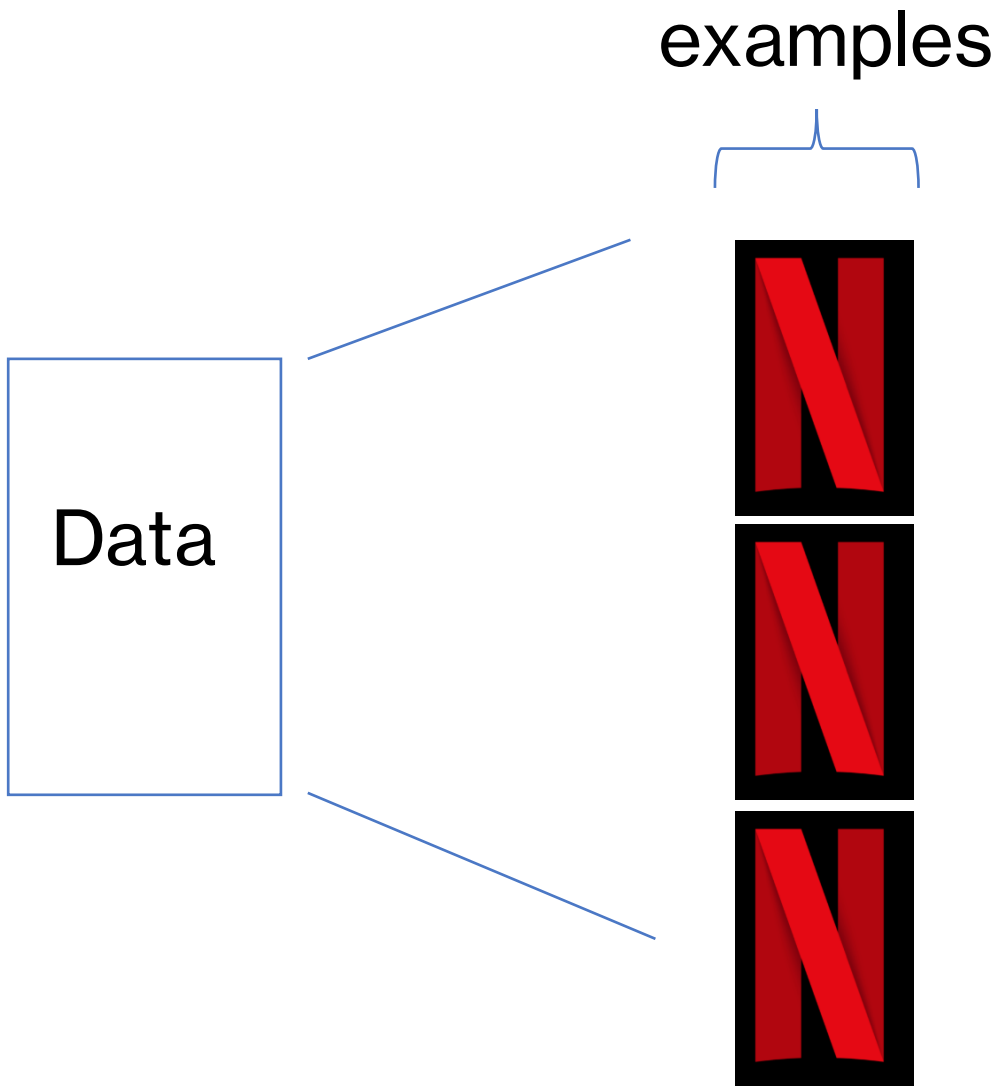
Machine Learning



Data



Data



Supervised learning learns from labeled examples

examples	labels	class
	red, round, ...	apple
	green, round, ...	apple
	yellow, oblong, ...	banana
	green, oblong, ...	banana

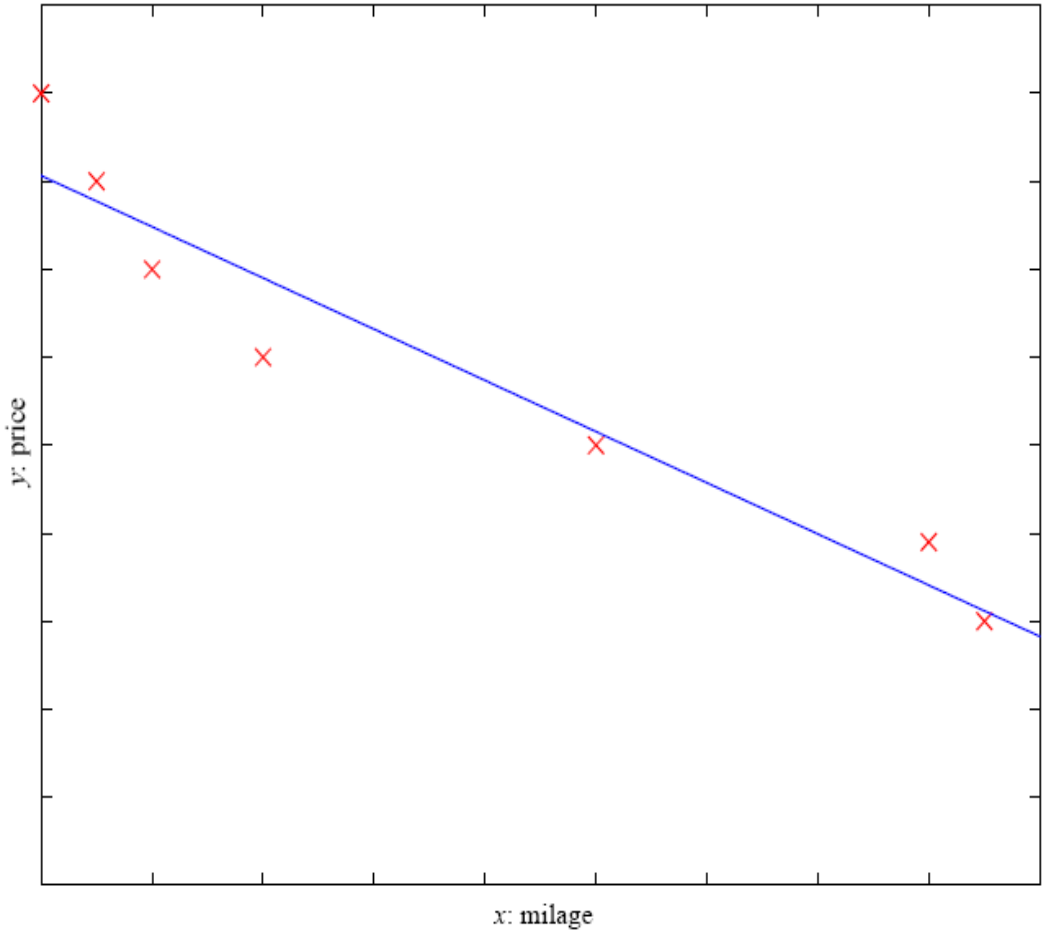
Data

Classification

- Predicts a discrete category (e.g., yes/no, or a discrete number of classes)
- Face recognition
- Character recognition
- Spam detection
- Medical diagnosis
- Biometrics

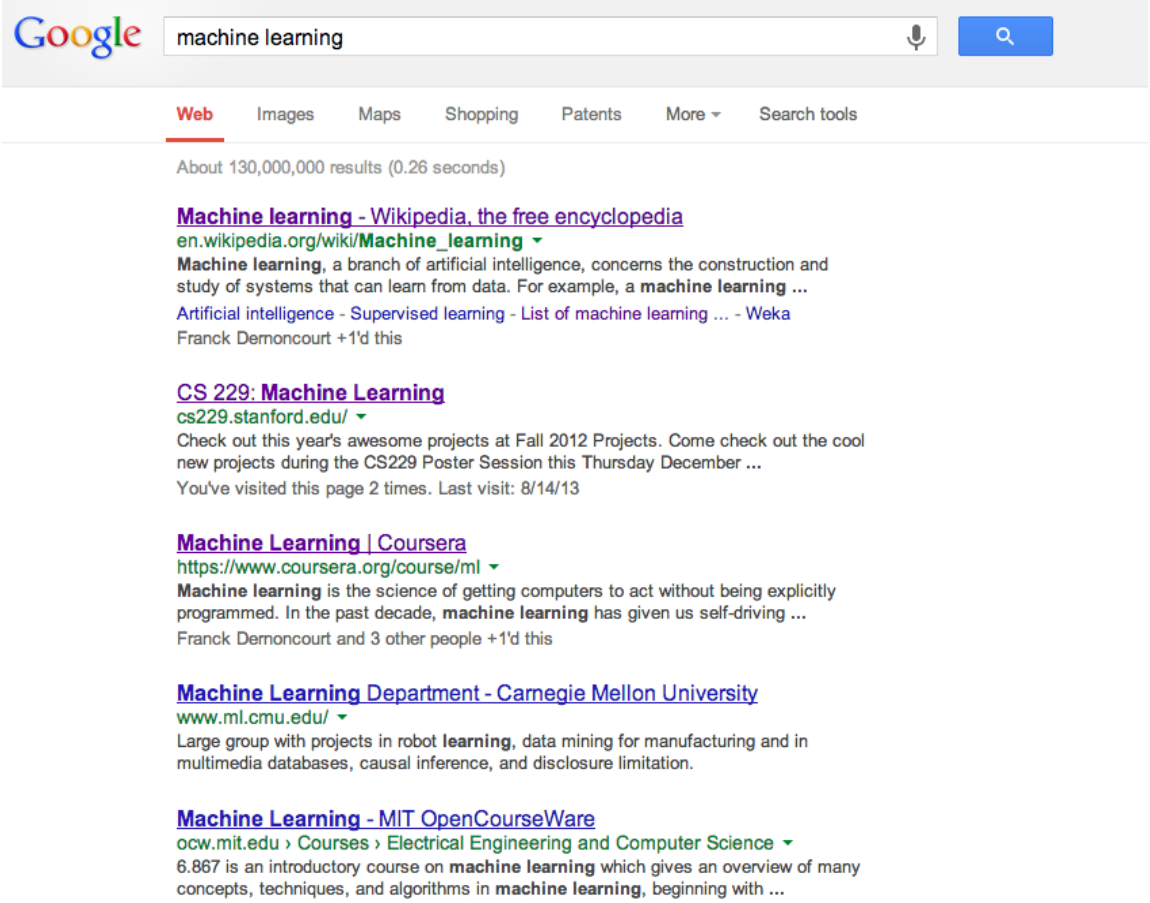
Regression

- Predicts a continuous numerical value
- Economics/Finance
- Epidemiology
- Car/plane navigation
- Temporal trends

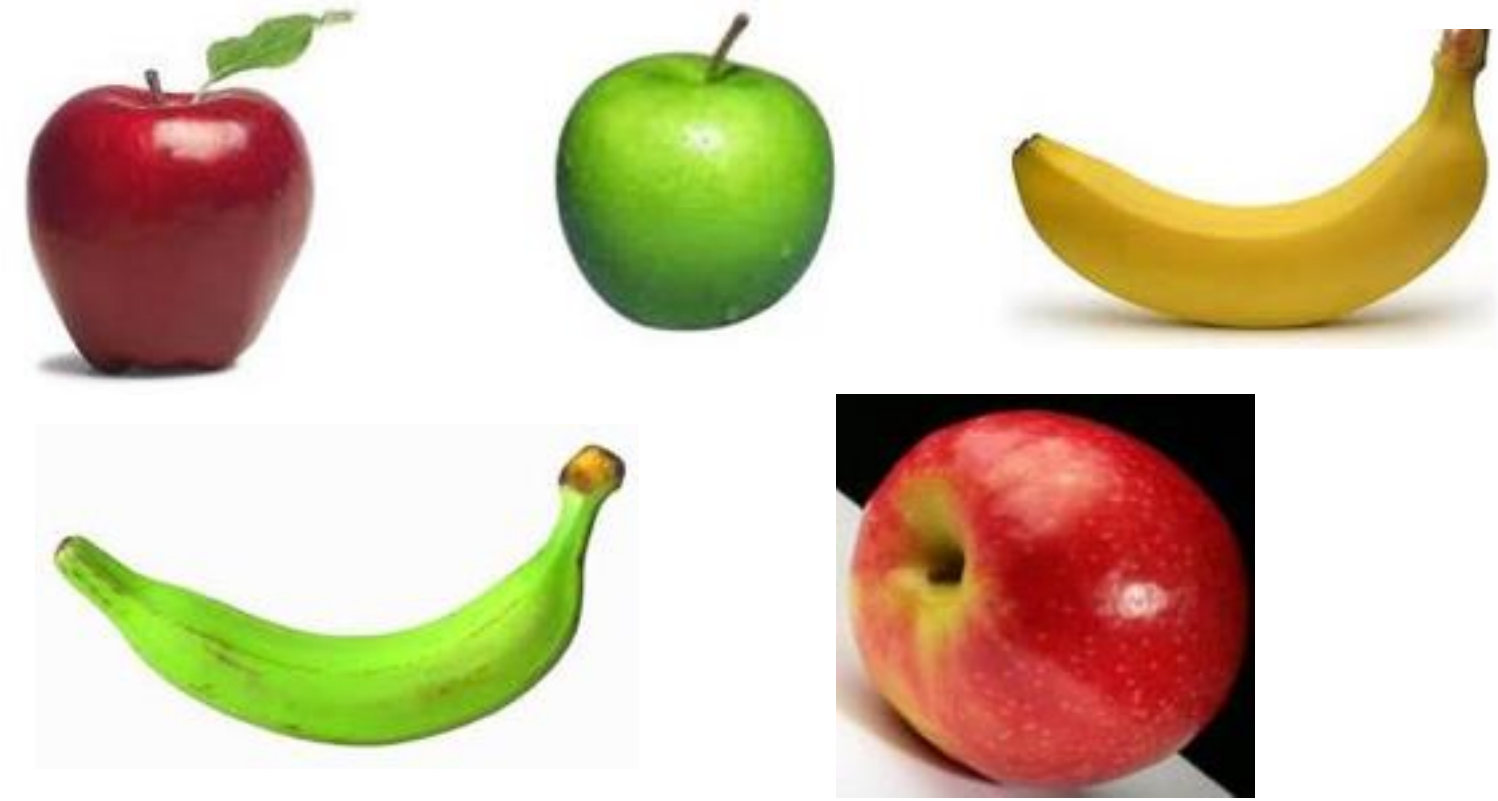


Ranking

- Predicts a ranking of items
- Web search
- Movie/music recommendation
- Flight search



Unsupervised learning



- Provided with data but not labels

Reinforcement learning

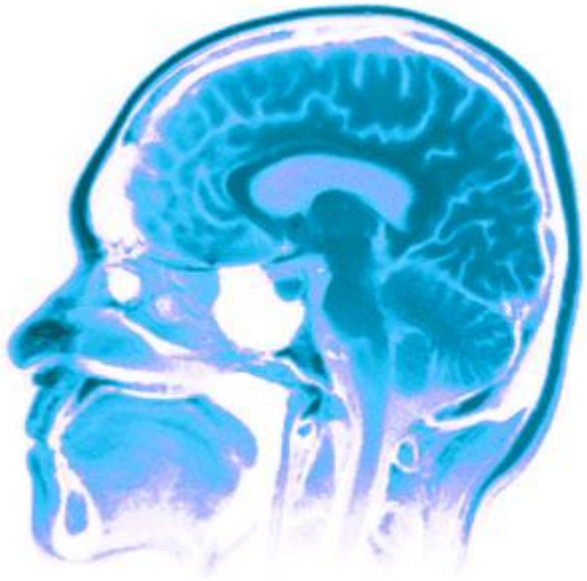
left, right, straight, left, left, left, straight	GOOD
left, straight, straight, left, right, straight, straight	BAD

left, right, straight, left, left, left, straight	18.5
left, straight, straight, left, right, straight, straight	-3

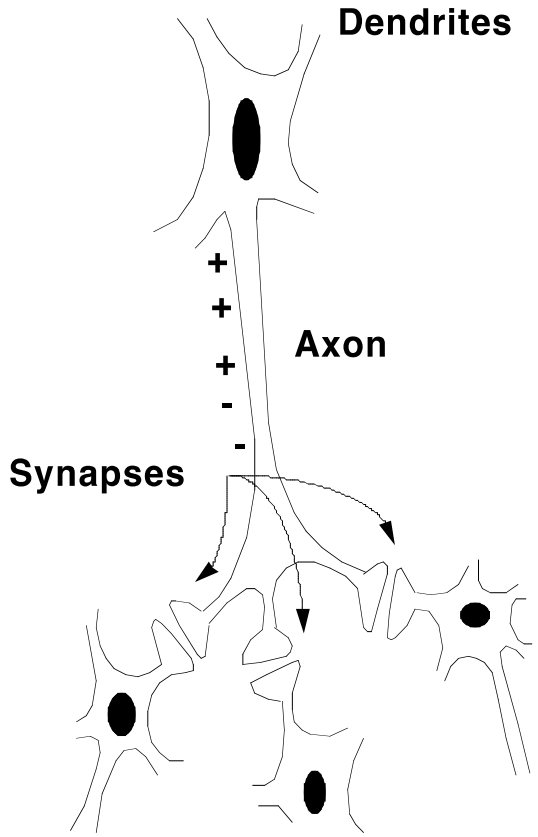
Given a sequence of examples/states and a reward after completing that sequence, learn to predict the action to take in for an individual example/state

Neural networks

- Neural Networks try to mimic the structure and function of our nervous system.



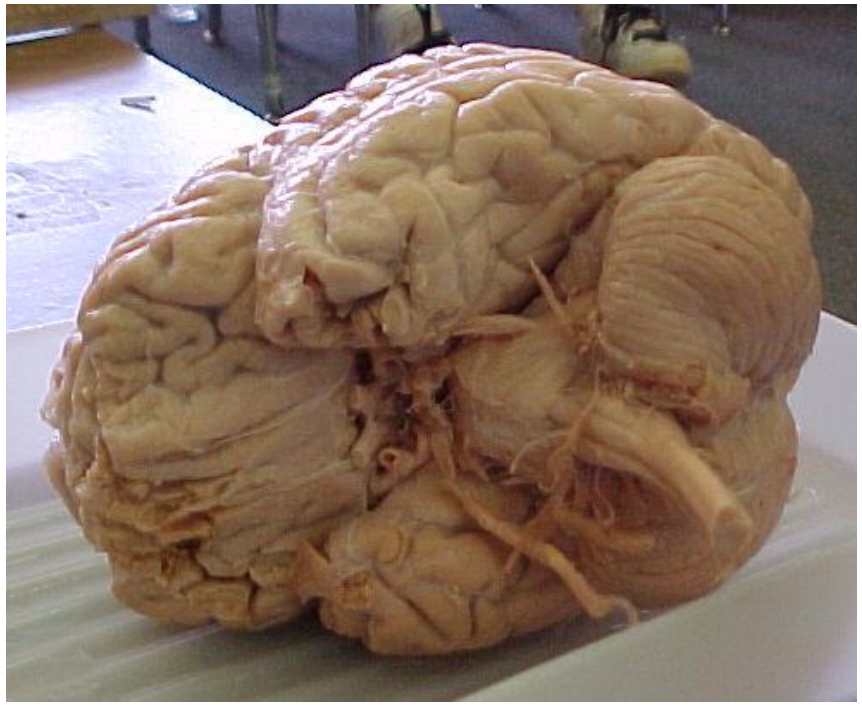
Our nervous system according to CS



Neuron

- the human brain is a large collection of interconnected neurons
- a neuron is a brain cell
  - they collect, process, and disseminate electrical signals
  - hey are connected via synapses
  - they fire depending on the conditions of the neighboring neurons

Our brains in numbers

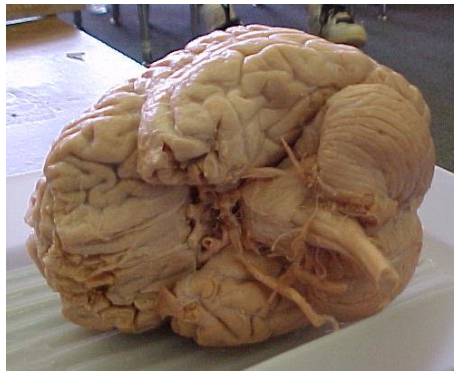


The human brain

- contains ~10^{11} (100 billion) neurons
- each neuron is connected to ~10^{4} (10,000) other neurons
- Neurons can fire as fast as 10^{-3} seconds

How does this compare to a computer?

Human Vs. Machine



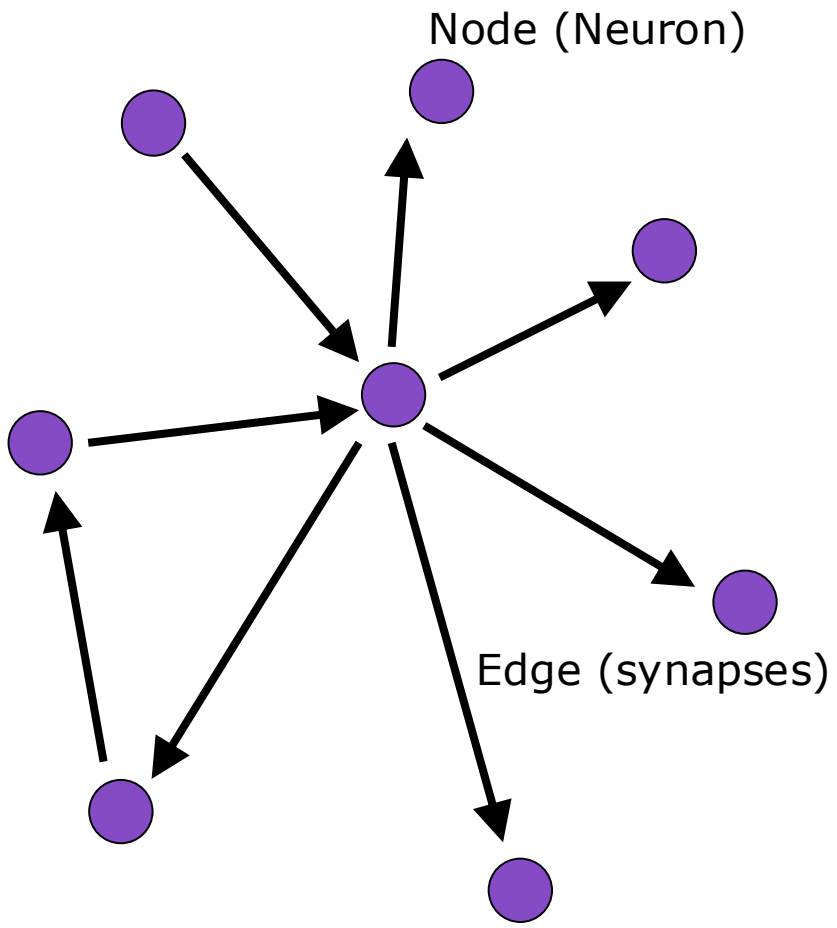
10^{11} neurons
10^{11} neurons
10^{14} synapses
10^{-3} “cycle” time

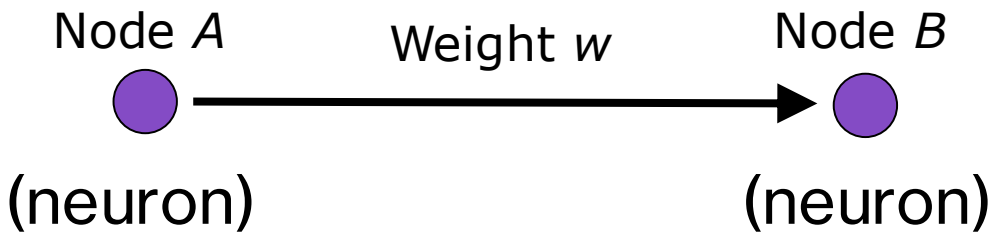
10^{10} transistors
10^{11} bits of RAM
10^{13} bits on disk
10^{-9} cycle time

Brains are still pretty fast

Who is this?

Artificial neural networks

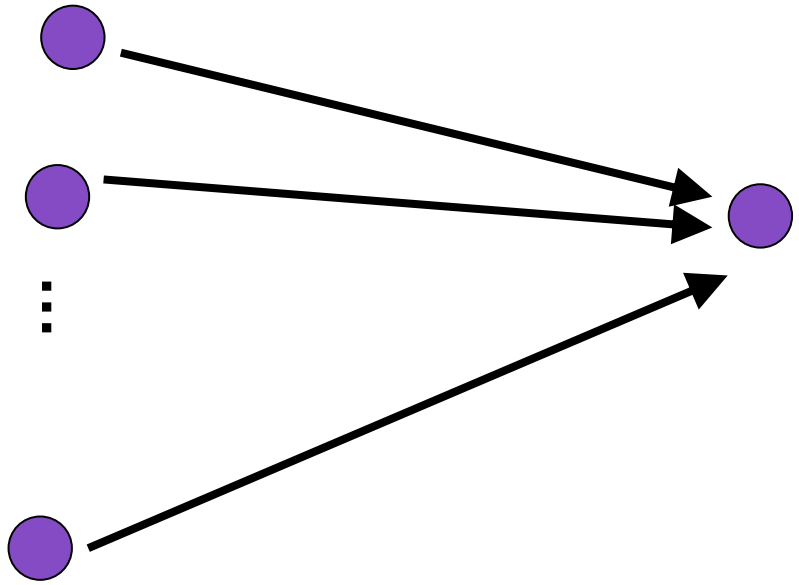




W is the strength of signal sent between A and B.

If *A* fires and *w* is **positive**, then *A* **stimulates** *B*.

If *A* *fires* and *w* is **negative**, then *A* **inhibits** *B*.

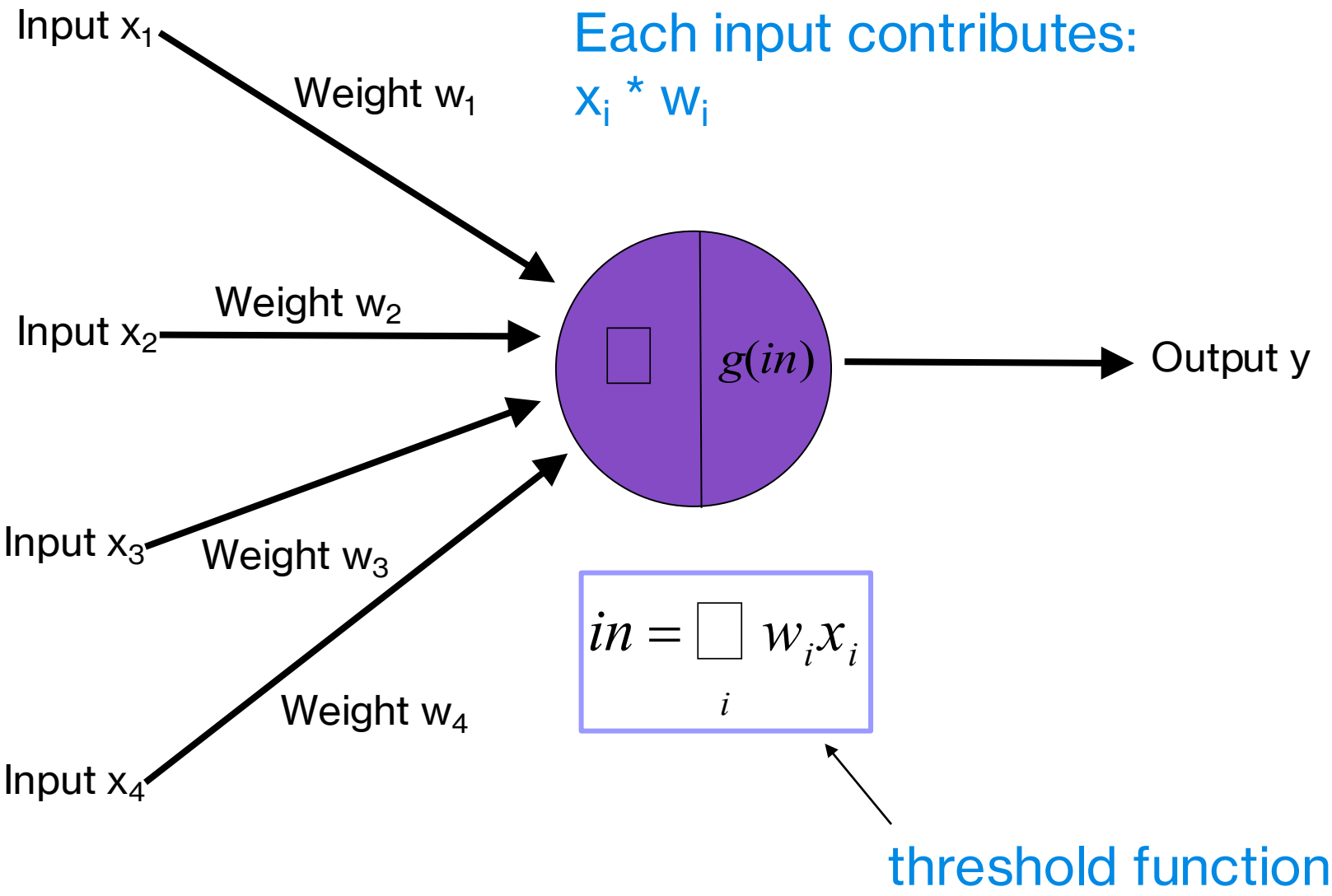


A given neuron has many, many connecting, input neurons

If a neuron is stimulated enough, then it also fires

How much stimulation is required is determined by its **threshold**

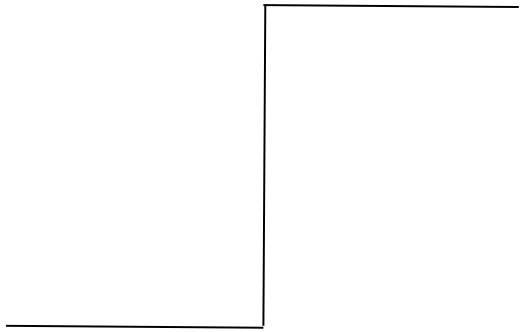
A single neuron/perceptron



Possible thresholds

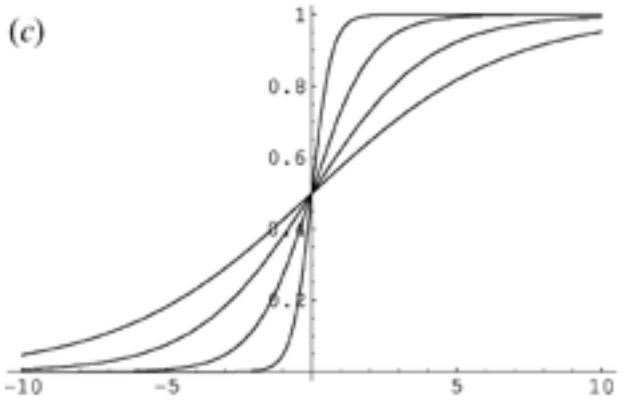
hard threshold

$$g(x) = \begin{cases} 1 & if \ x \geq threshold \\ 0 & otherwise \end{cases}$$

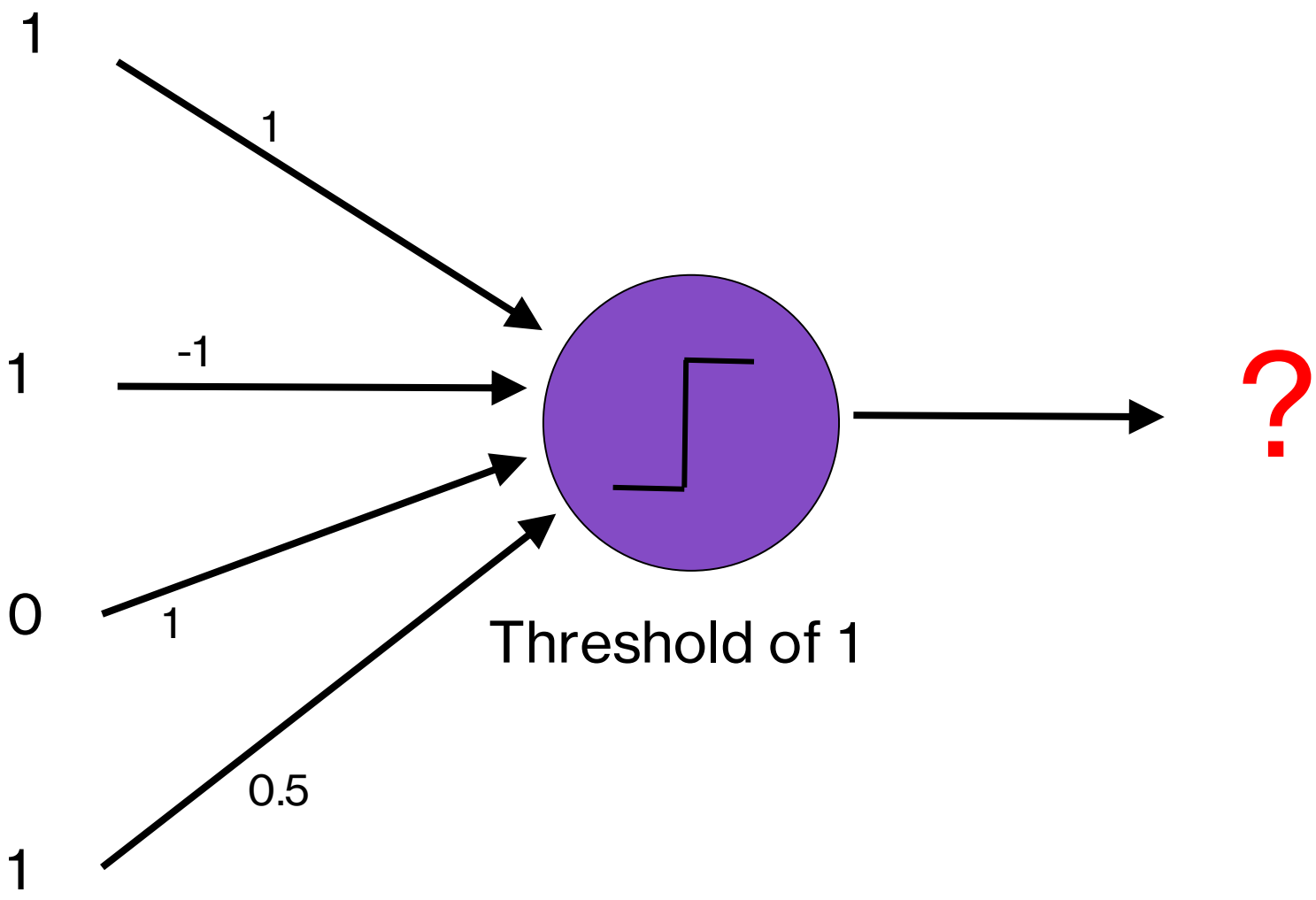


sigmoid

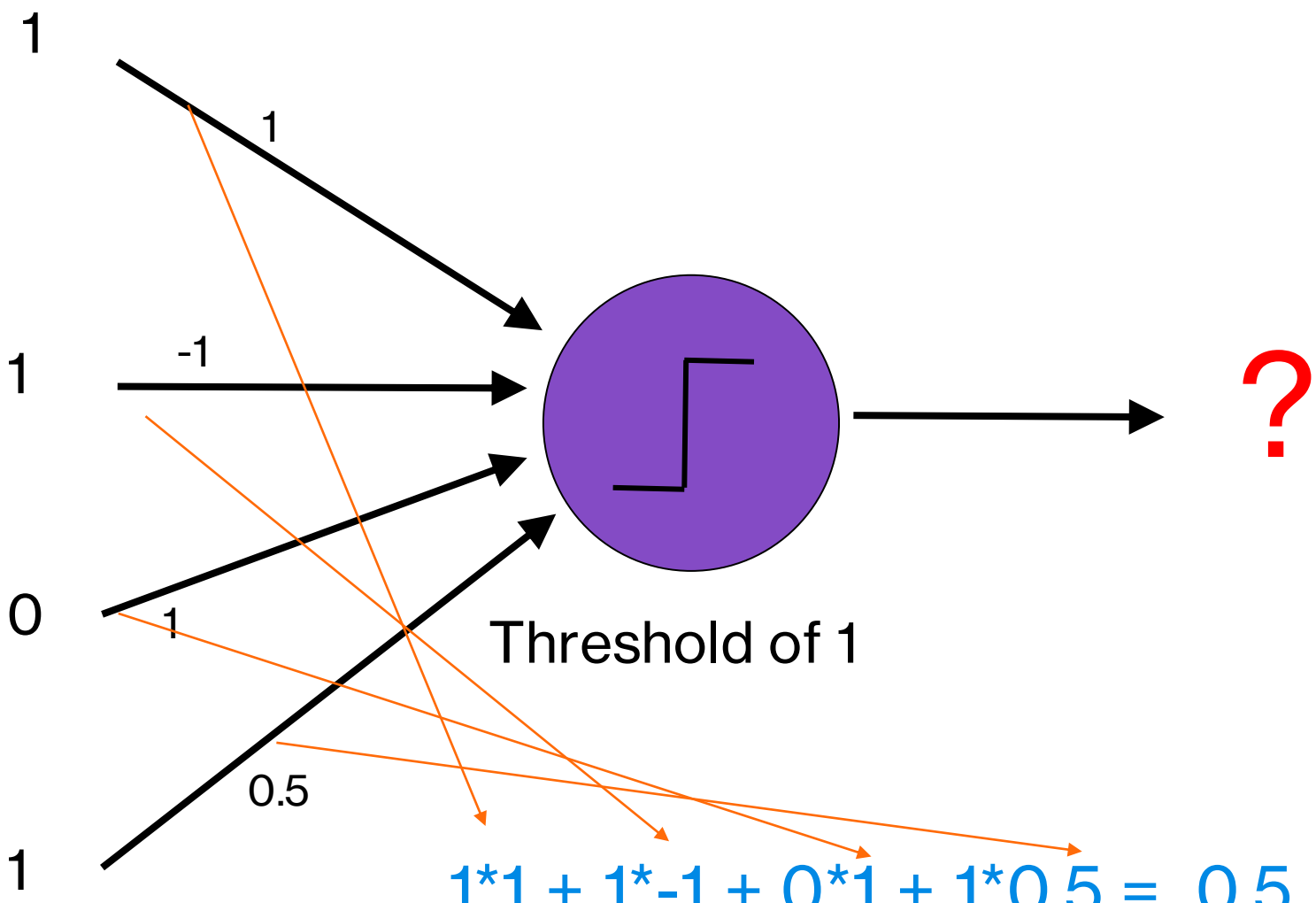
$$g(x) = \frac{1}{1 + e^{-ax}}$$



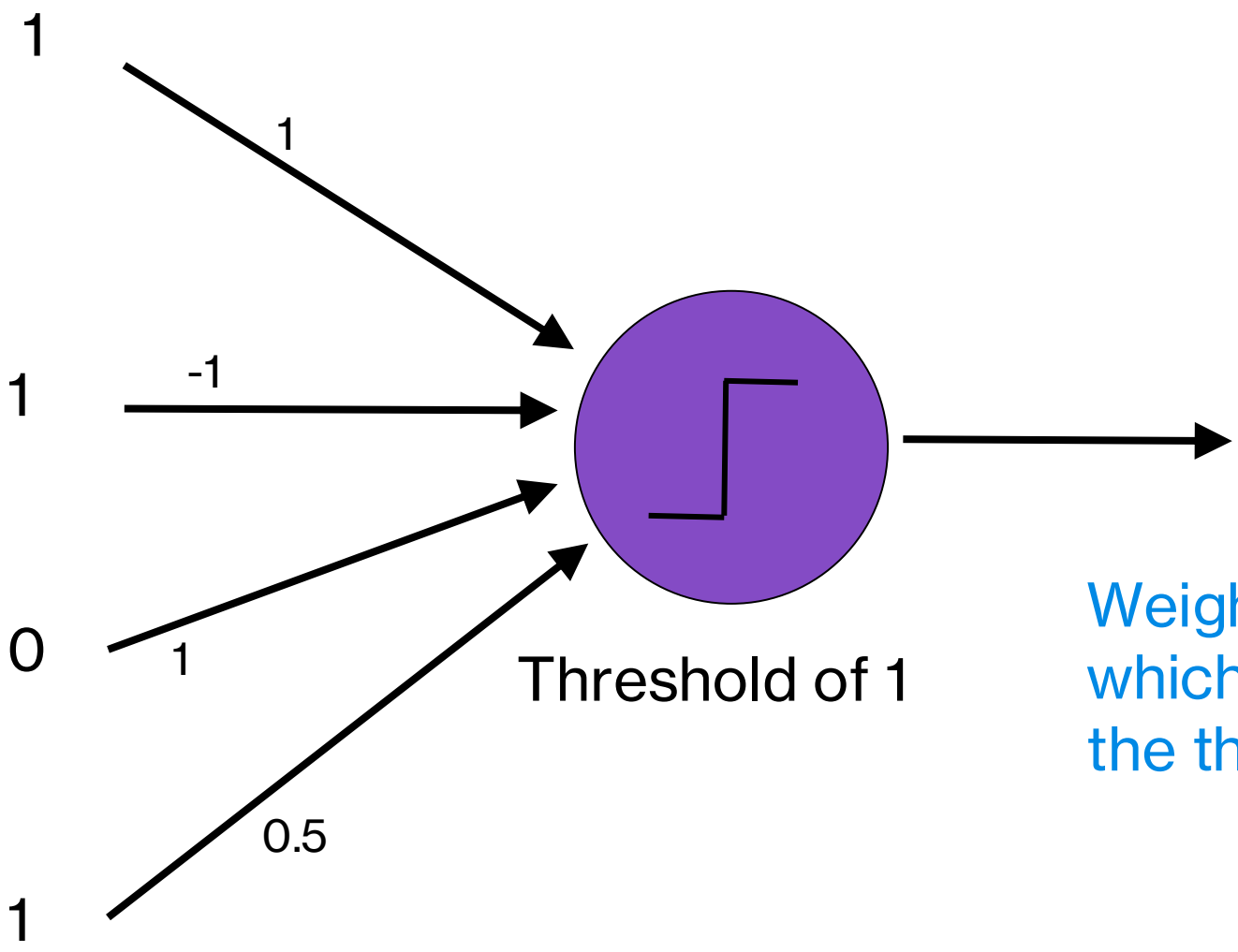
A single neuron/perceptron



A single neuron/perceptron

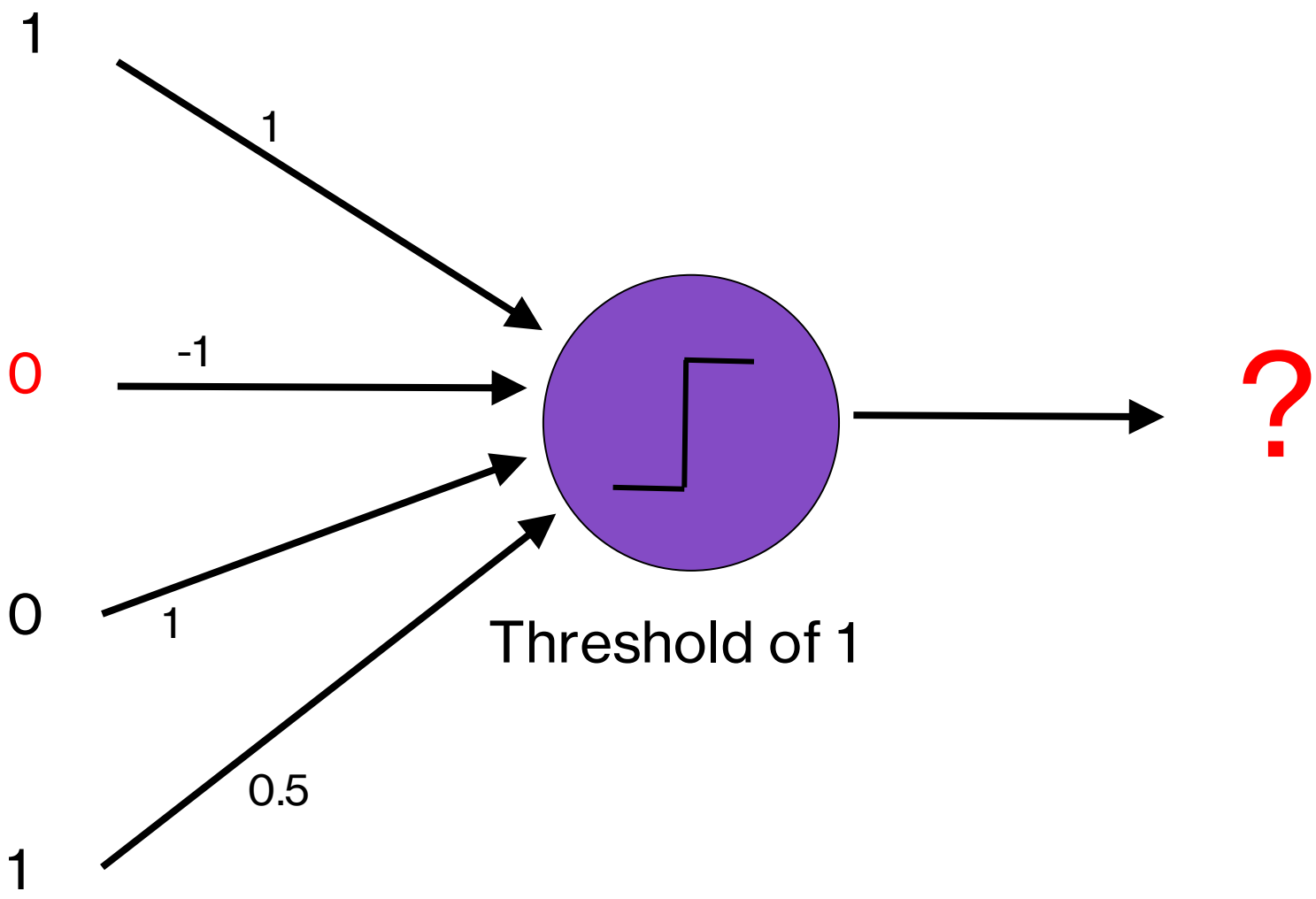


A single neuron/perceptron

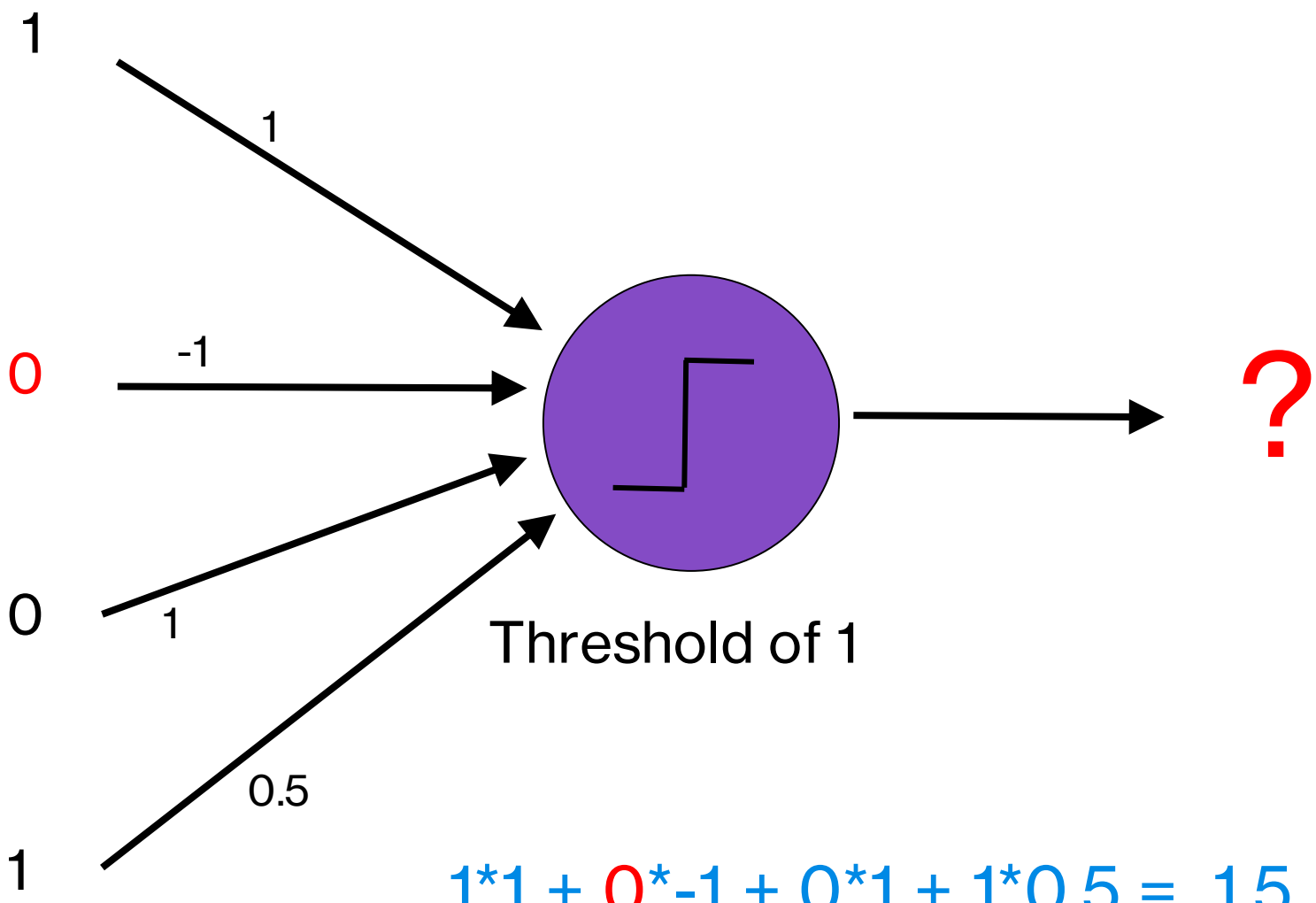


Weighted sum is 0.5, which is not larger than the threshold

A single neuron/perceptron

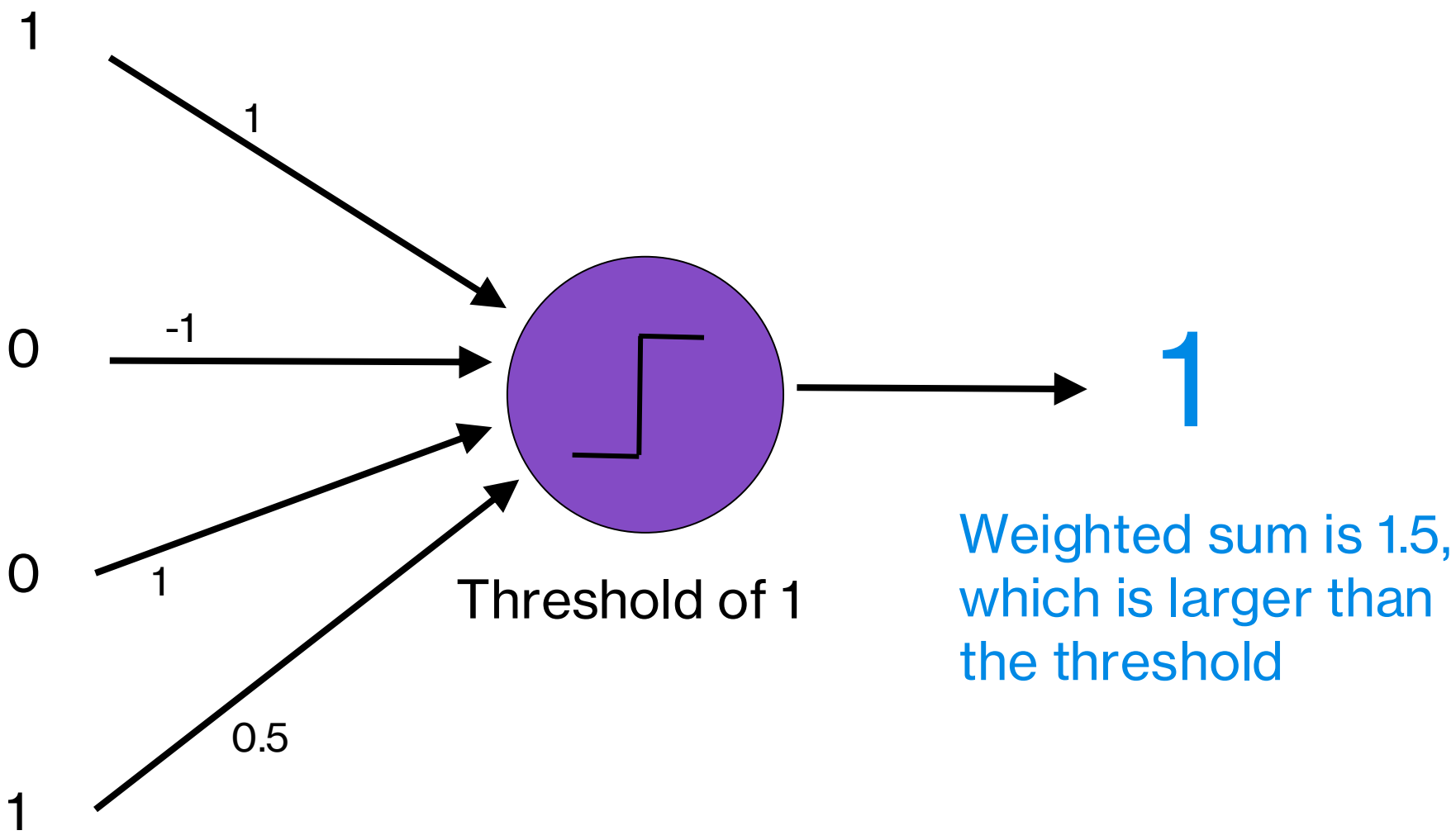


A single neuron/perceptron



1*1 + 0*-1 + 0*1 + 1*0.5 = 1.5

A single neuron/perceptron

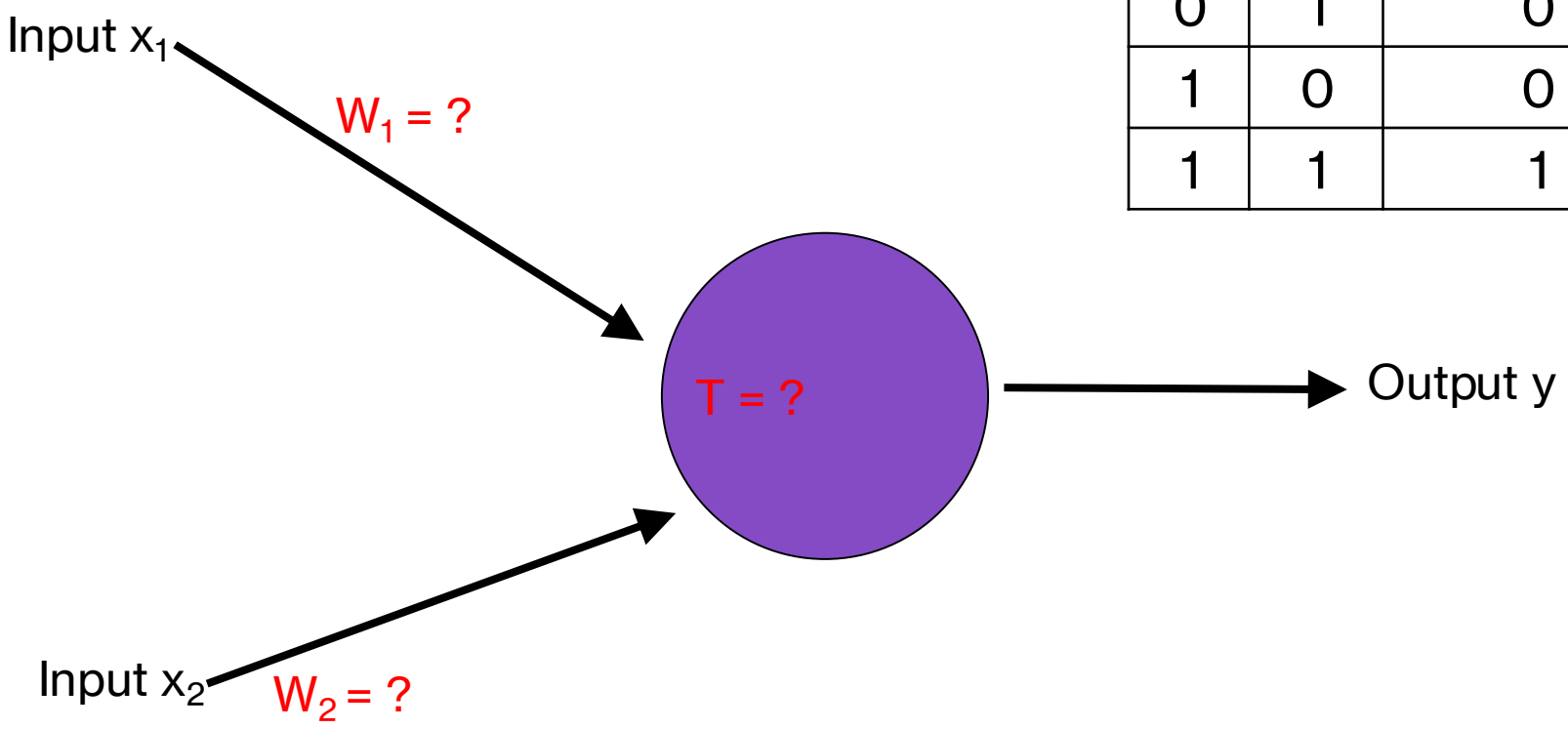


AND logical operator

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

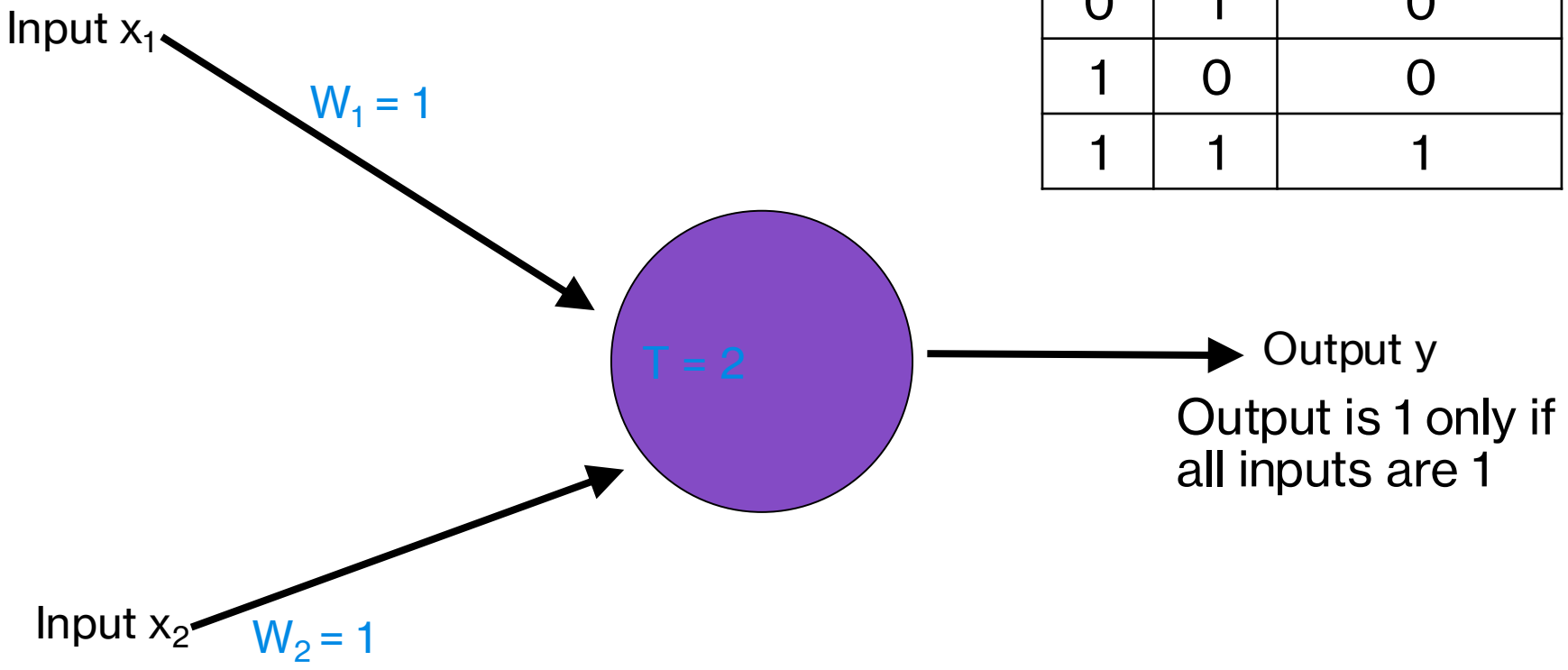
AND logical operator

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1



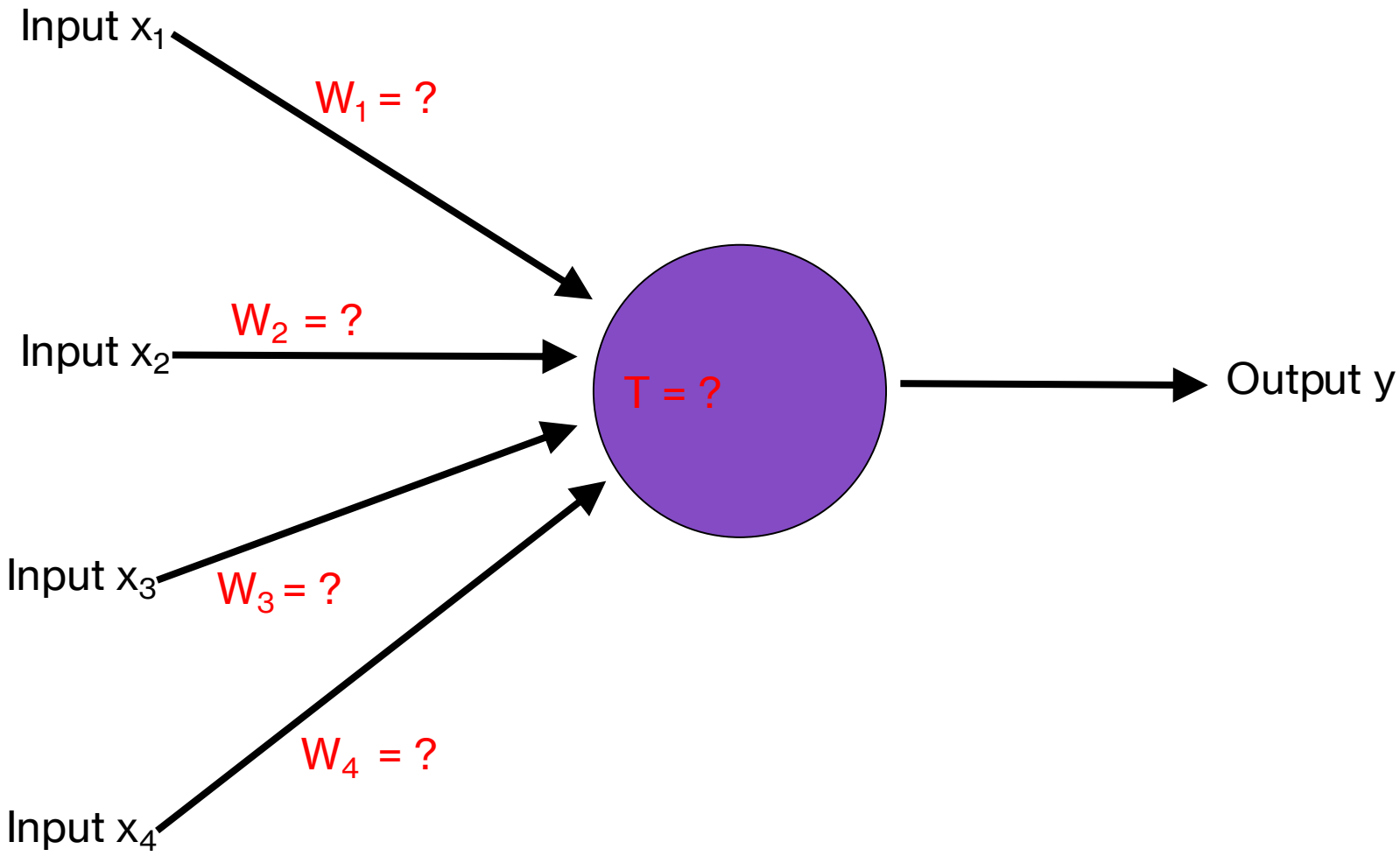
AND logical operator

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

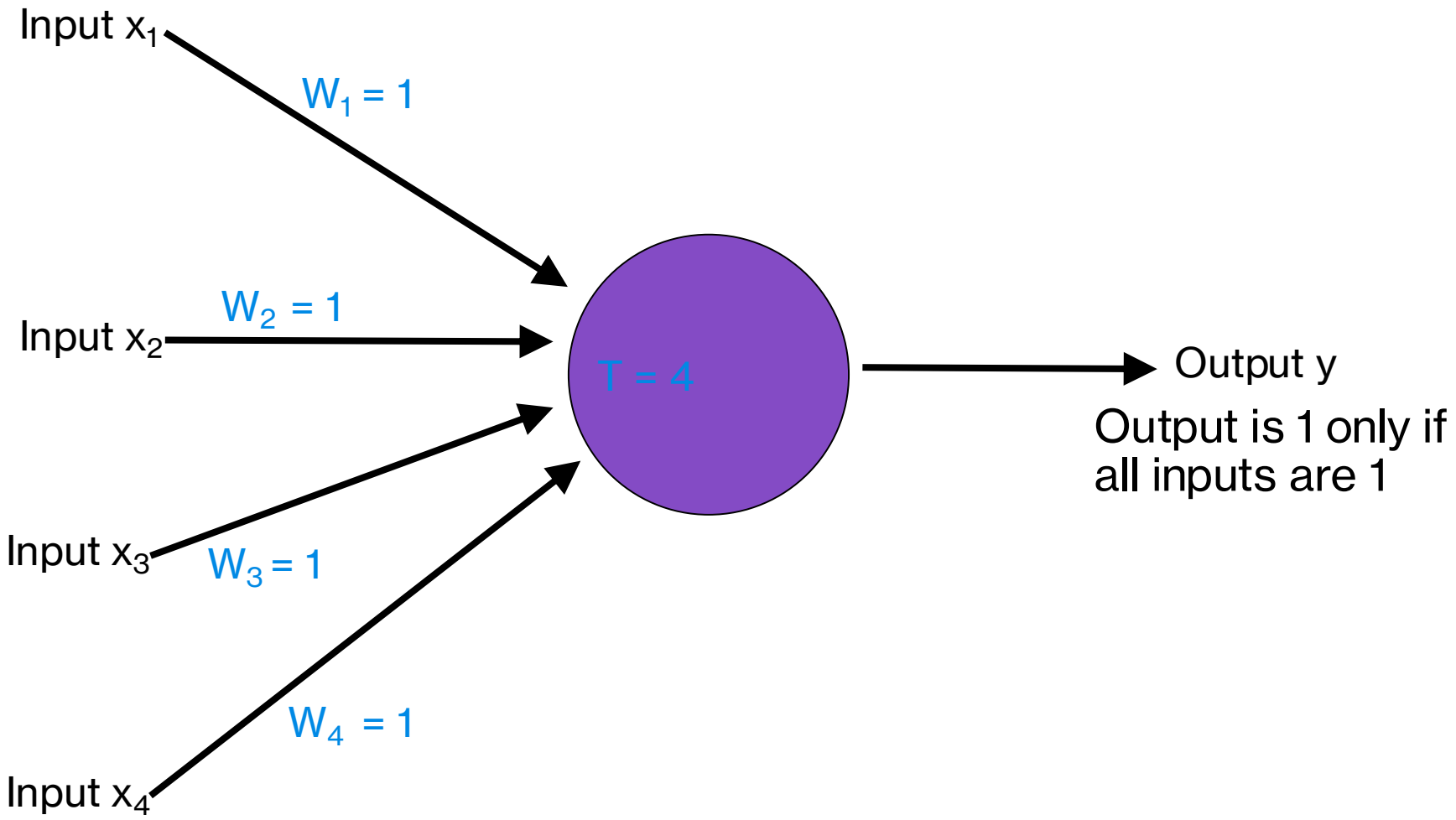


Inputs are either 0 or 1

AND logical operator



AND logical operator



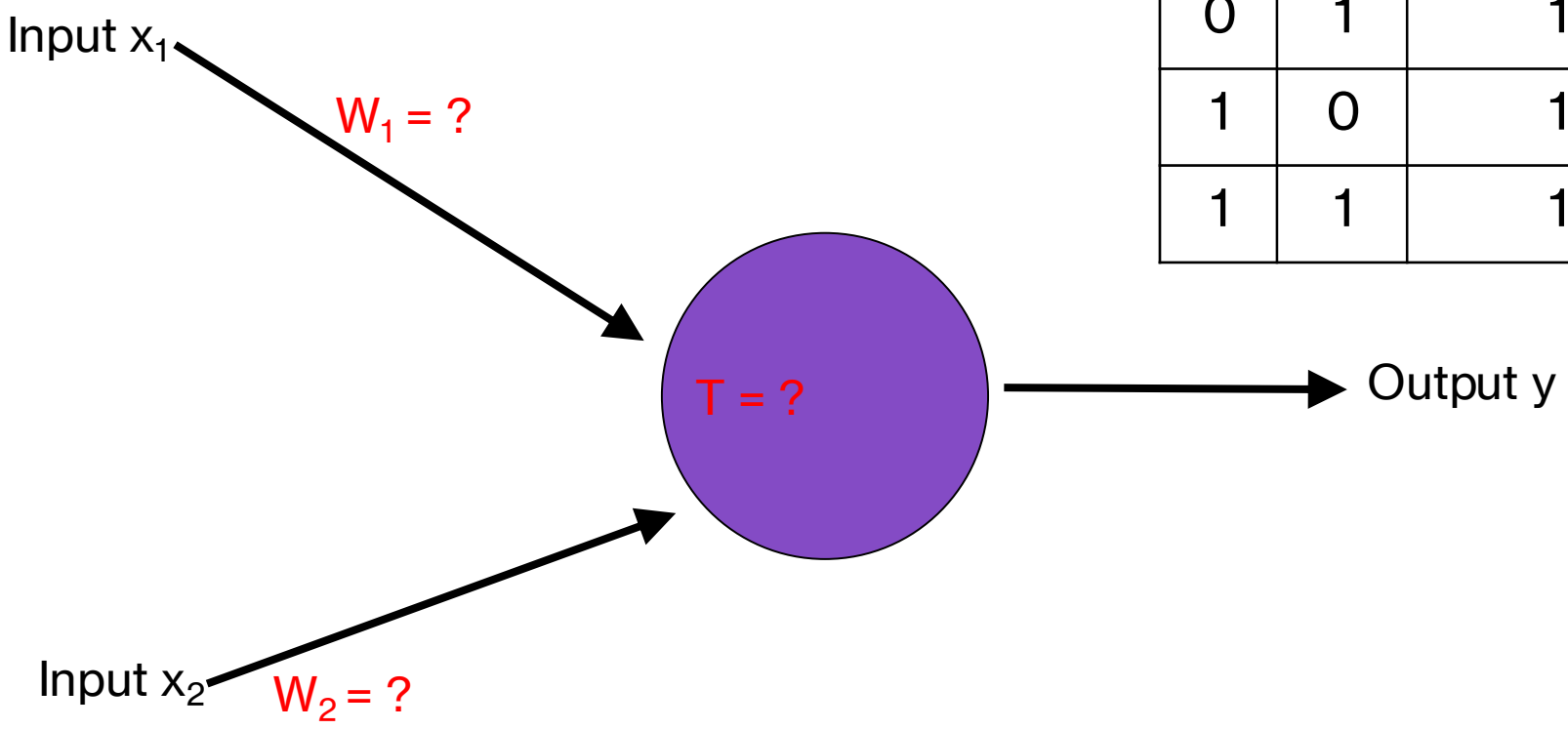
Inputs are either 0 or 1

OR logical operator

x_1	x_2	x_1 or x_2
0	0	0
0	1	1
1	0	1
1	1	1

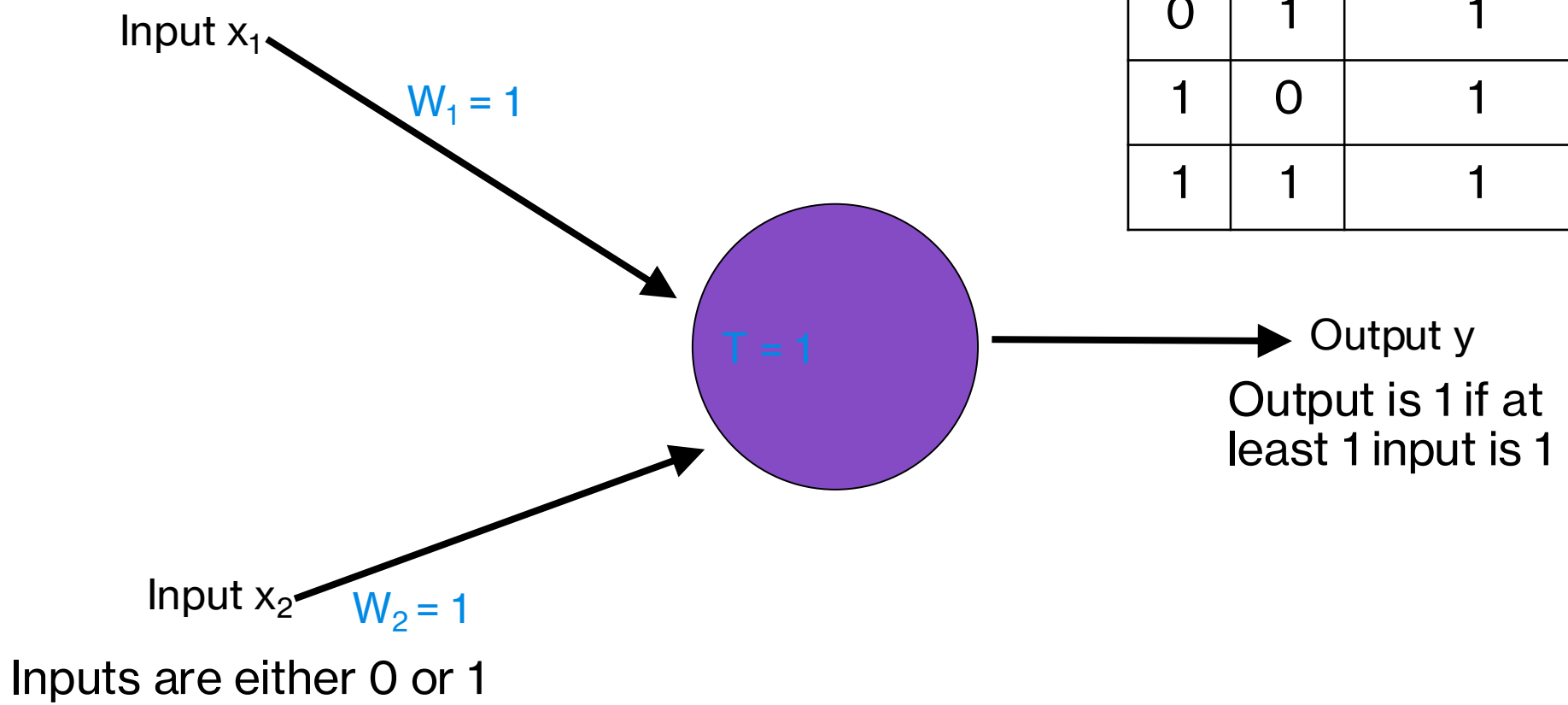
OR logical operator

x_1	x_2	x_1 or x_2
0	0	0
0	1	1
1	0	1
1	1	1

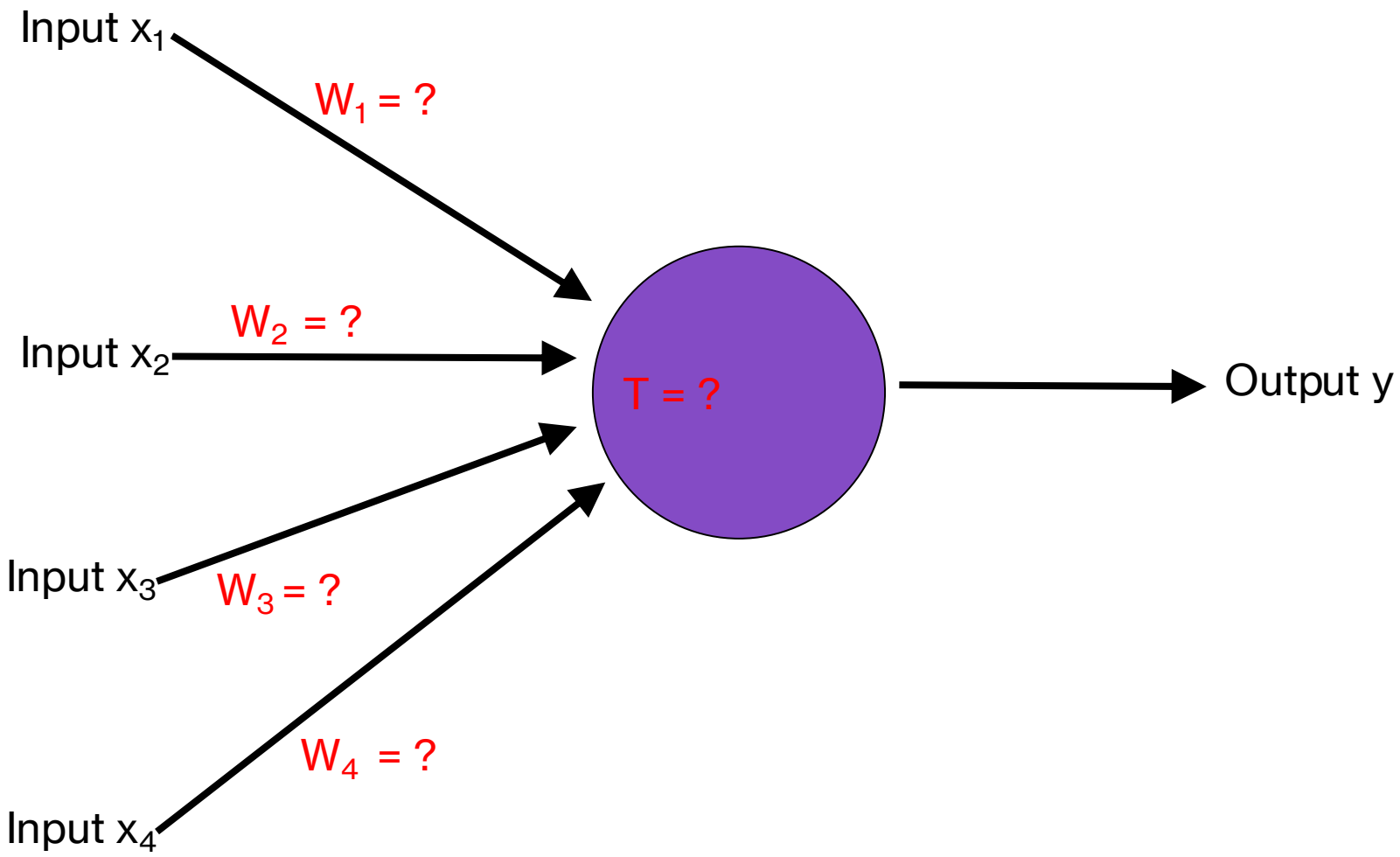


OR logical operator

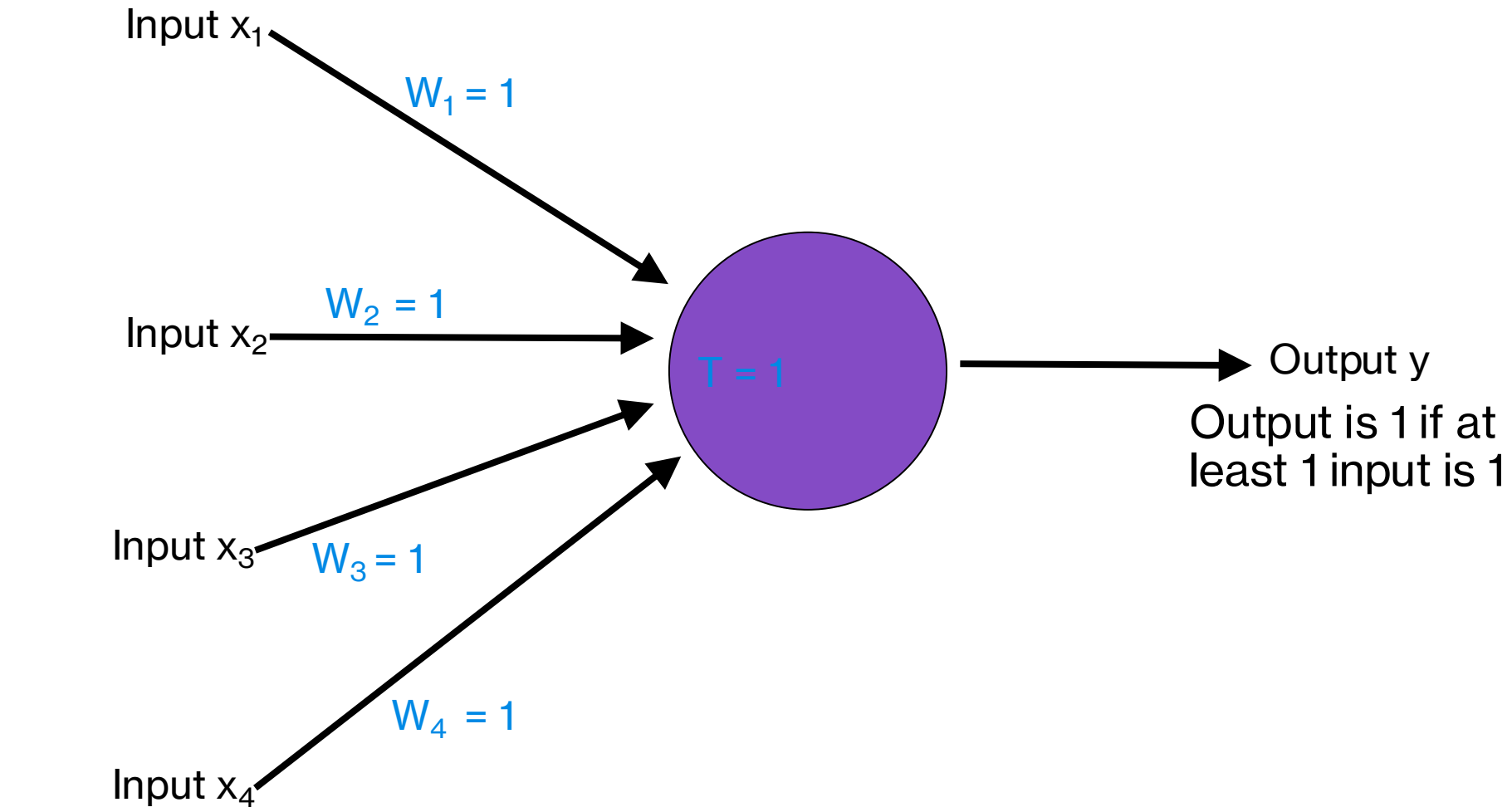
x_1	x_2	x_1 or x_2
0	0	0
0	1	1
1	0	1
1	1	1



OR logical operator



OR logical operator

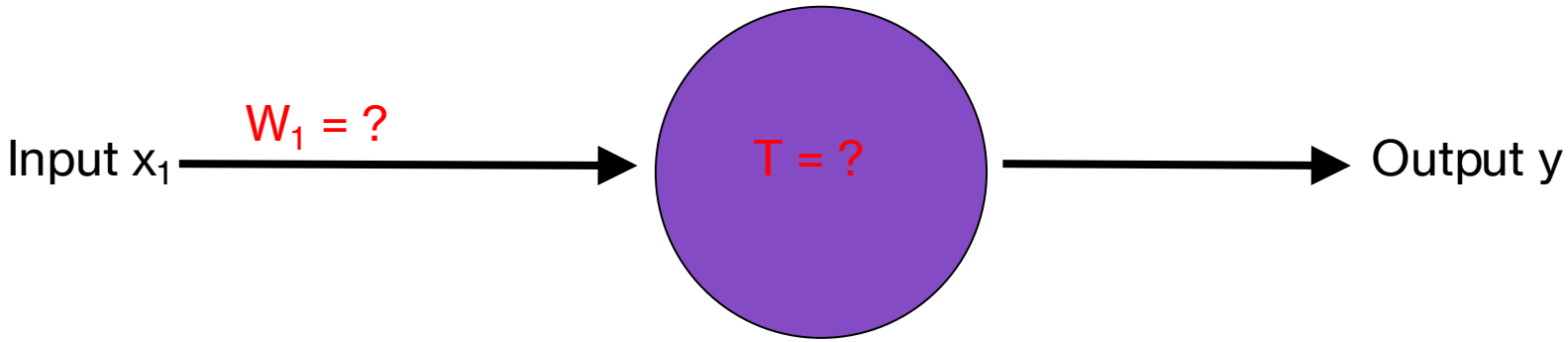


NOT logical operator

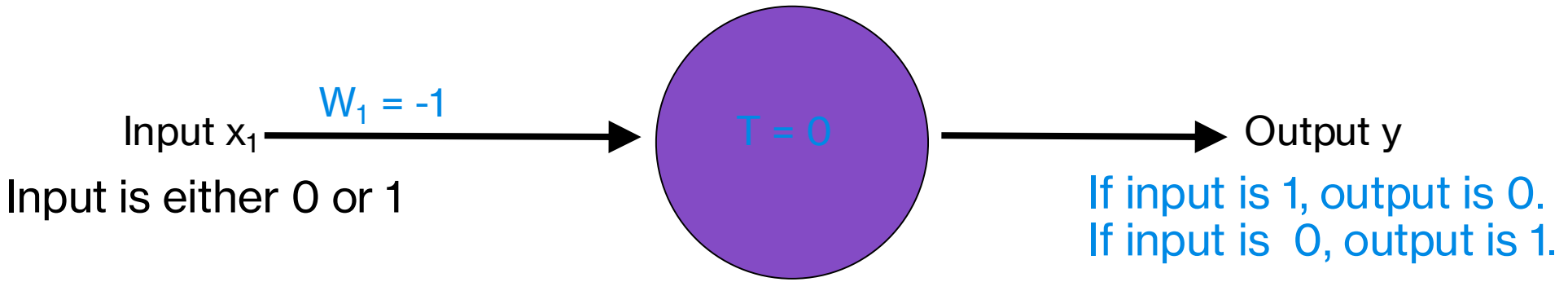
x_1	not x_1
0	1
1	0

NOT logical operator

x_1	not x_1
0	1
1	0

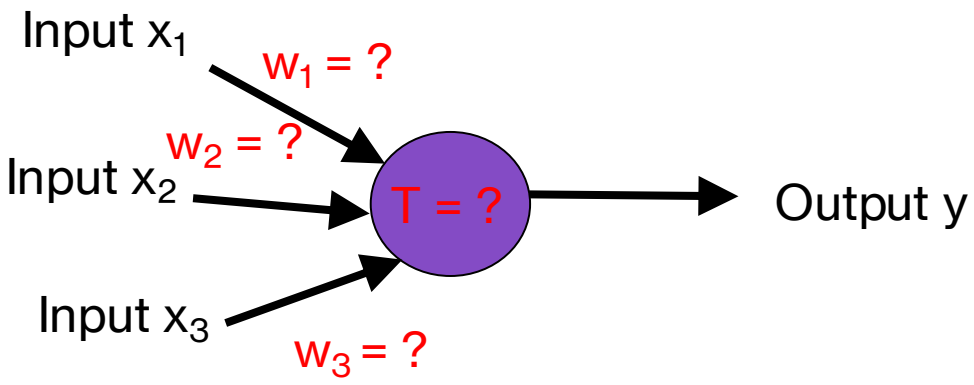


NOT logical operator

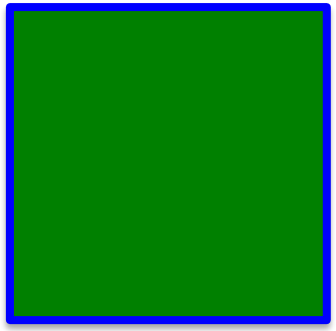


How about...

x_1	x_2	x_3	y
0	0	0	1
0	1	0	0
1	0	0	1
1	1	0	0
0	0	1	1
0	1	1	1
1	0	1	1
1	1	1	0

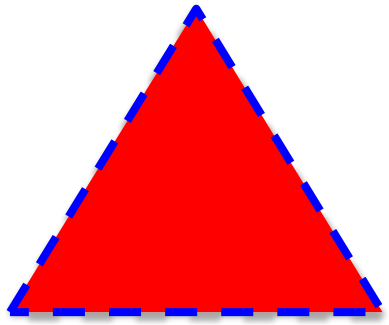


Positive or negative?



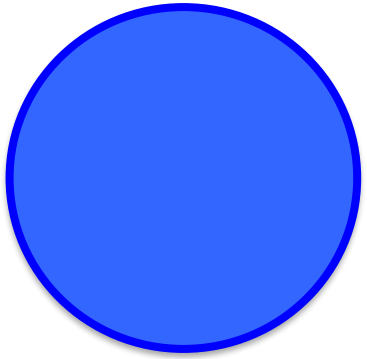
NEGATIVE

Positive or negative?



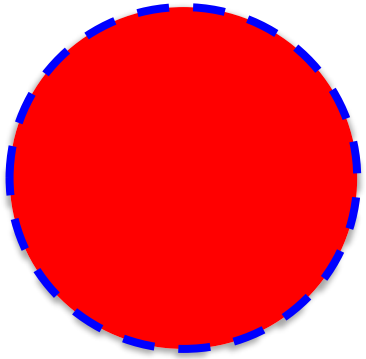
NEGATIVE

Positive or negative?



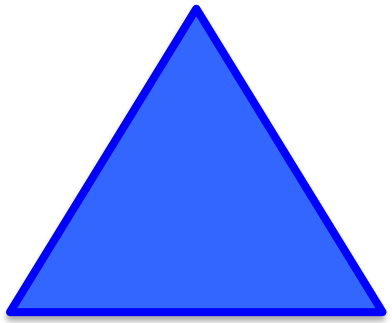
POSITIVE

Positive or negative?



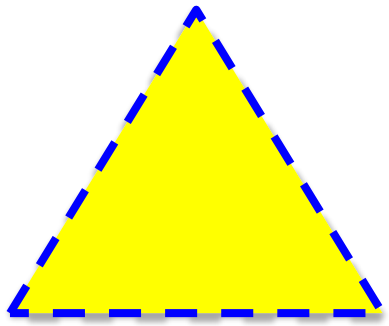
NEGATIVE

Positive or negative?



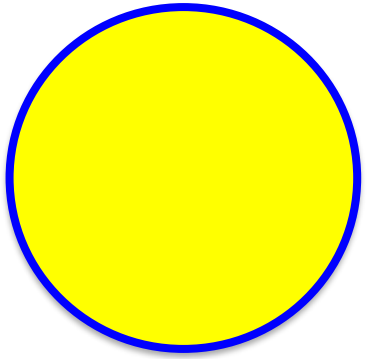
POSITIVE

Positive or negative?



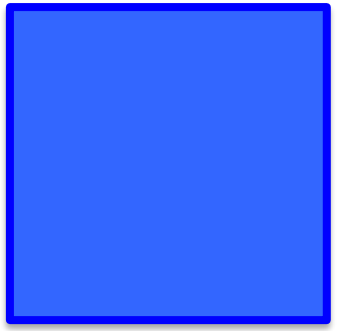
POSITIVE

Positive or negative?



NEGATIVE

Positive or negative?



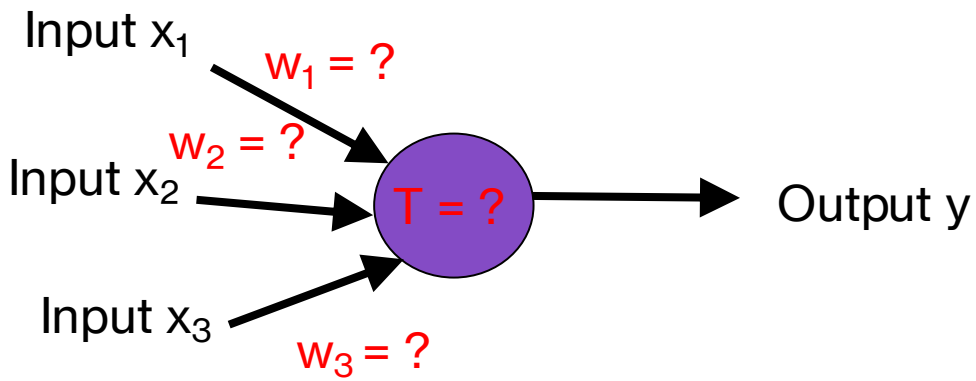
POSITIVE

A method to the madness...

- blue = positive
- yellow triangles = positive
- all others negative

Training neural networks

x_1	x_2	x_3	y
0	0	0	1
0	1	0	0
1	0	0	1
1	1	0	0
0	0	1	1
0	1	1	1
1	0	1	1
1	1	1	0

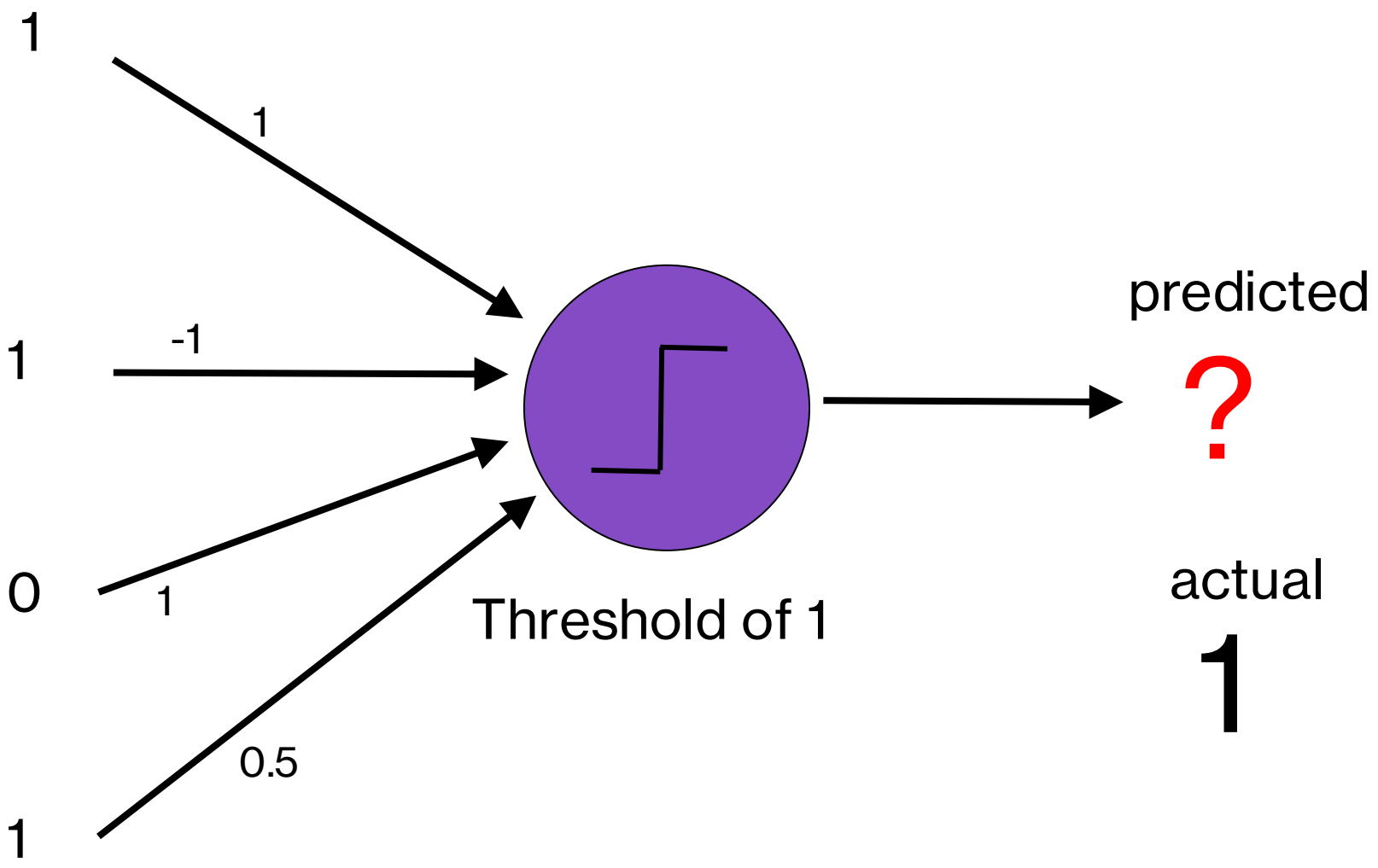


1. start with some initial weights and thresholds
2. show examples repeatedly to NN
3. update weights/thresholds by comparing NN output to actual output

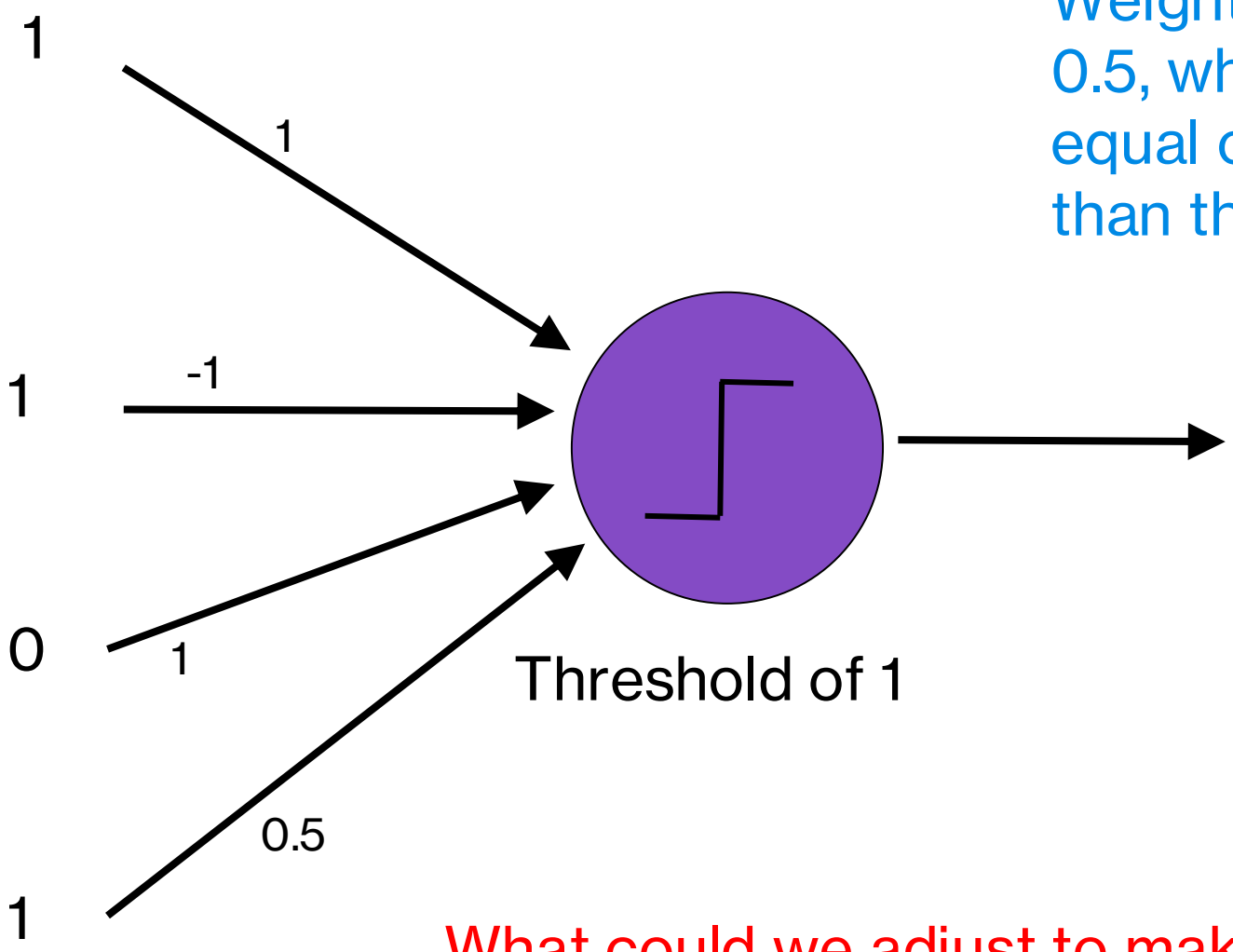
Perceptron learning algorithm

- repeat until you get all examples right:
- for each “training” example:
  - calculate current prediction on example
  - if wrong:
    - update weights and threshold towards getting this example correct

Perceptron learning algorithm



Perceptron learning algorithm



Weighted sum is 0.5, which is not equal or larger than the threshold

predicted

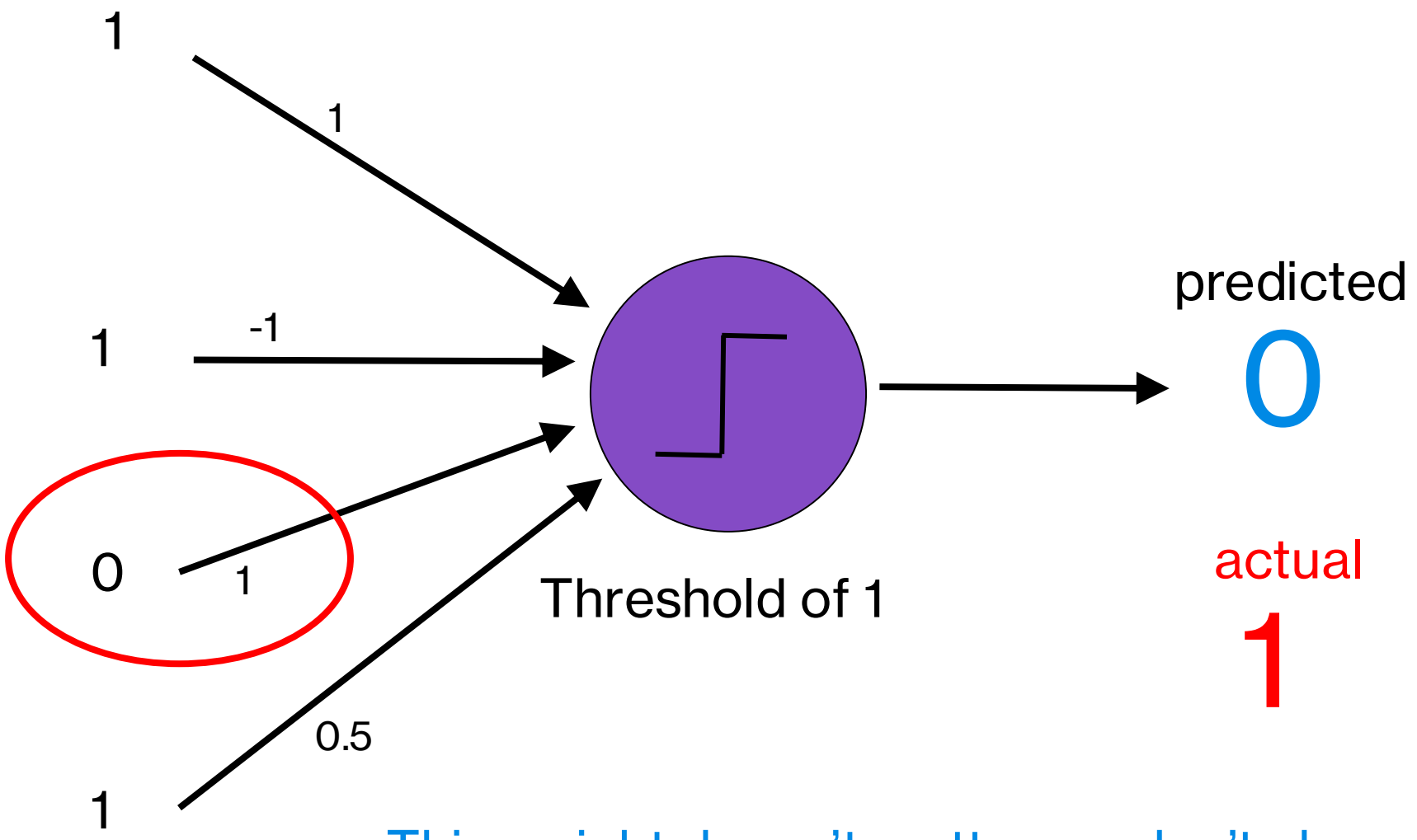
0

actual

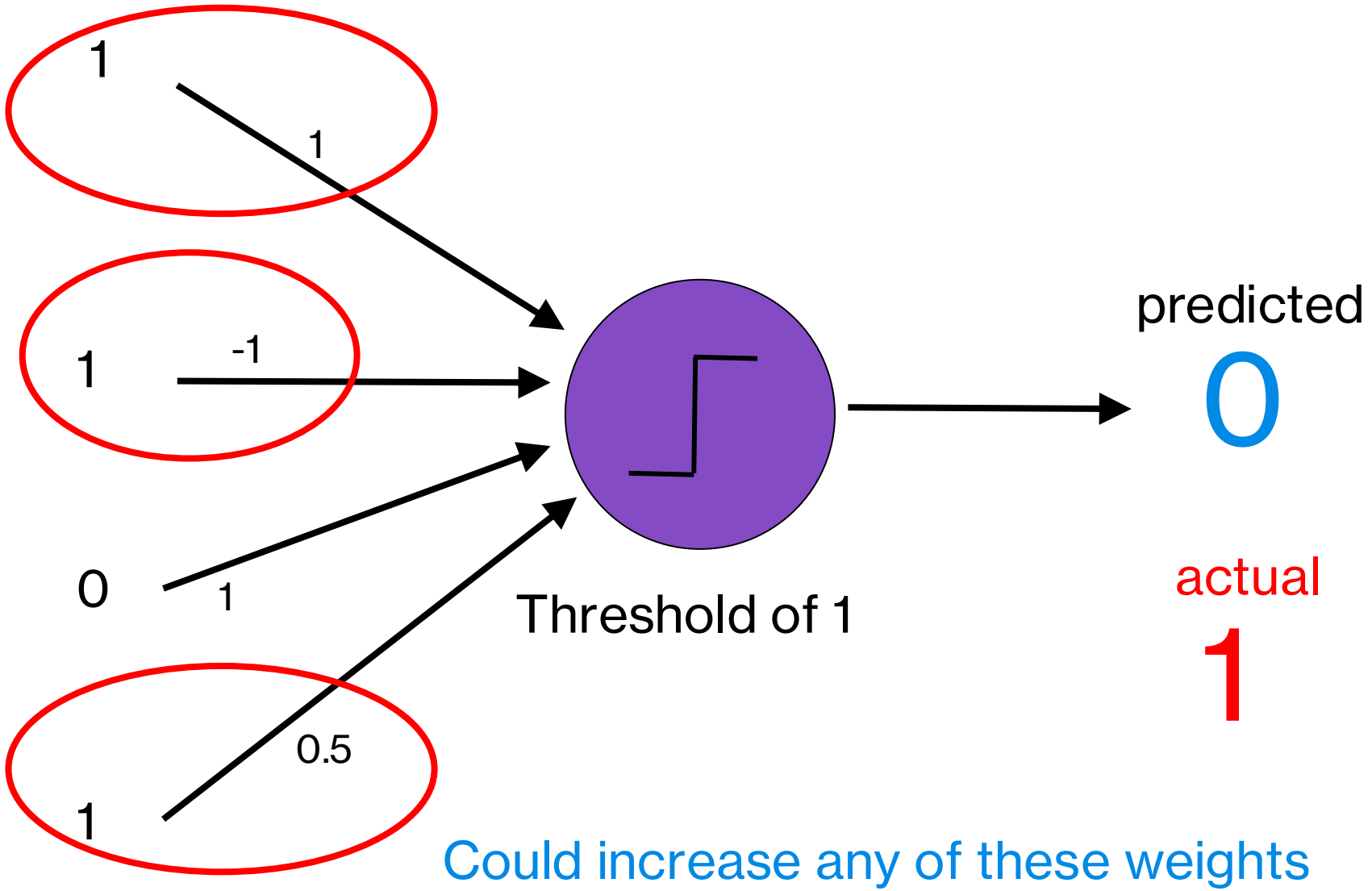
1

What could we adjust to make it right?

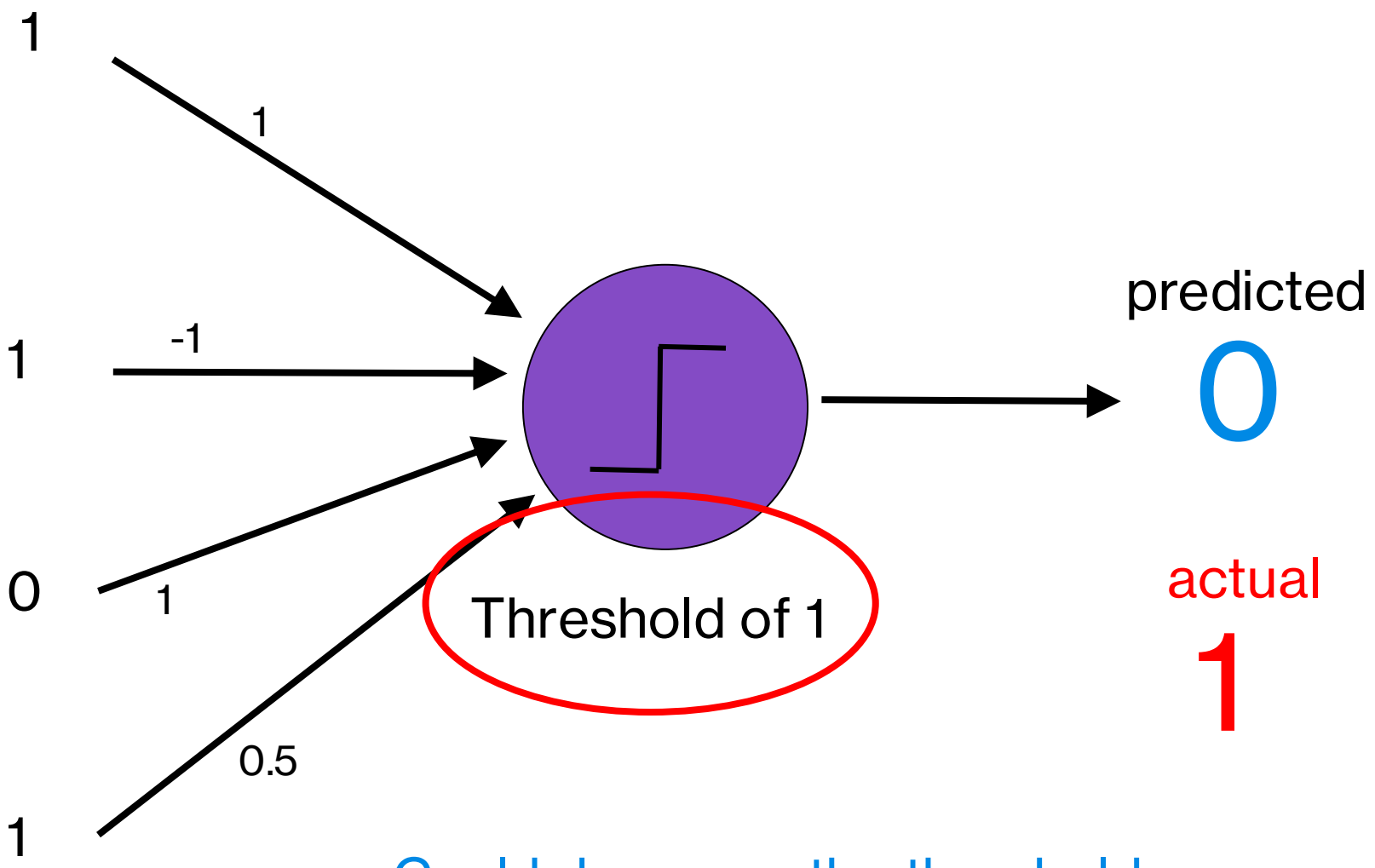
Perceptron learning algorithm



Perceptron learning algorithm



Perceptron learning algorithm

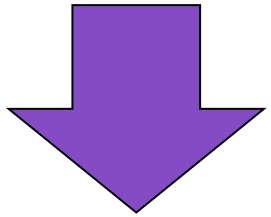


Could decrease the threshold

Perceptron update rule

if wrong:

update weights and threshold towards getting this example correct

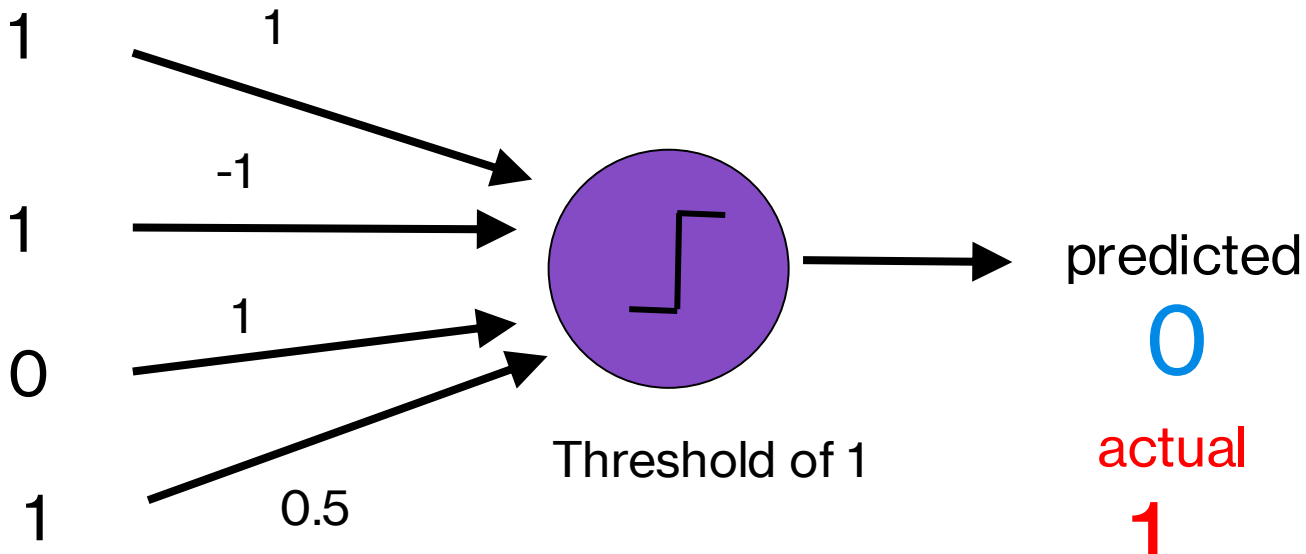


if wrong:

$$w_i = w_i + \Delta w_i$$

$$\Delta w_i = \lambda * (\text{actual} - \text{predicted}) * x_i$$

Perceptron learning algorithm

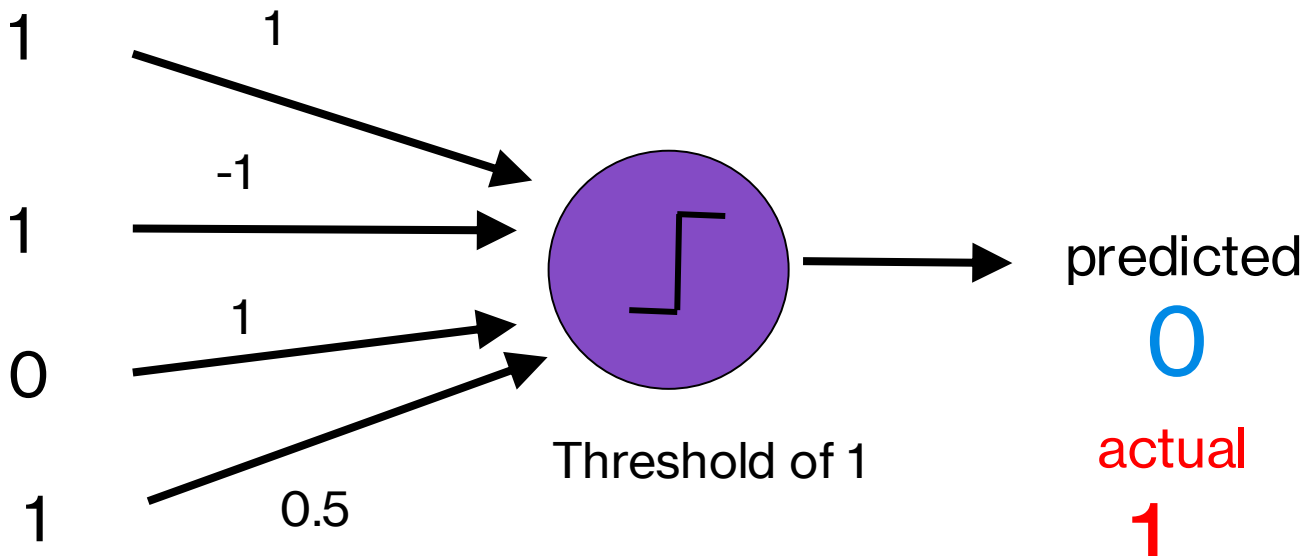


$$w_i = w_i + \Delta wi$$

$$\Delta w_i = \lambda * (\text{actual} - \text{predicted}) * x_i$$

What does this do in this case?

Perceptron learning algorithm

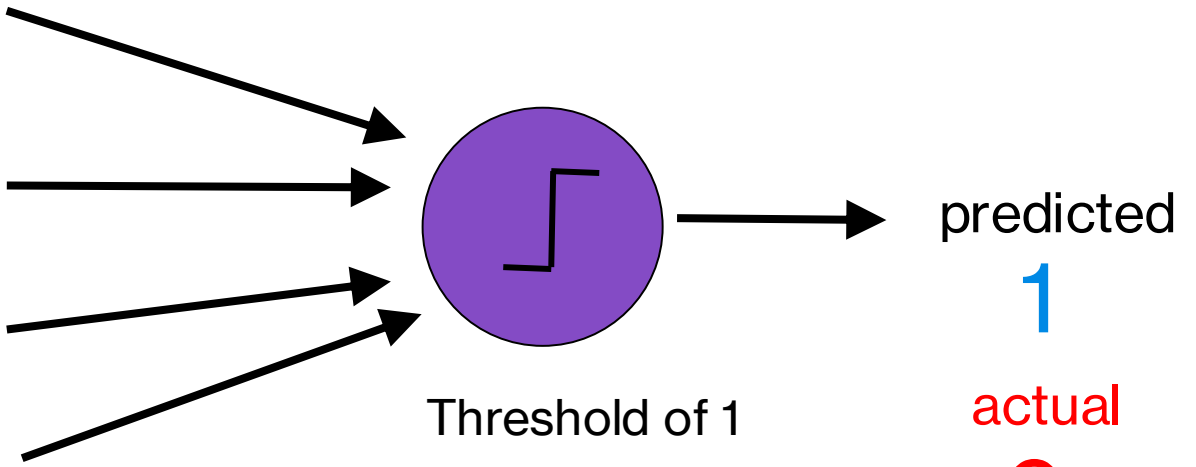


$$w_i = w_i + \Delta wi$$

$$\Delta w_i = \lambda * (\text{actual} - \text{predicted}) * x_i$$

causes us to increase the weights!

Perceptron learning algorithm

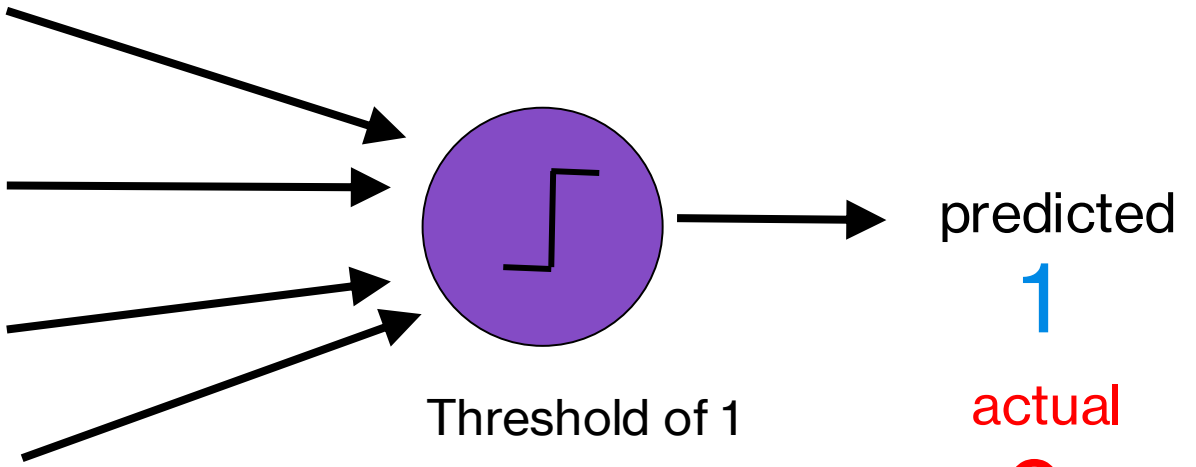


$$w_i = w_i + \Delta wi$$

$$\Delta w_i = \lambda * (\text{actual} - \text{predicted}) * x_i$$

What if predicted = 1 and actual = 0?

Perceptron learning algorithm

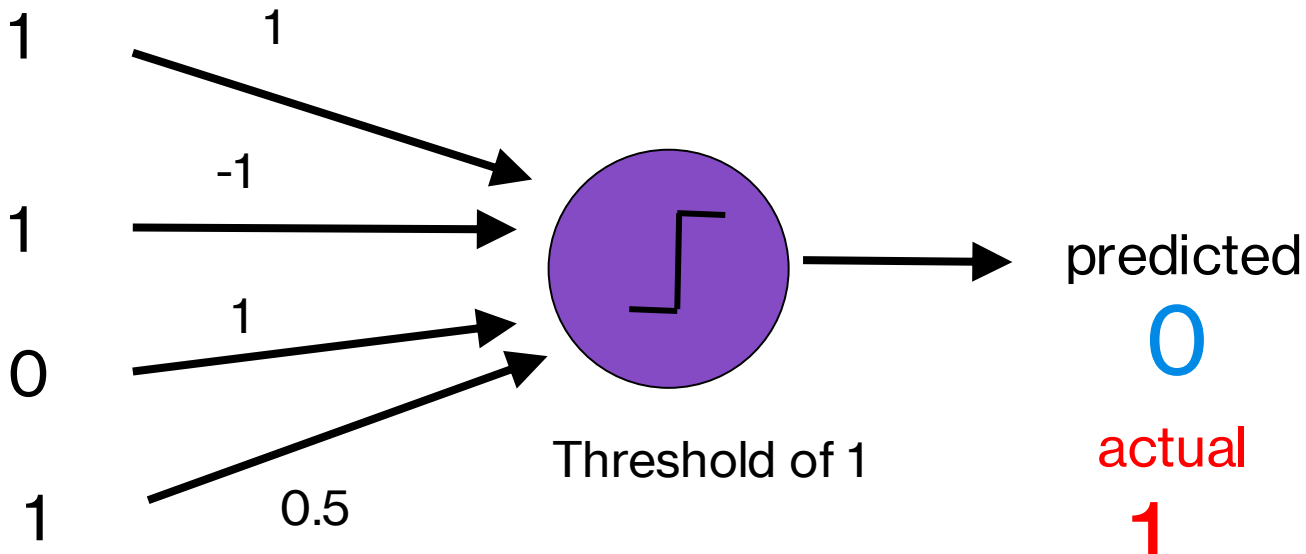


$$w_i = w_i + \Delta wi$$

$$\Delta w_i = \lambda * (\text{actual} - \text{predicted}) * x_i$$

We're over the threshold, so want to decrease weights:
actual - predicted = -1

Perceptron learning algorithm

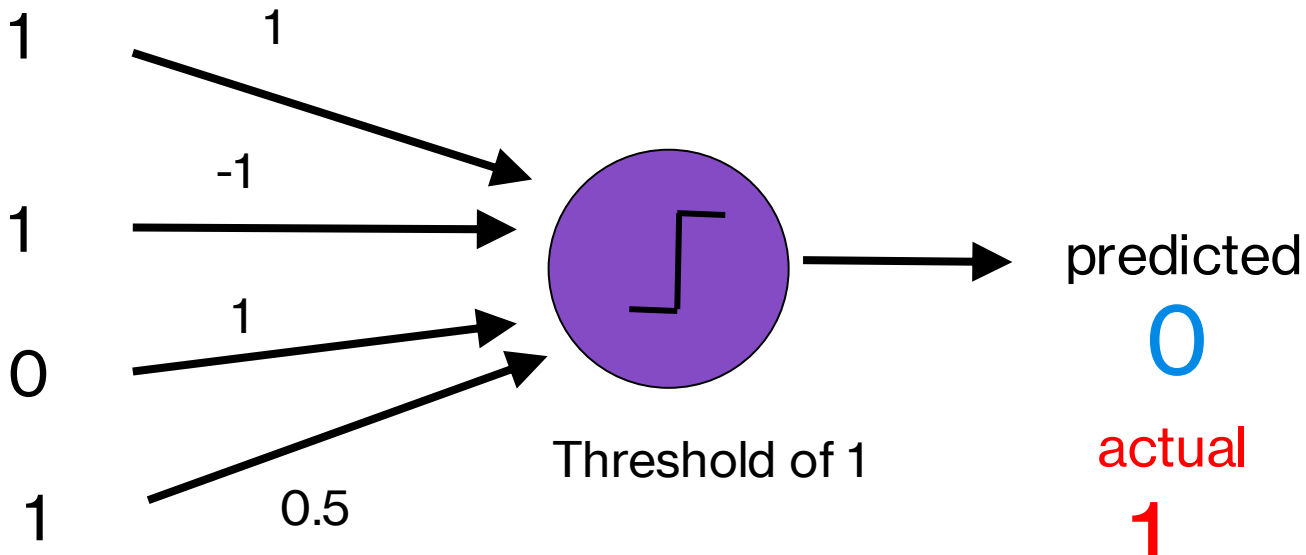


$$w_i = w_i + \Delta wi$$

$$\Delta w_i = \lambda * (\text{actual} - \text{predicted}) * x_i$$

What does this do?

Perceptron learning algorithm

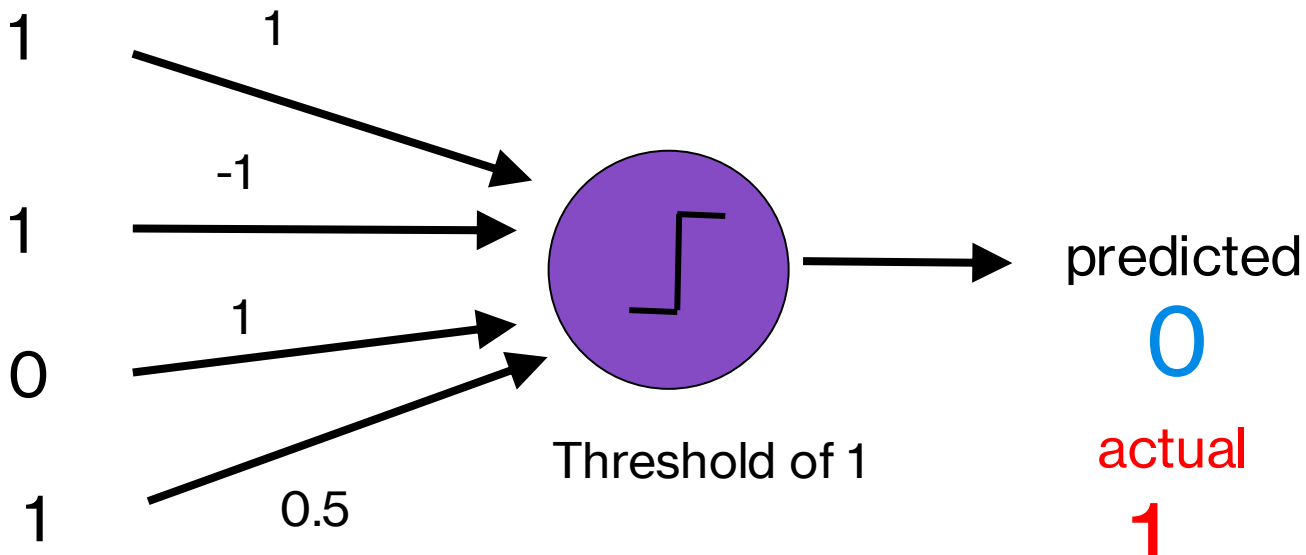


$$w_i = w_i + \Delta wi$$

$$\Delta w_i = \lambda * (\text{actual} - \text{predicted}) * x_i$$

Only adjust those weights that actually contributed!

Perceptron learning algorithm

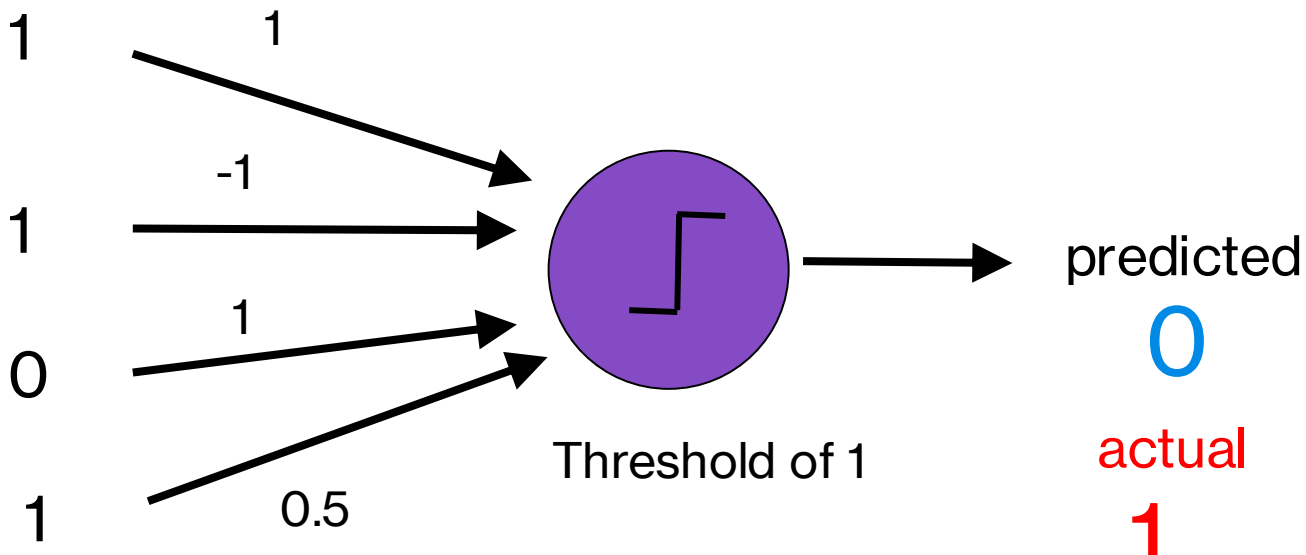


$$w_i = w_i + \Delta wi$$

$$\Delta w_i = \lambda * (\text{actual} - \text{predicted}) * x_i$$

What does this do?

Perceptron learning algorithm

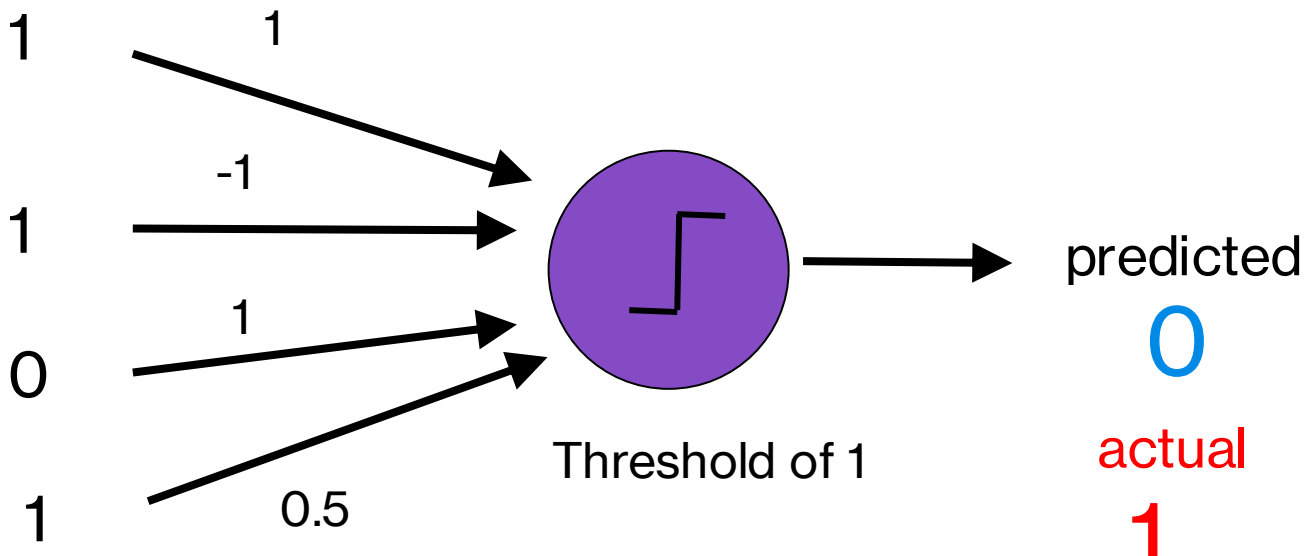


$$w_i = w_i + \Delta wi$$

$$\Delta w_i = \lambda * (\text{actual} - \text{predicted}) * x_i$$

"learning rate": value between 0 and 1 (e.g., 0.1)
adjusts how abrupt the changes are to the model

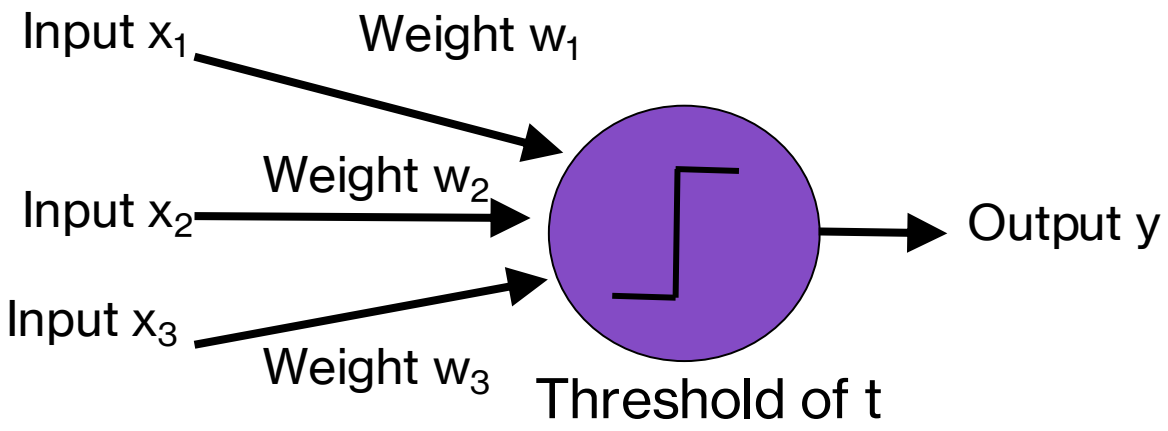
Perceptron learning algorithm



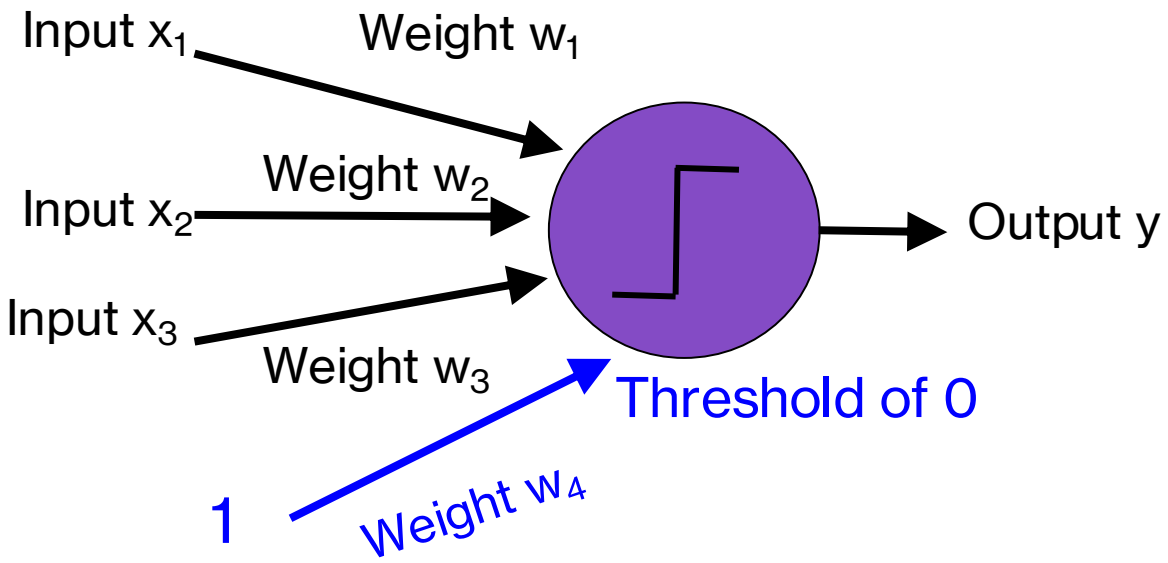
$$w_i = w_i + \Delta wi$$

$$\Delta w_i = \lambda * (\text{actual} - \text{predicted}) * x_i$$

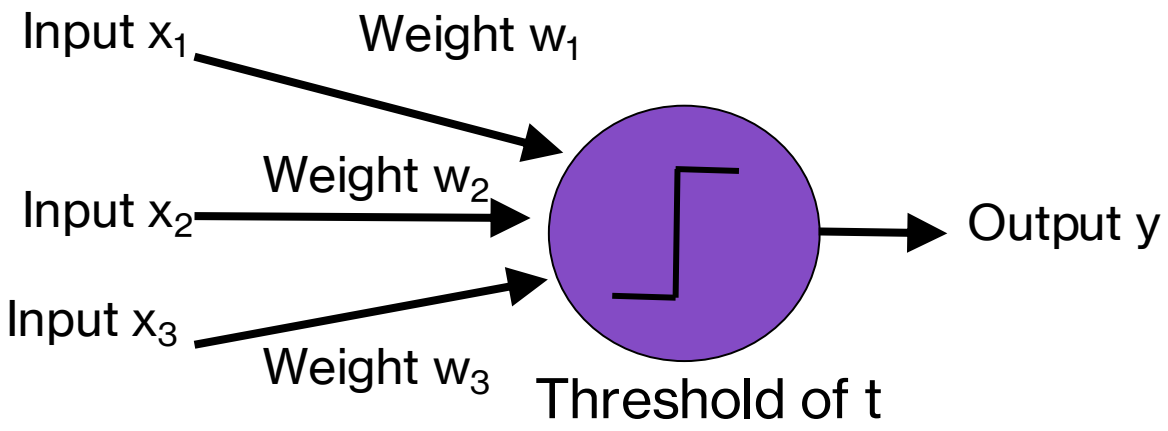
What about the threshold?



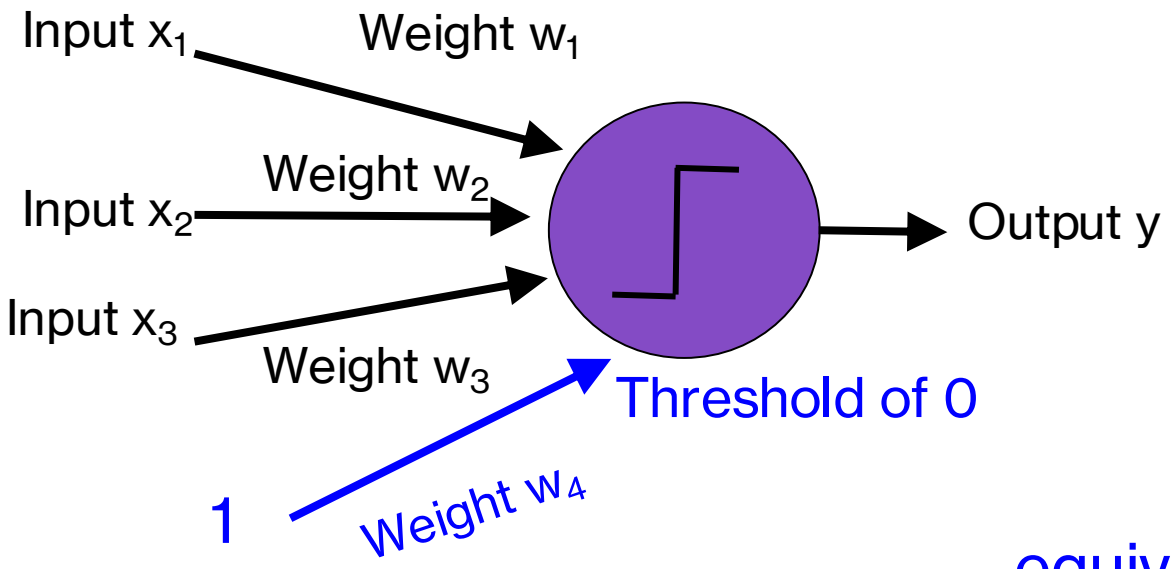
$$1 \text{ if } \sum_{i=1}^{3} w_i x_i \geq t$$



$$1 \text{ if } w_4 + \sum_{i=1}^{3} w_i x_i \geq 0$$



$$1 \text{ if } \sum_{i=1}^{3} w_i x_i \geq t$$



$$1 \text{ if } w_4 + \sum_{i=1}^{3} w_i x_i \geq 0$$

equivalent when $w_4 = -t$

Perceptron learning algorithm

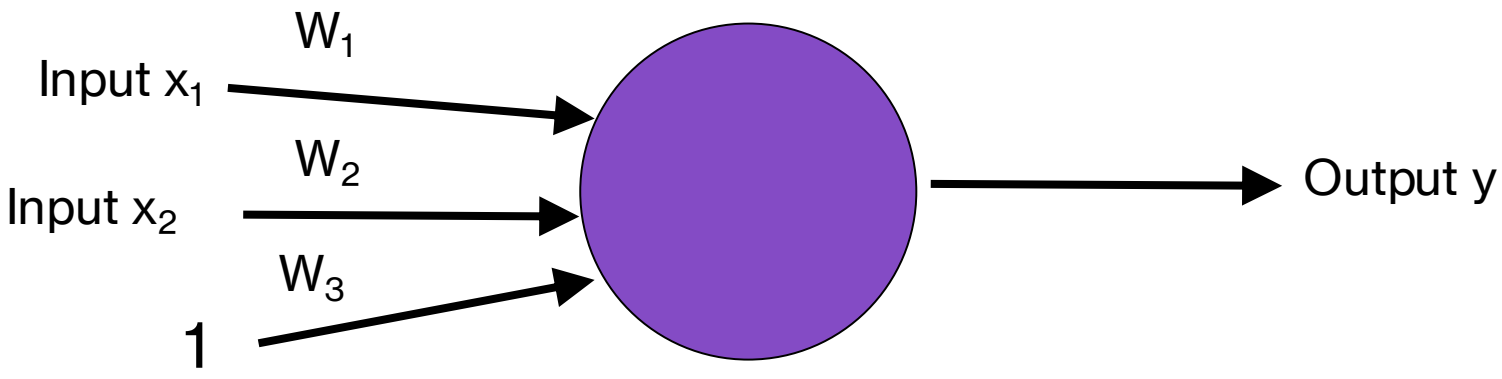
- initialize weights of the model randomly
- repeat until you get all examples right:
- for each “training” example (in a random order):
- calculate current prediction on the example
- if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

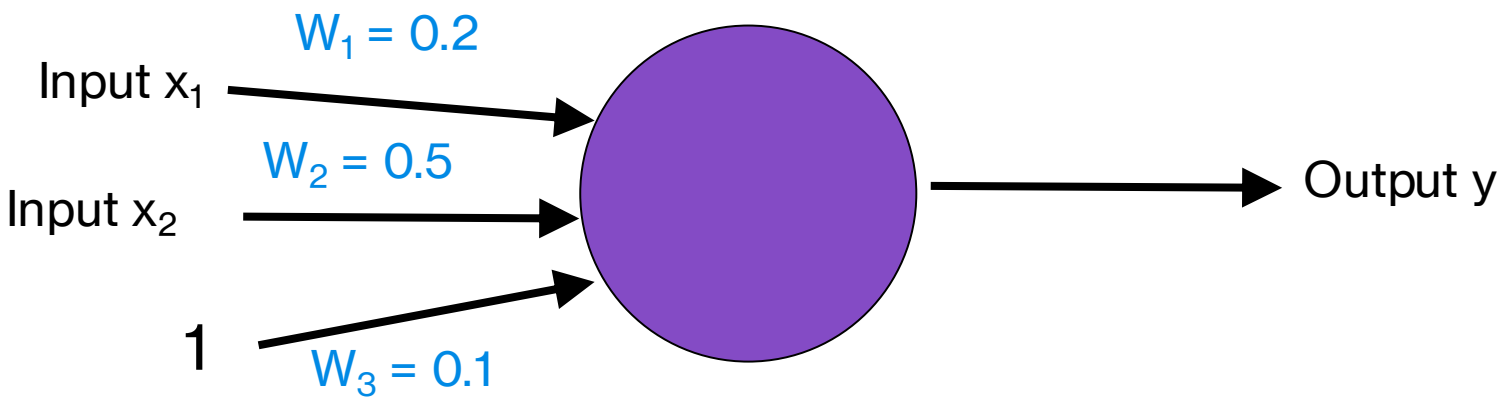
$\lambda = 0.1$

initialize with random weights



x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

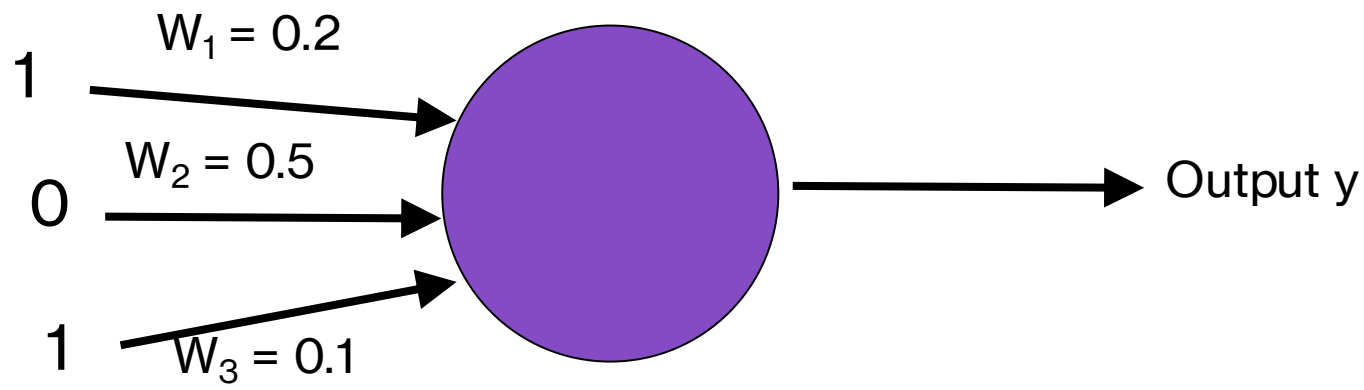


x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



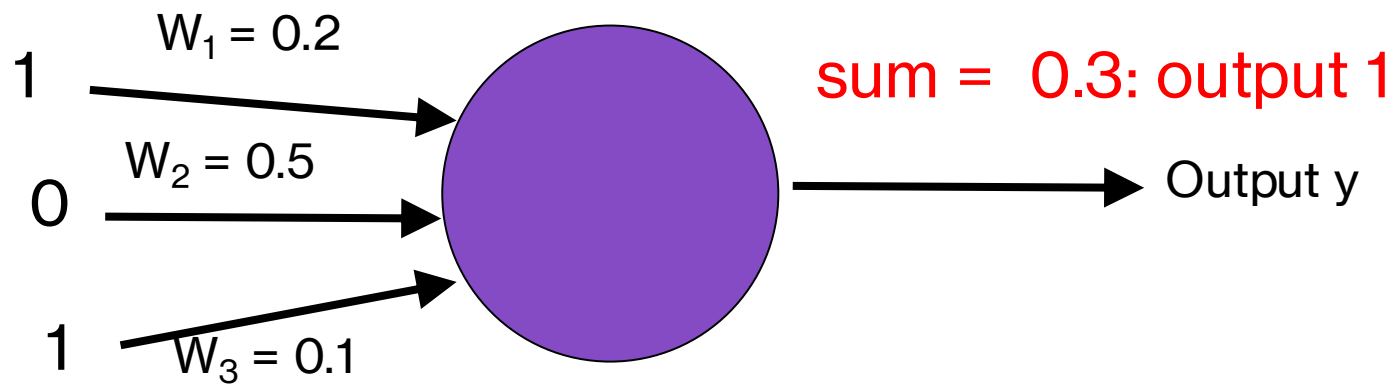
Right or wrong?

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



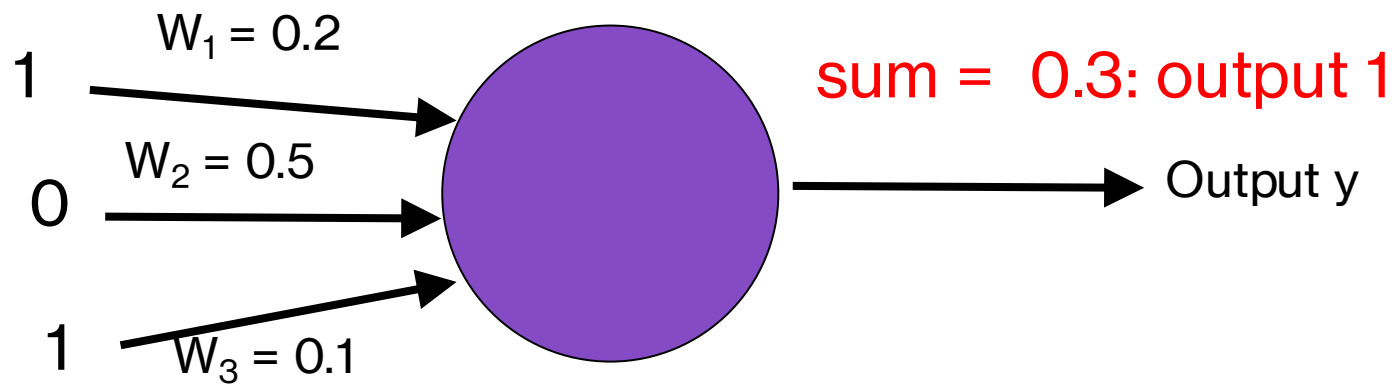
Wrong

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



new weights?

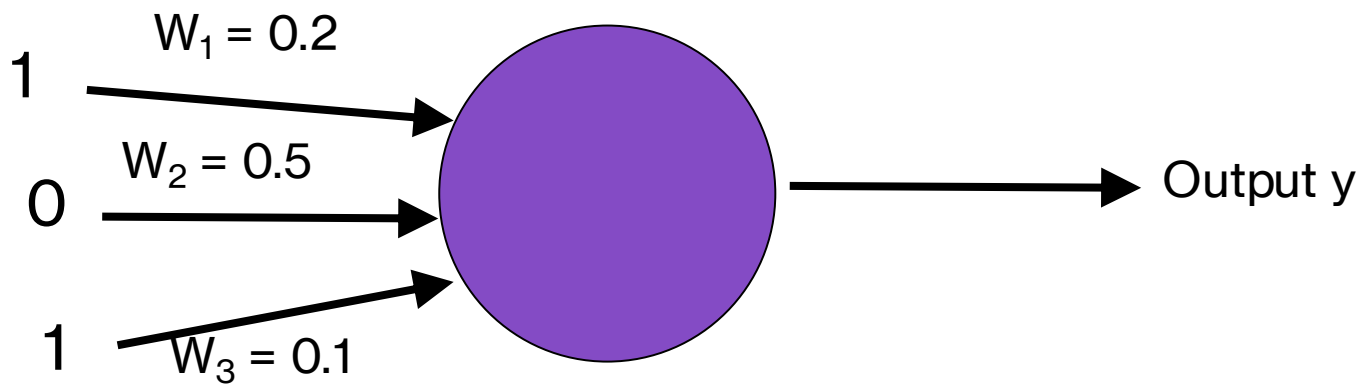
x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$

decrease (0-1=-1) all non-zero x_i by 0.1



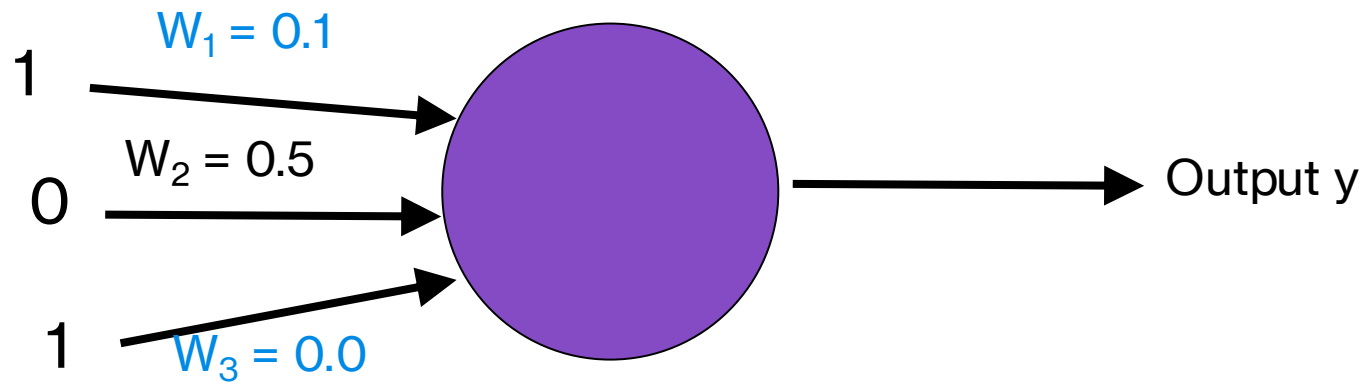
x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$

decrease (0-1=-1) all non-zero x_i by 0.1

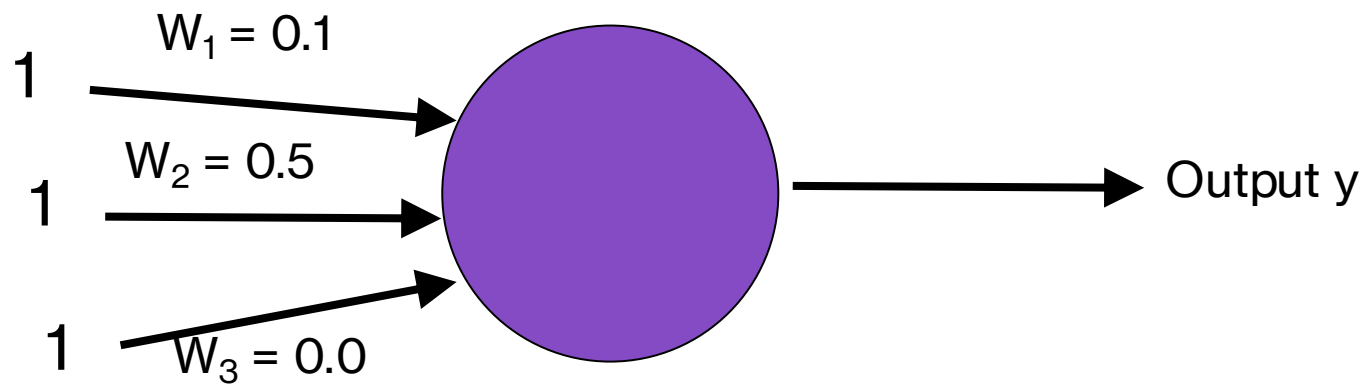


x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



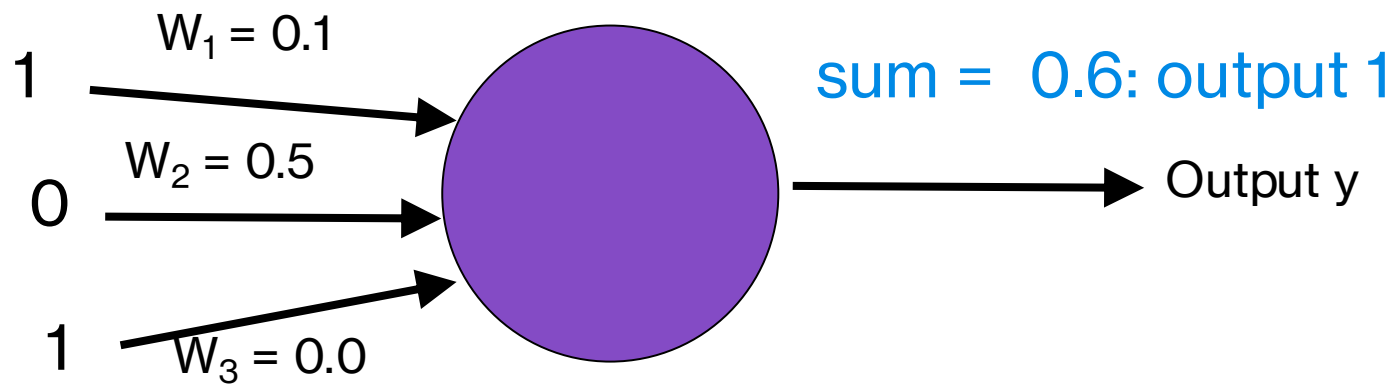
Right or wrong?

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



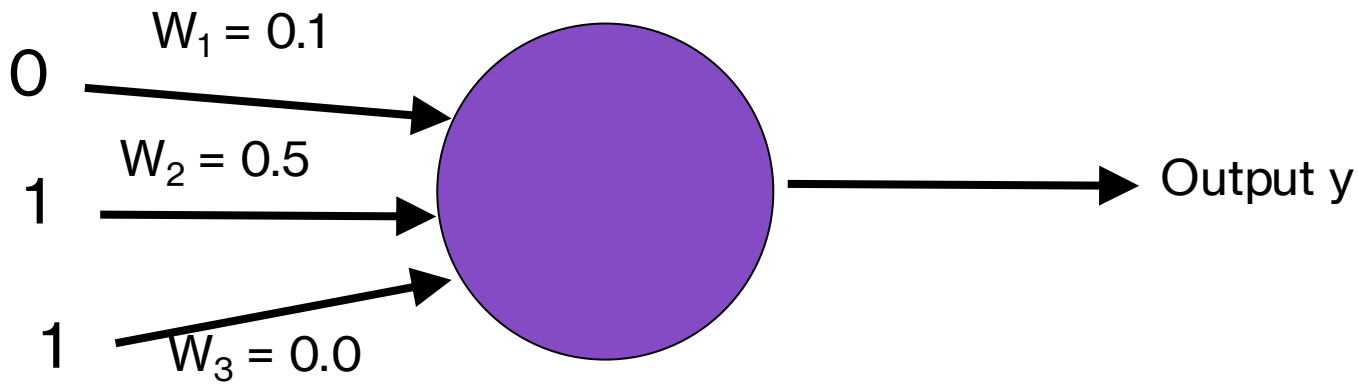
Right. No update!

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



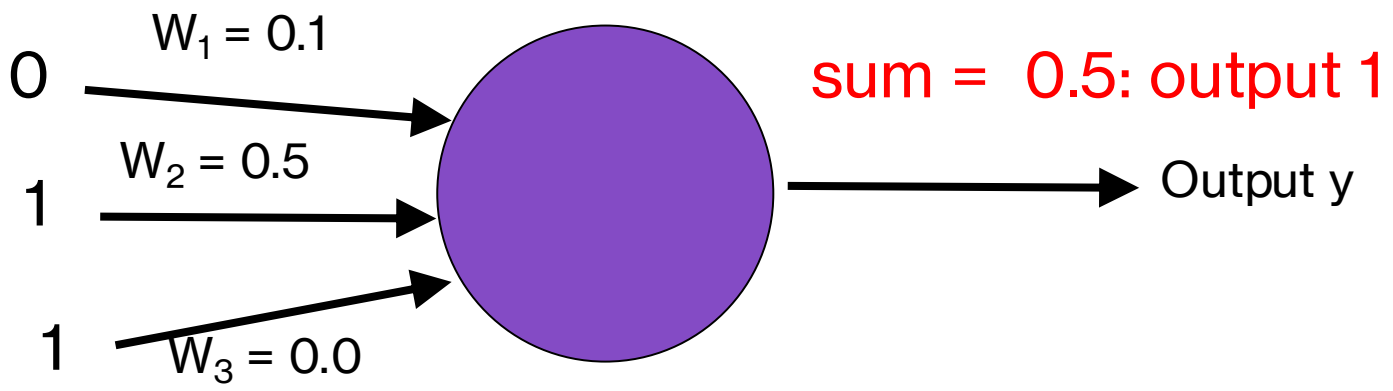
Right or wrong?

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



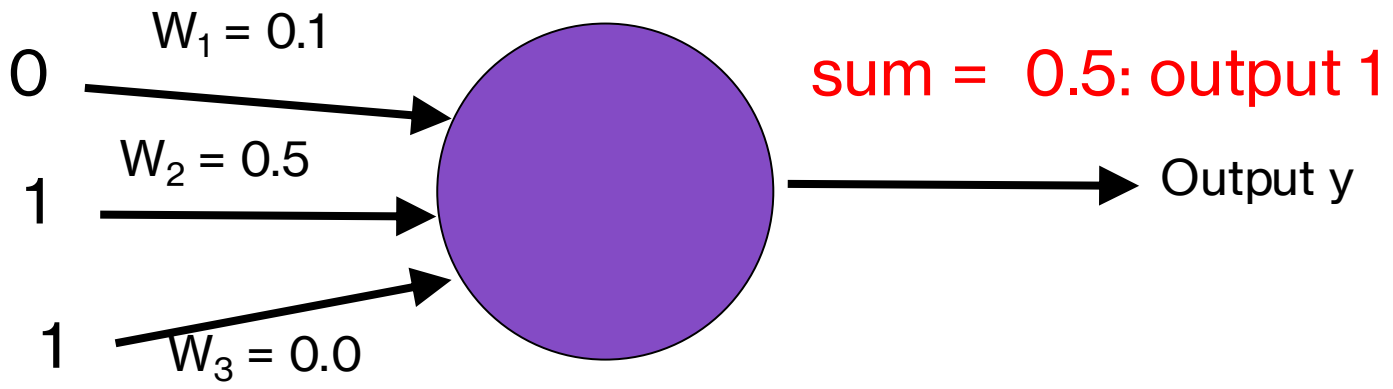
Wrong

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



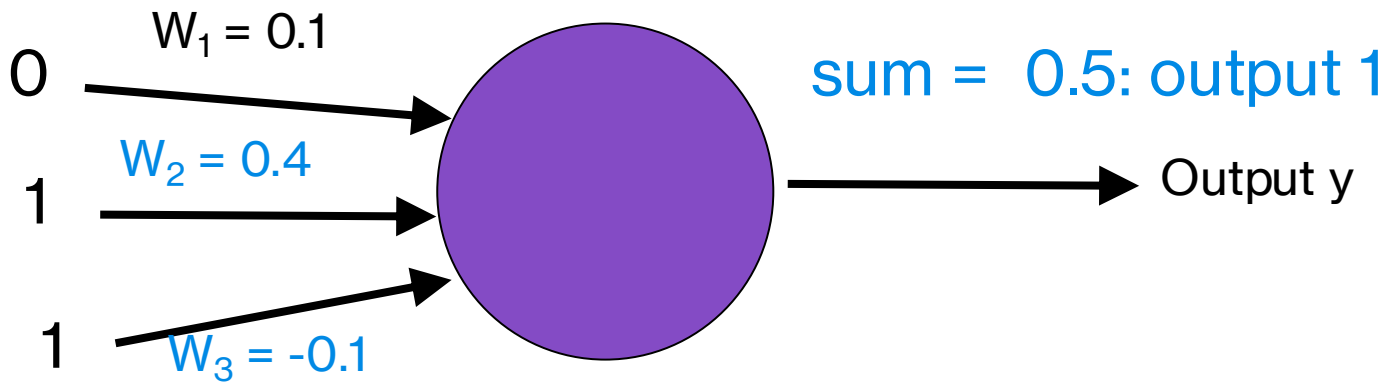
new weights?

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



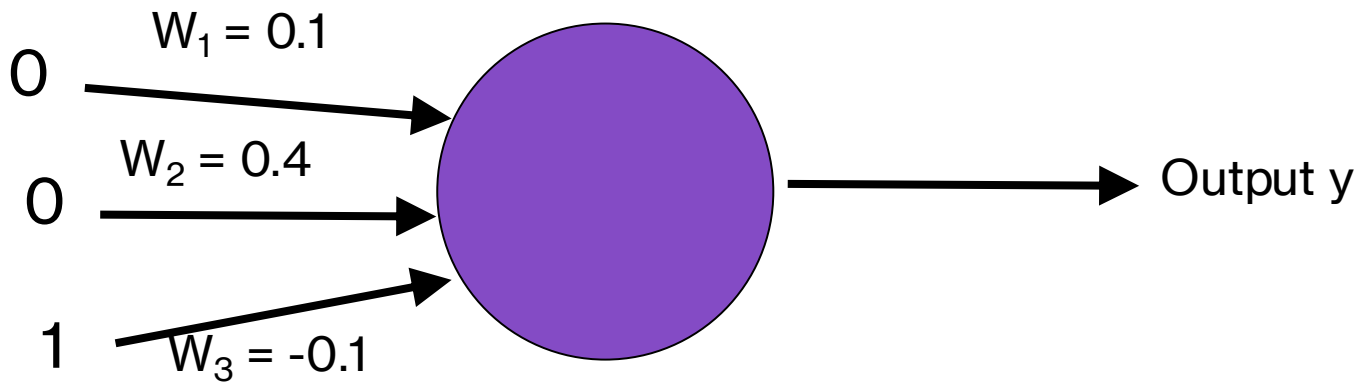
decrease (0-1=-1) all non-zero x_i by 0.1

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



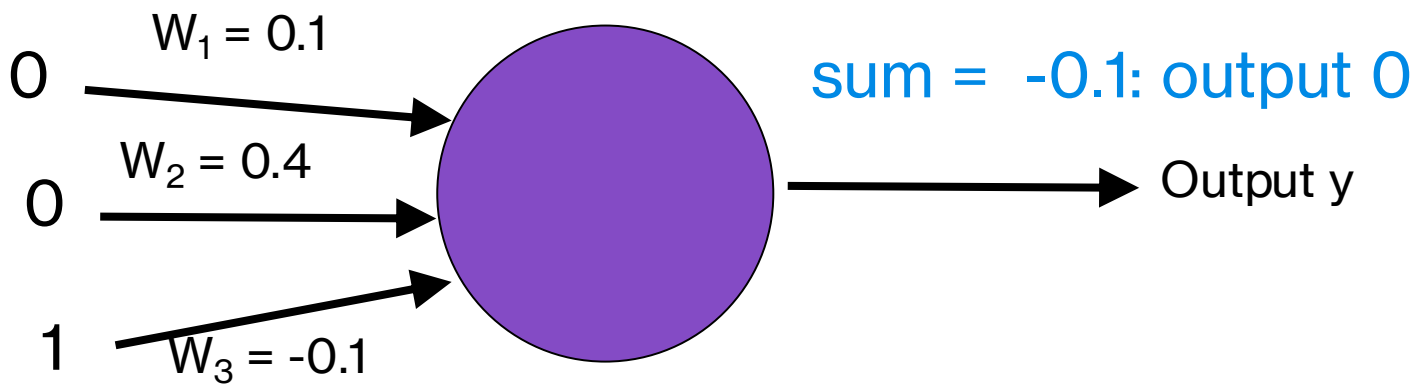
Right or wrong?

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



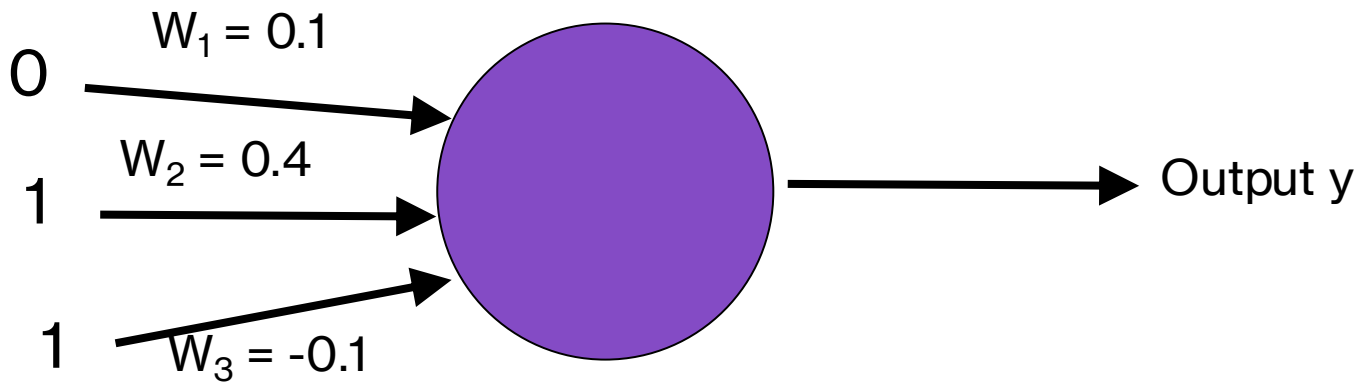
Right. No update!

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



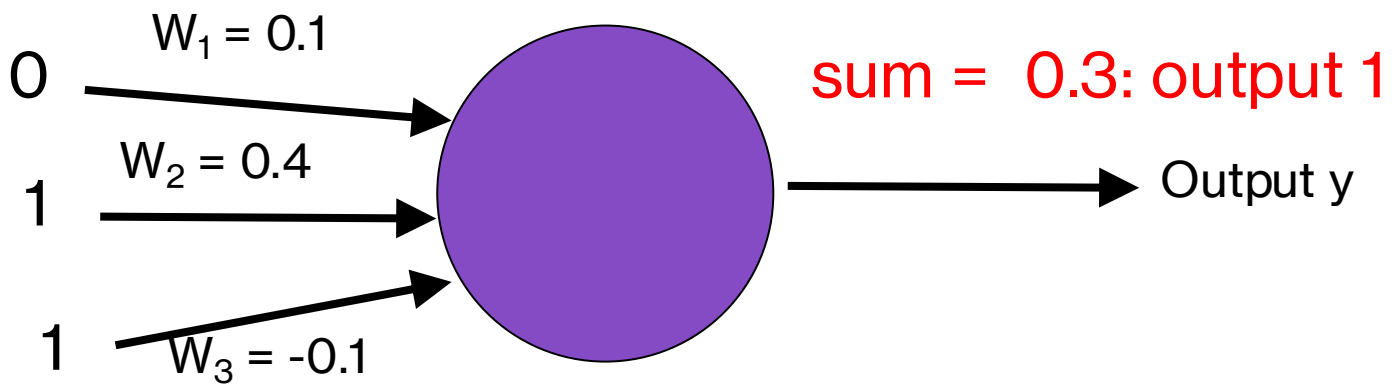
Right or wrong?

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



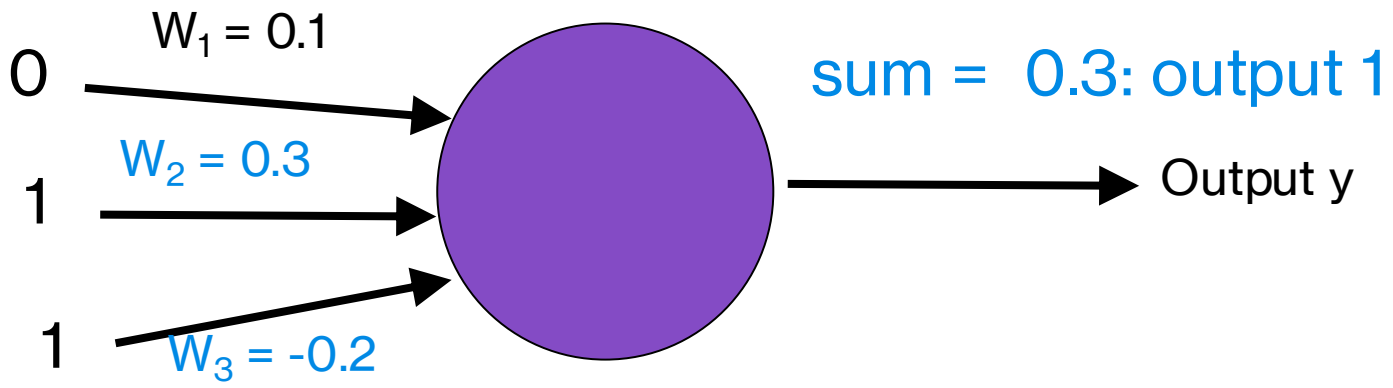
Wrong

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



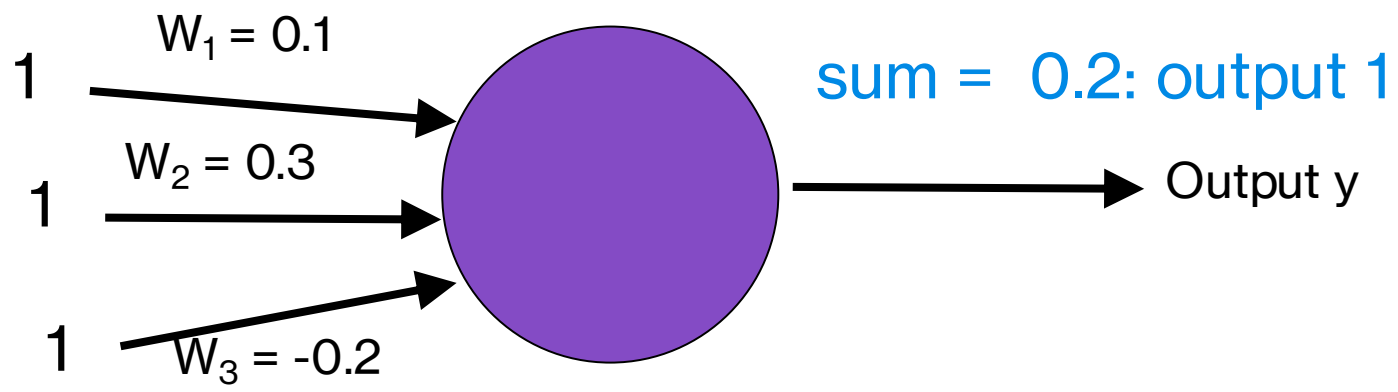
decrease (0-1=-1) all non-zero x_i by 0.1

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



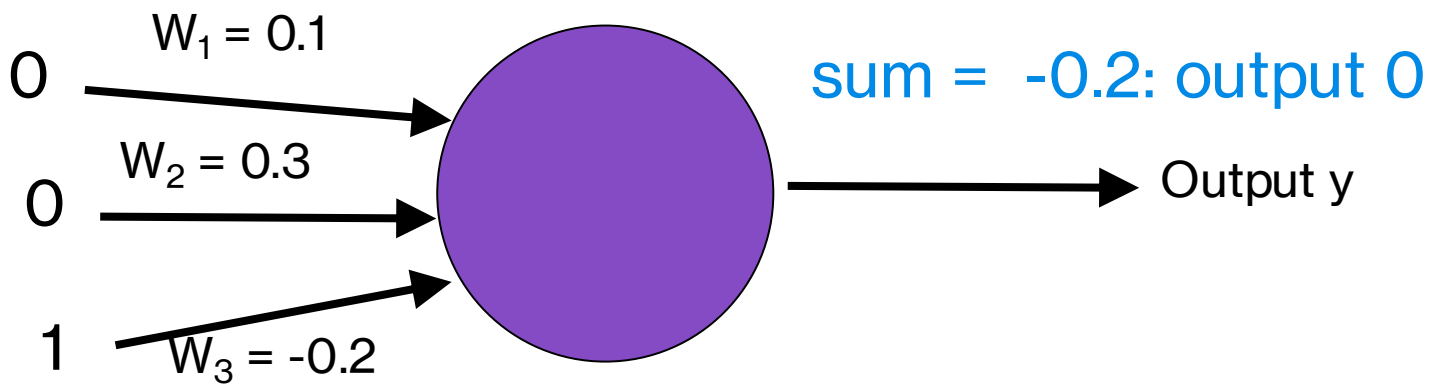
Right. No update!

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



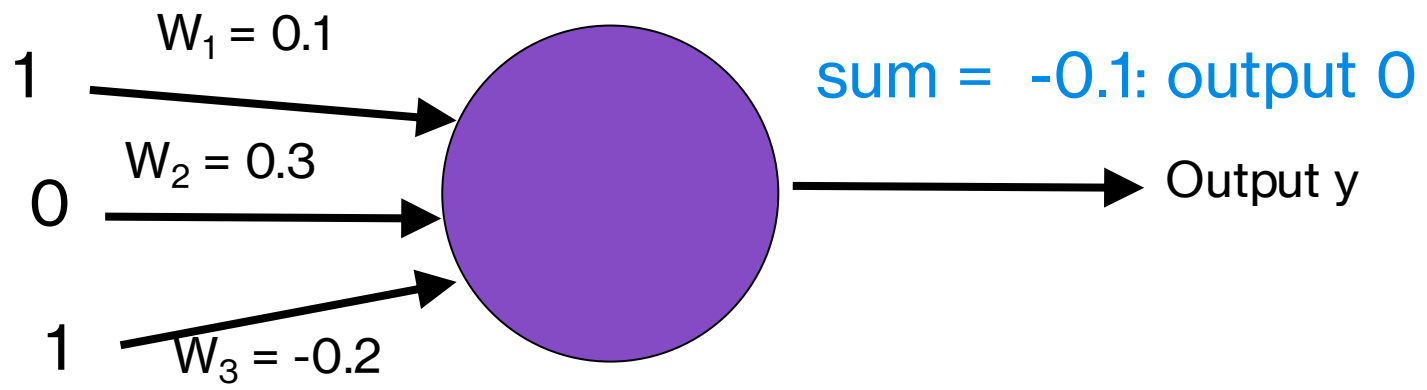
Right. No update!

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



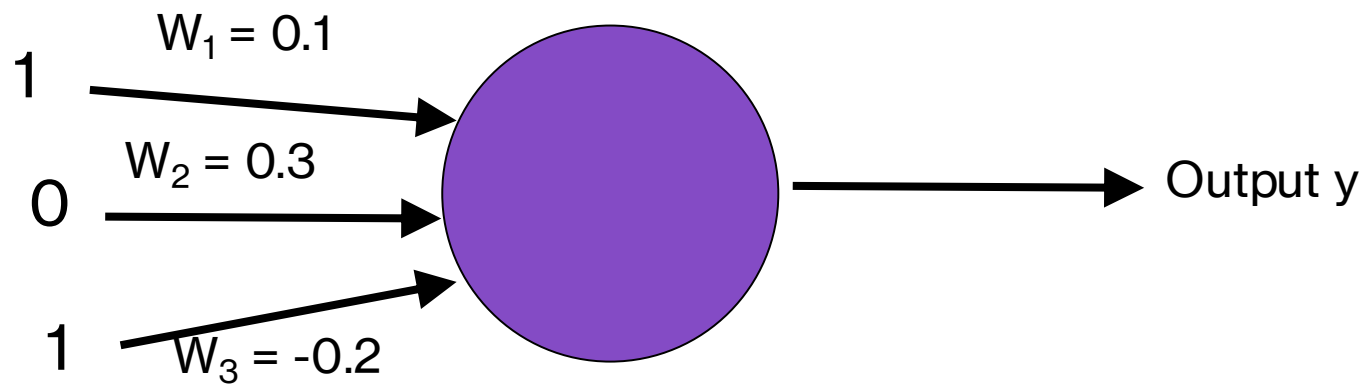
Right. No update!

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



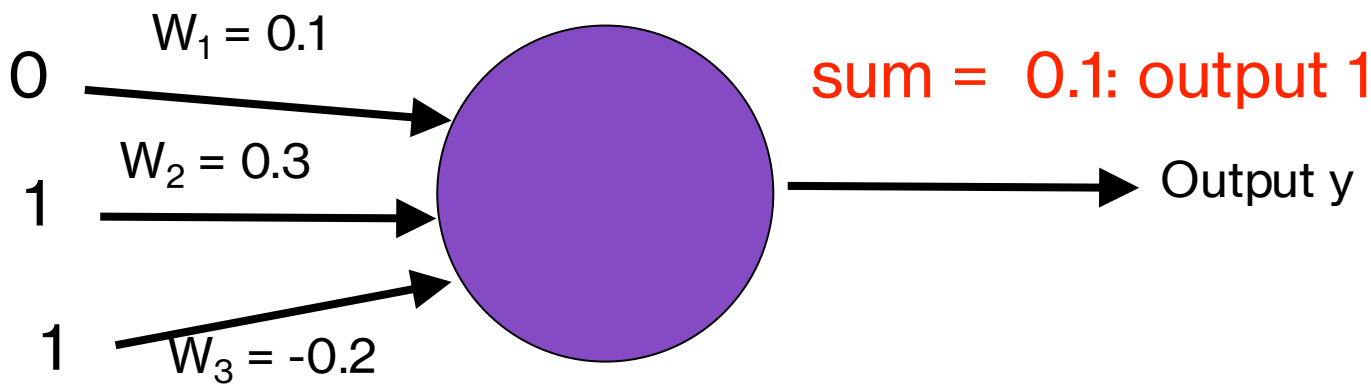
Are they all right?

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



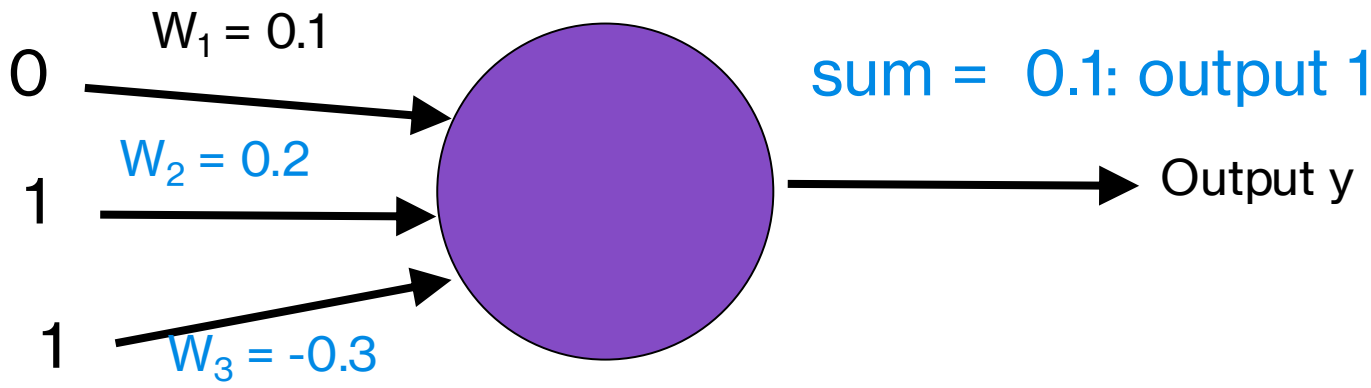
Wrong

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



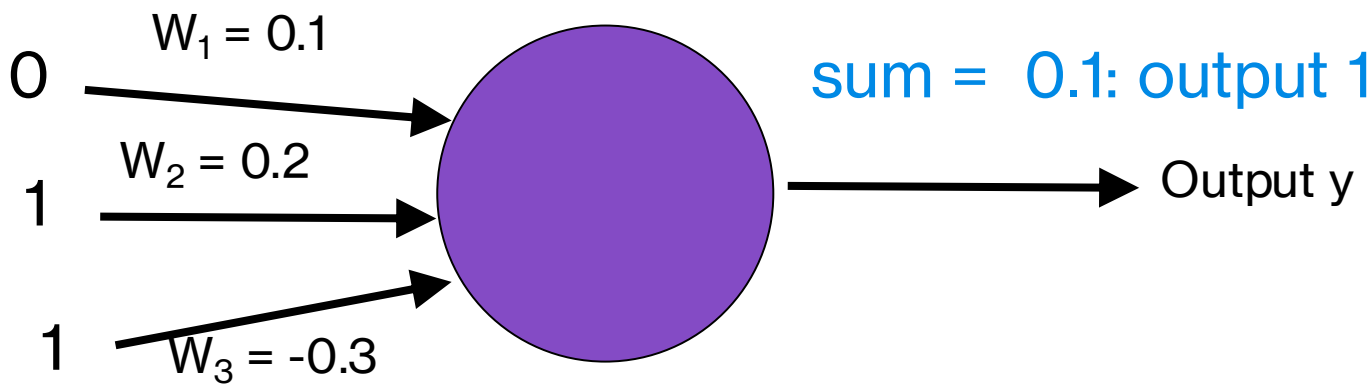
decrease (0-1=-1) all non-zero x_i by 0.1

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



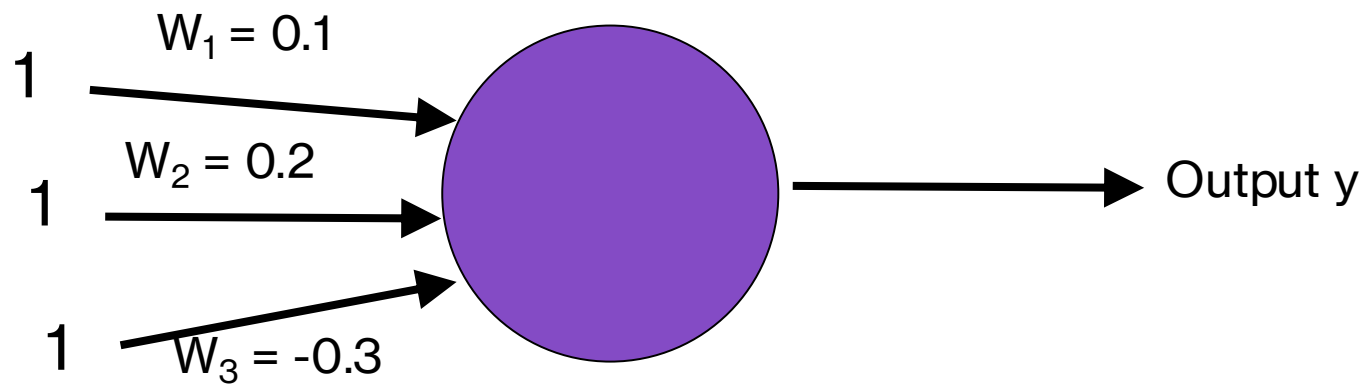
Are they all right?

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

$$w_i = w_i + \lambda * (\text{actual} - \text{predicted}) * x_i$$



We've learned AND!

Perceptron learning algorithm

- A few missing details, but not much more than this
- Keeps adjusting weights as long as it makes mistakes
- If the training data is linearly separable, the perceptron learning algorithm is guaranteed to converge to the “correct” solution (where it gets all examples right)