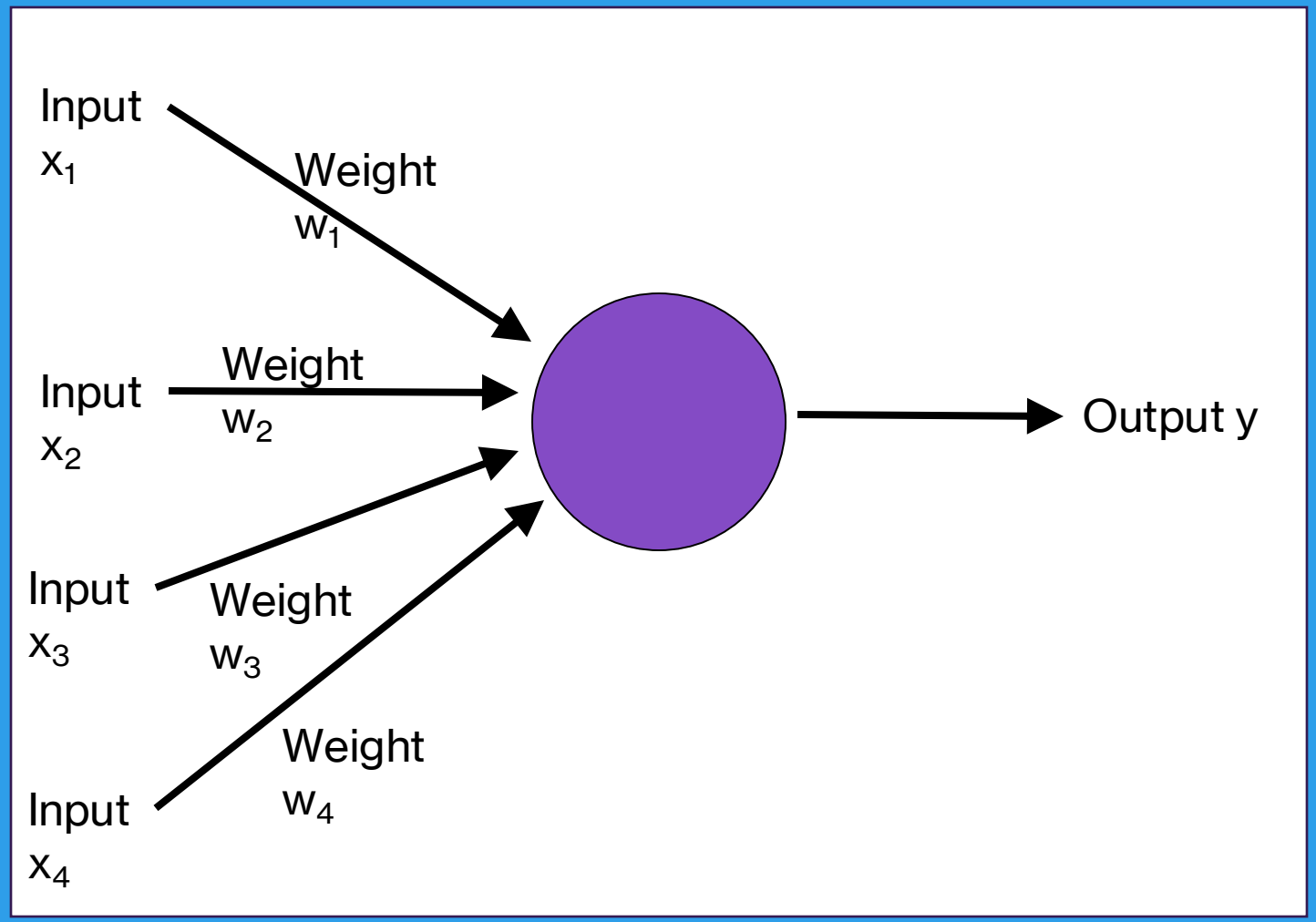
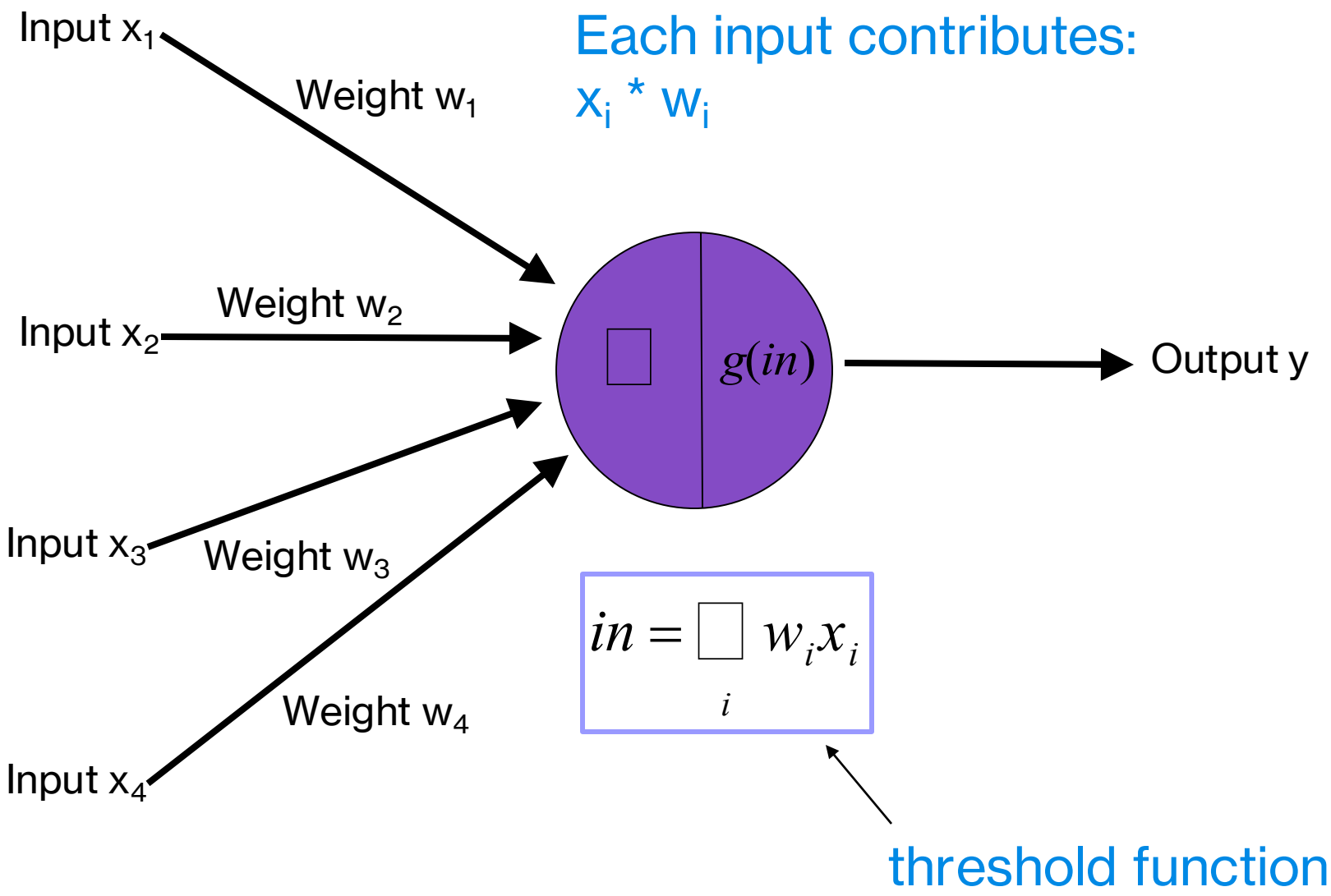
Machine Learning



Perceptron Learning and Recap

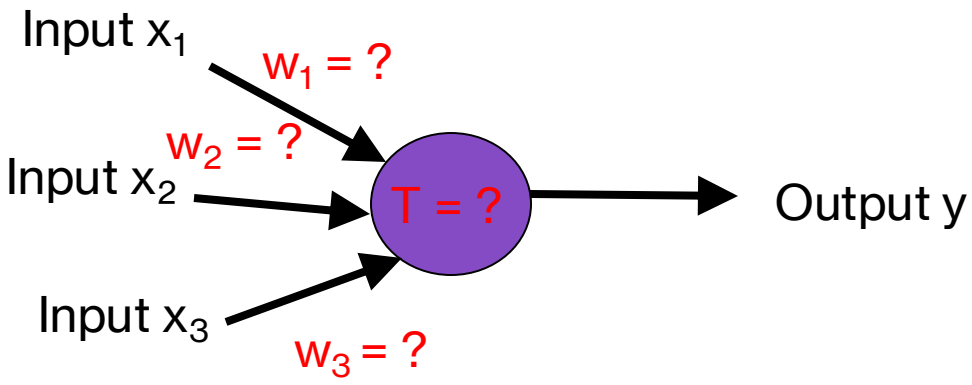
CS51 – Fall 2025

A single neuron/perceptron



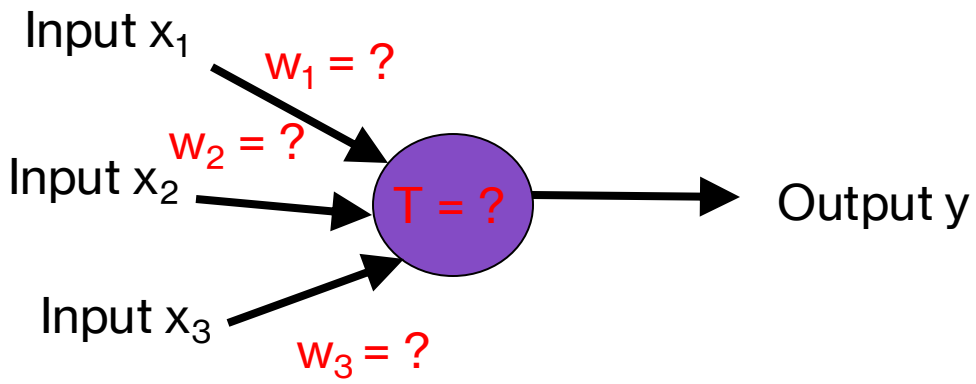
How about...

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



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

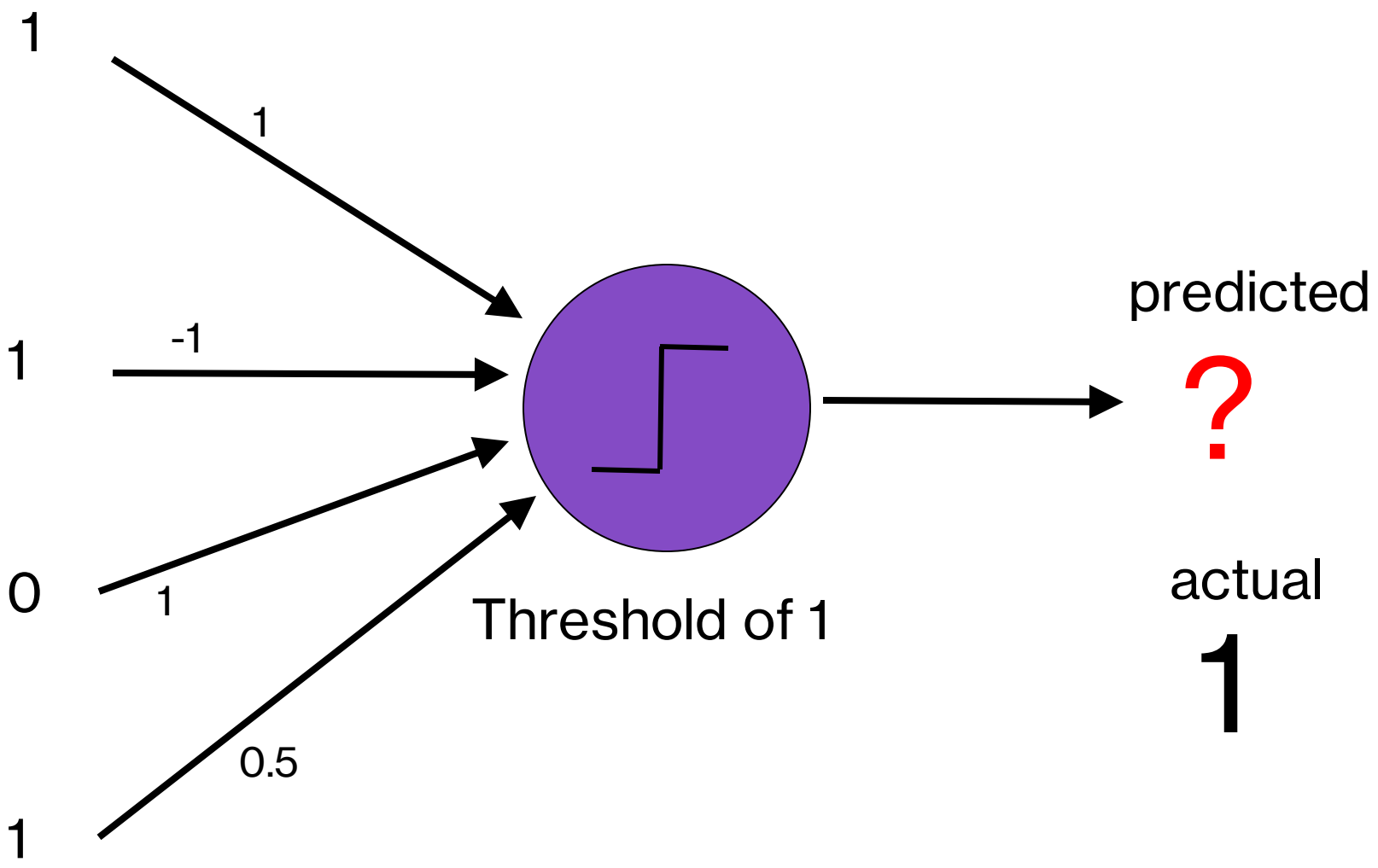


1. start with some initial weights and thresholds
2. show examples repeatedly to NN
3. update weights/thresholds by comparing NN output to actual output

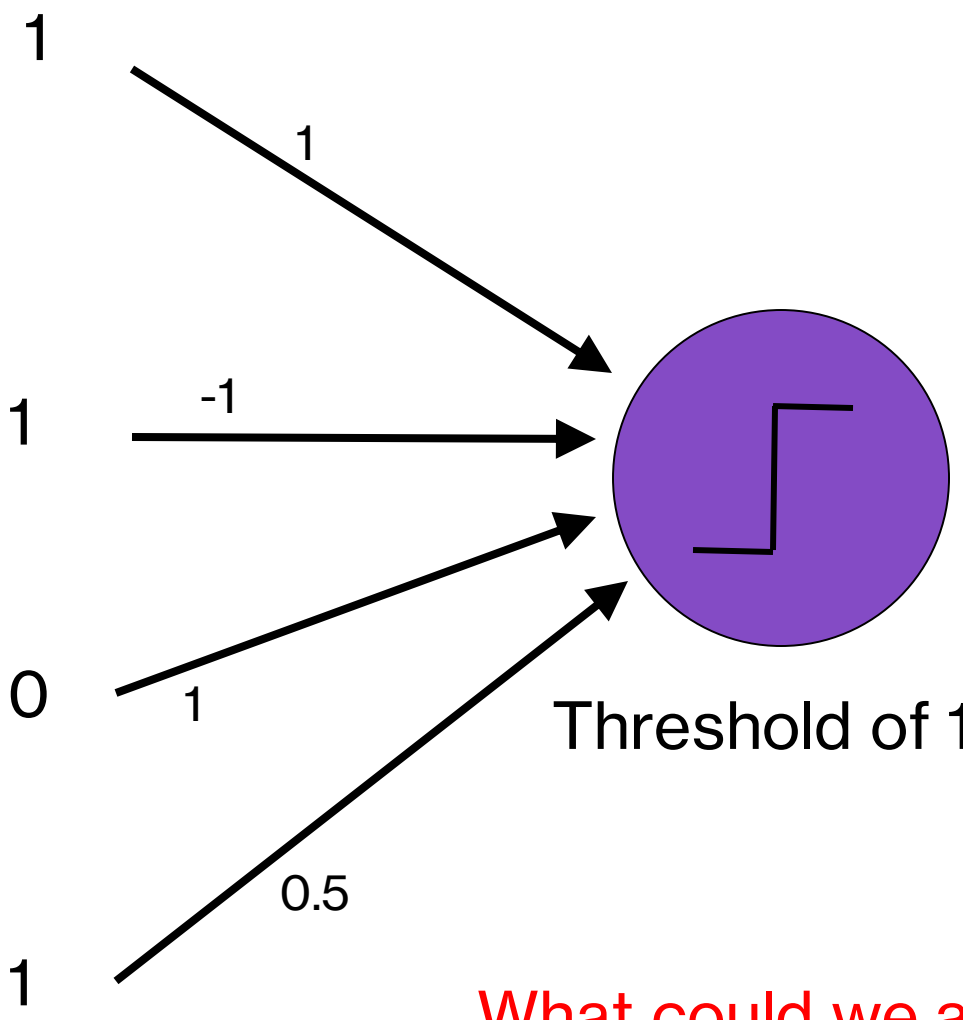
Perceptron learning algorithm

- 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

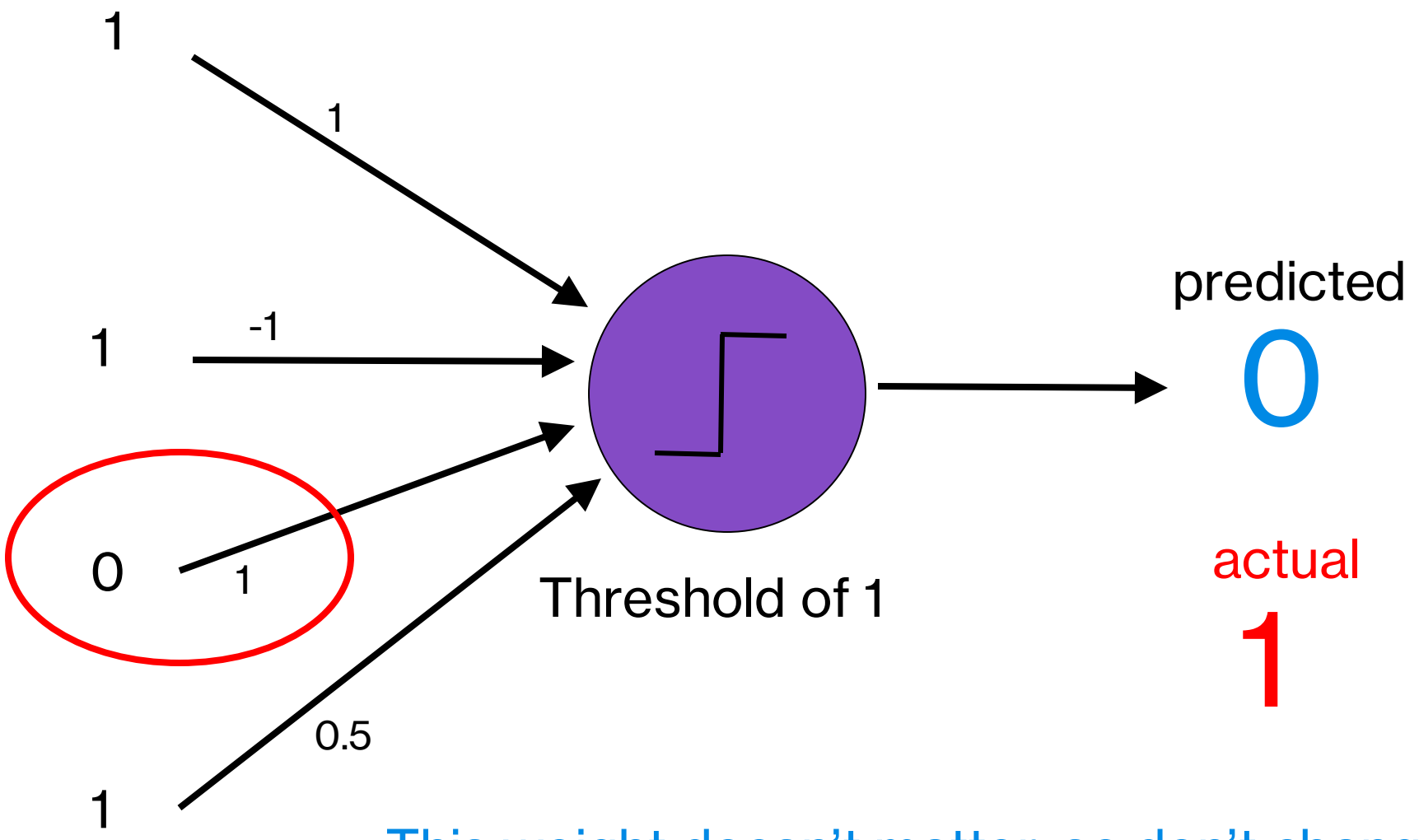
0

actual

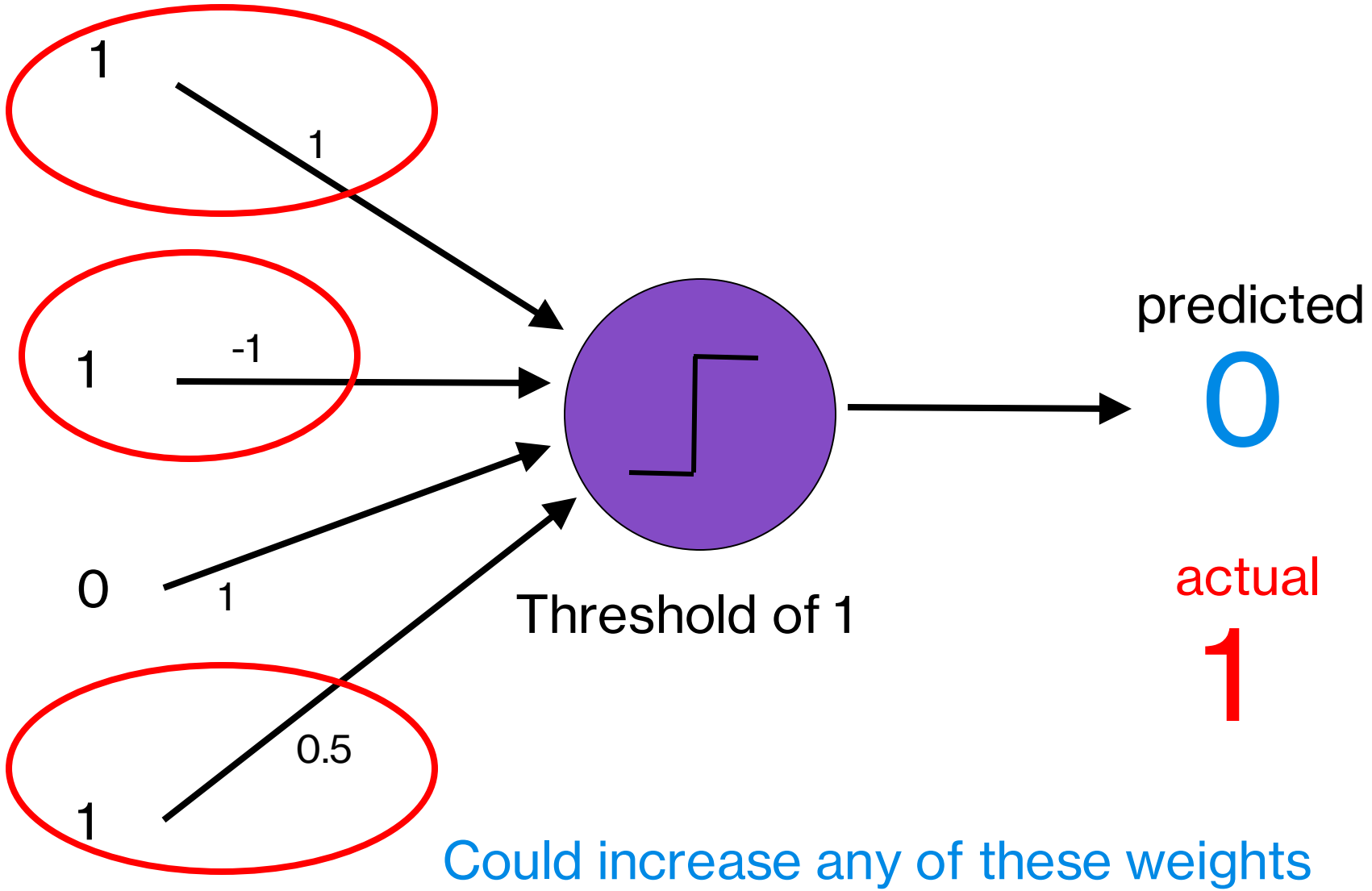
1

What could we adjust to make it right?

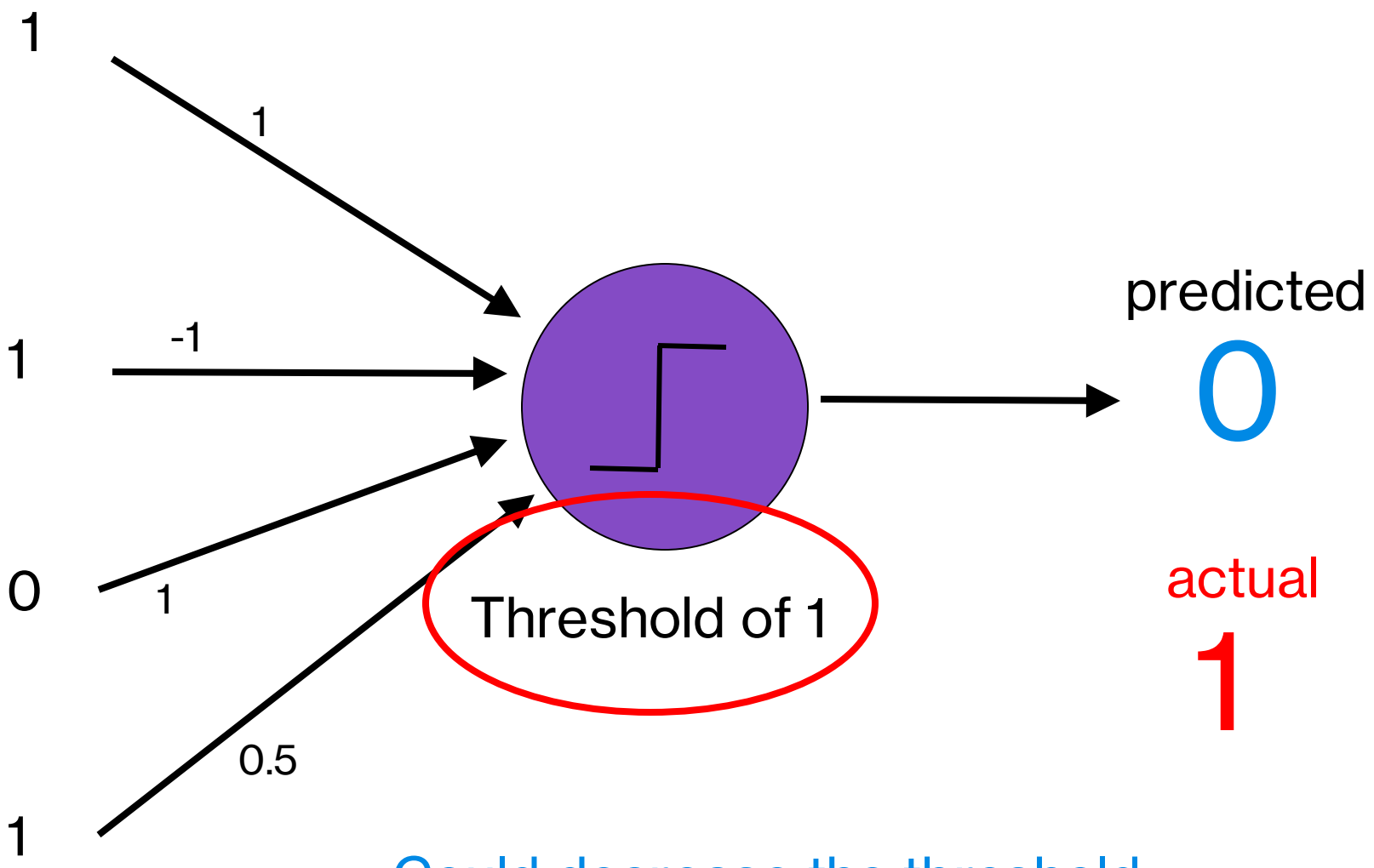
Perceptron learning algorithm



Perceptron learning algorithm



Perceptron learning algorithm

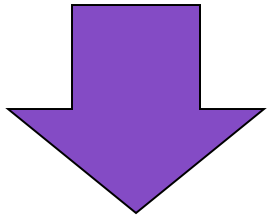


Could decrease the threshold

Perceptron update rule

if wrong:

update weights and threshold towards getting this example correct

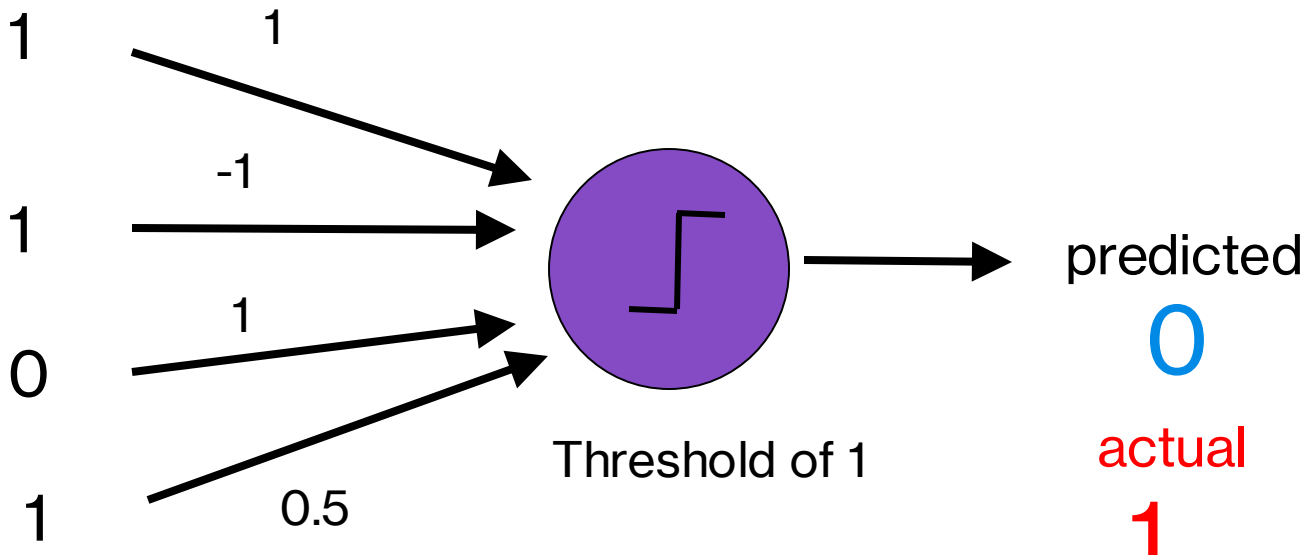


if wrong:

$$w_i = w_i + \Delta w_i$$

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

Perceptron learning algorithm

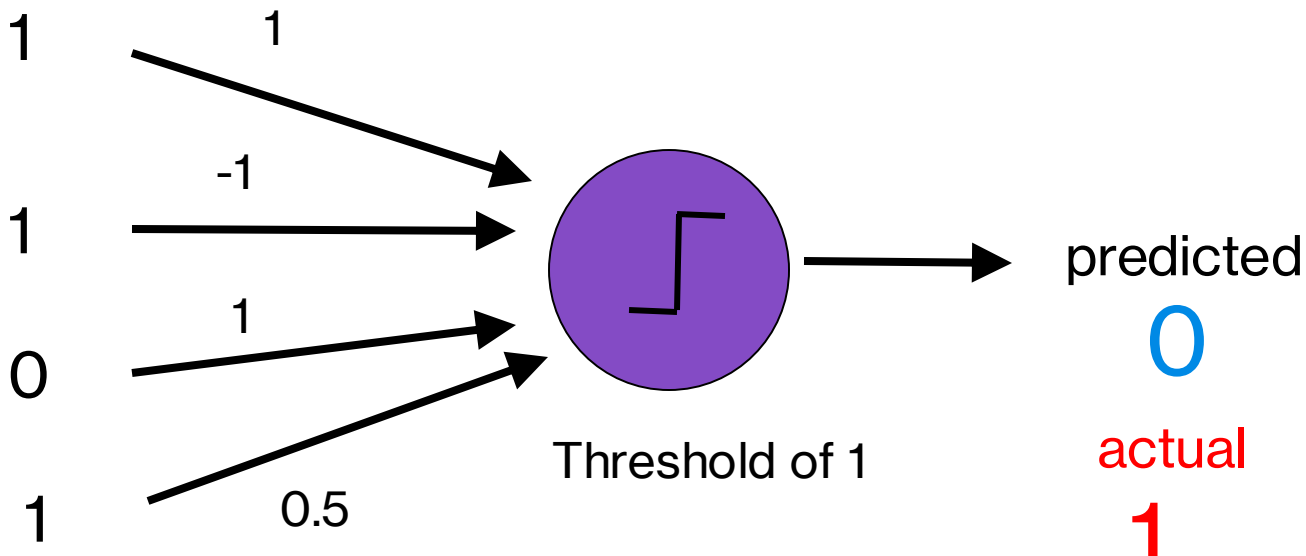


$$w_i = w_i + \Delta wi$$

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

What does this do in this case?

Perceptron learning algorithm

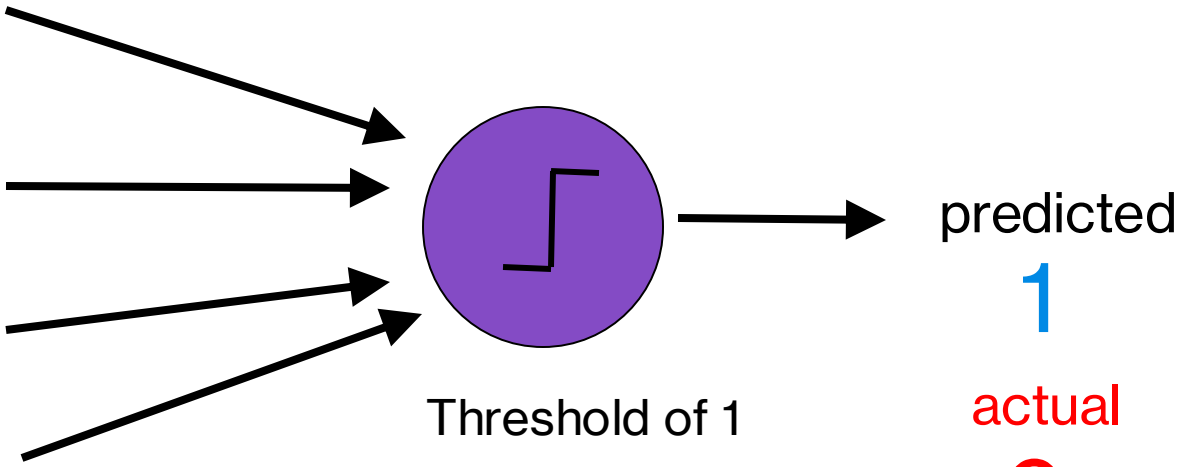


$$w_i = w_i + \Delta wi$$

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

causes us to increase the weights!

Perceptron learning algorithm

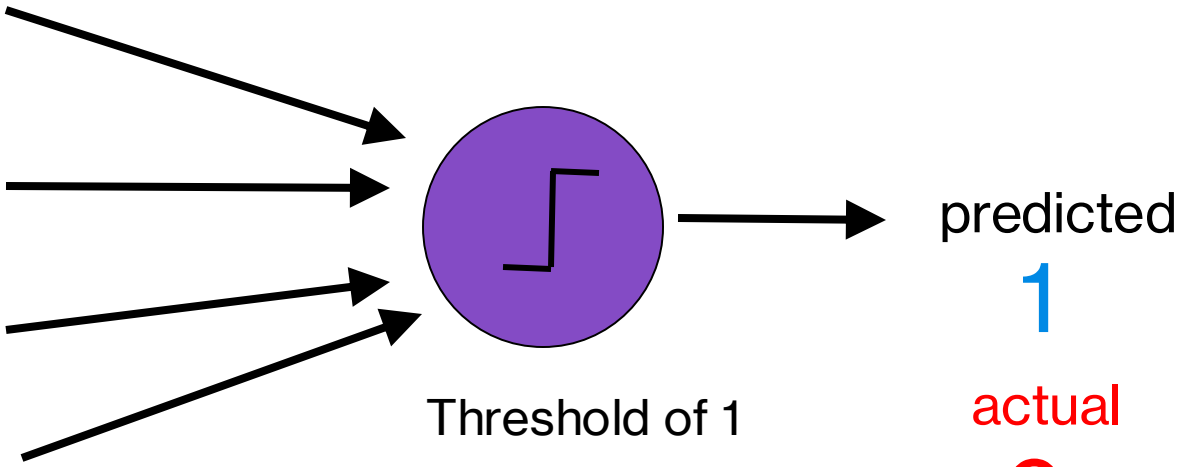


$$w_i = w_i + \Delta wi$$

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

What if predicted = 1 and actual = 0?

Perceptron learning algorithm

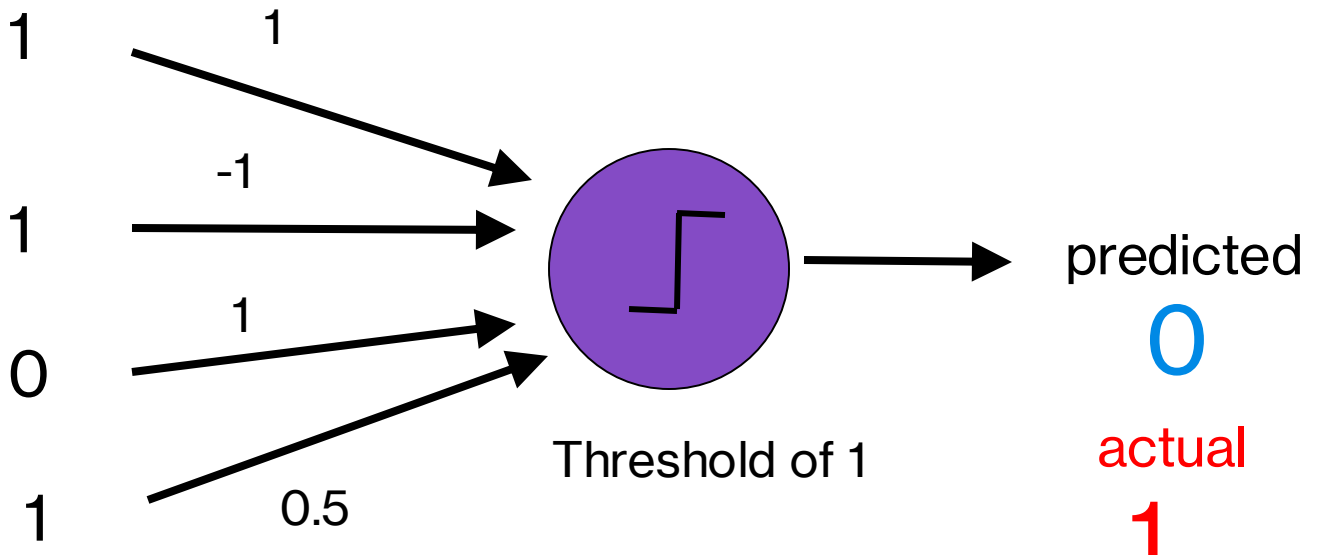


$$w_i = w_i + \Delta wi$$

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

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

Perceptron learning algorithm

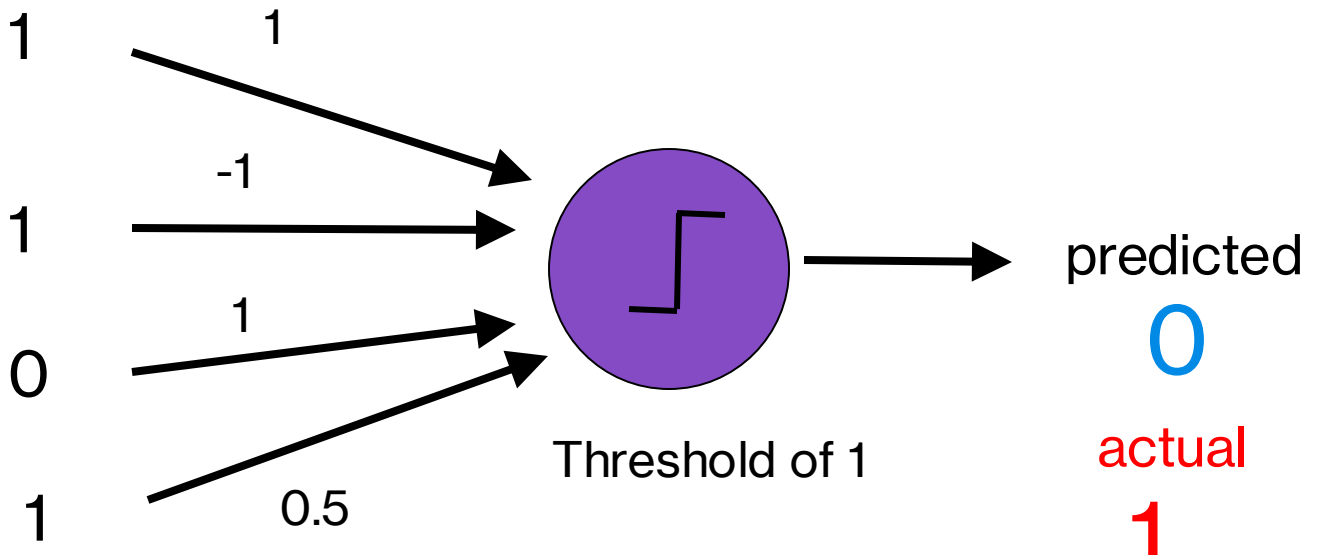


$$w_i = w_i + \Delta wi$$

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

What does this do?

Perceptron learning algorithm

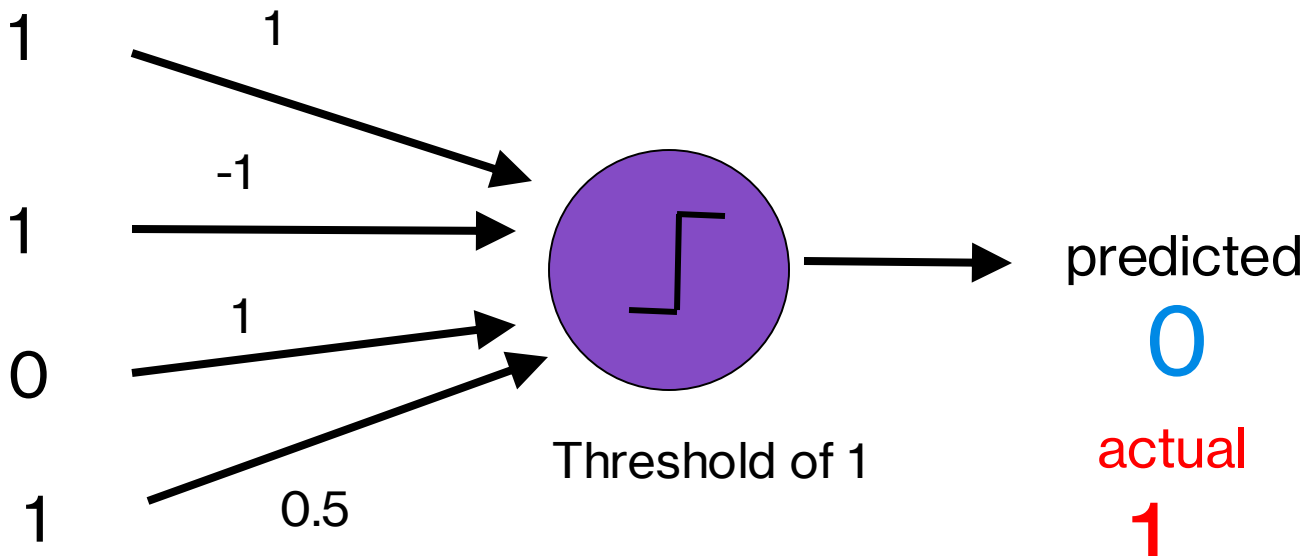


$$w_i = w_i + \Delta wi$$

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

Only adjust those weights that actually contributed!

Perceptron learning algorithm

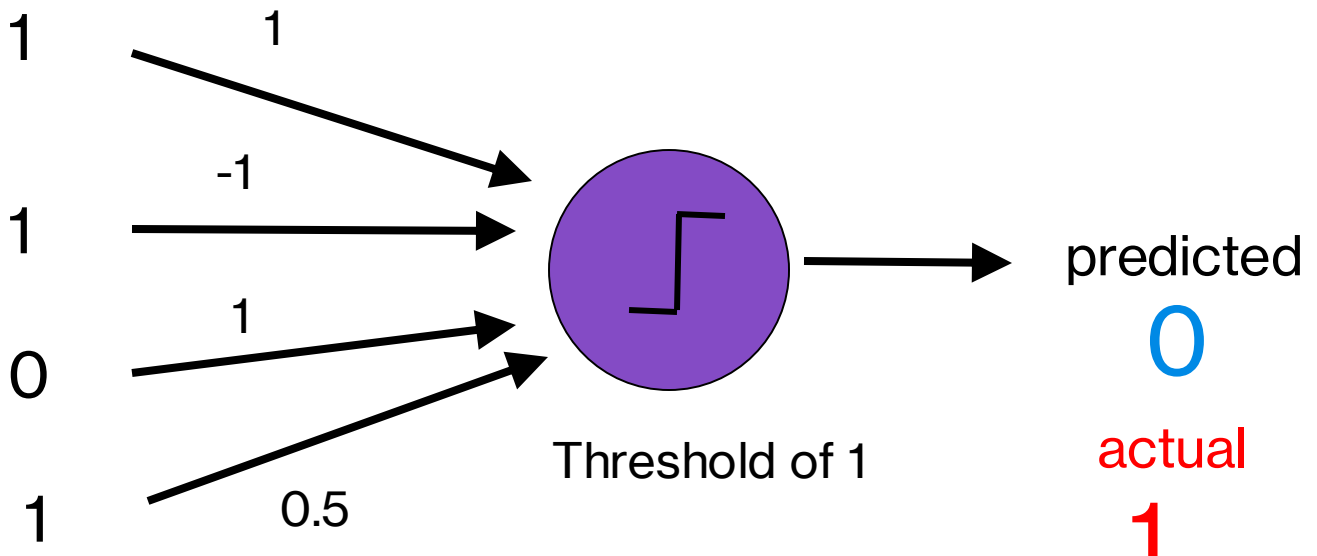


$$w_i = w_i + \Delta wi$$

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

What does this do?

Perceptron learning algorithm

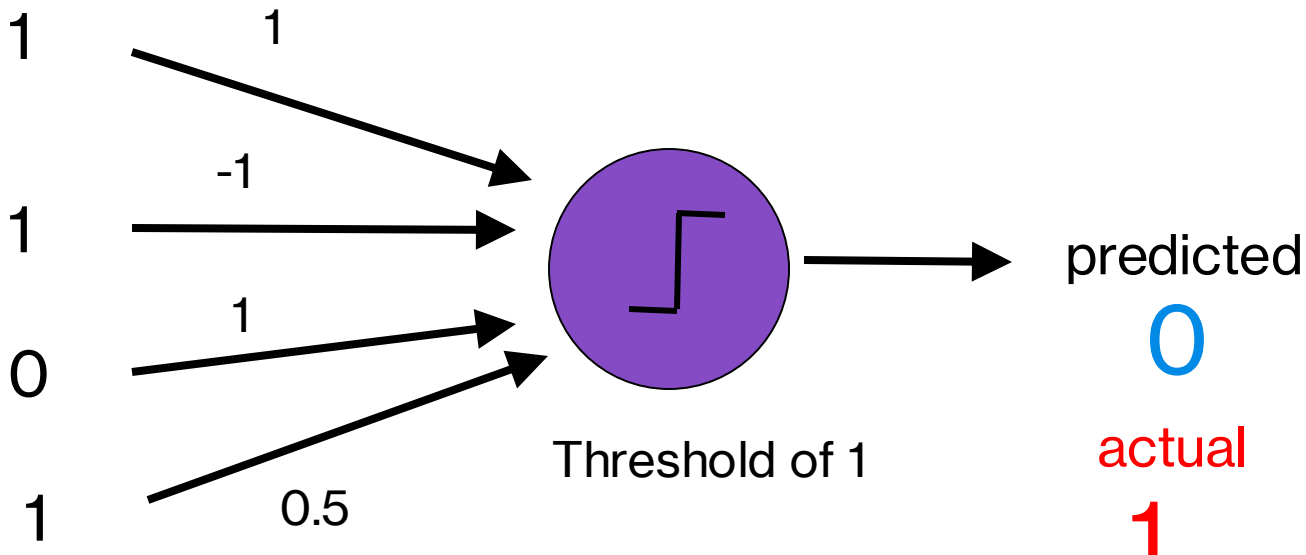


$$w_i = w_i + \Delta wi$$

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

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

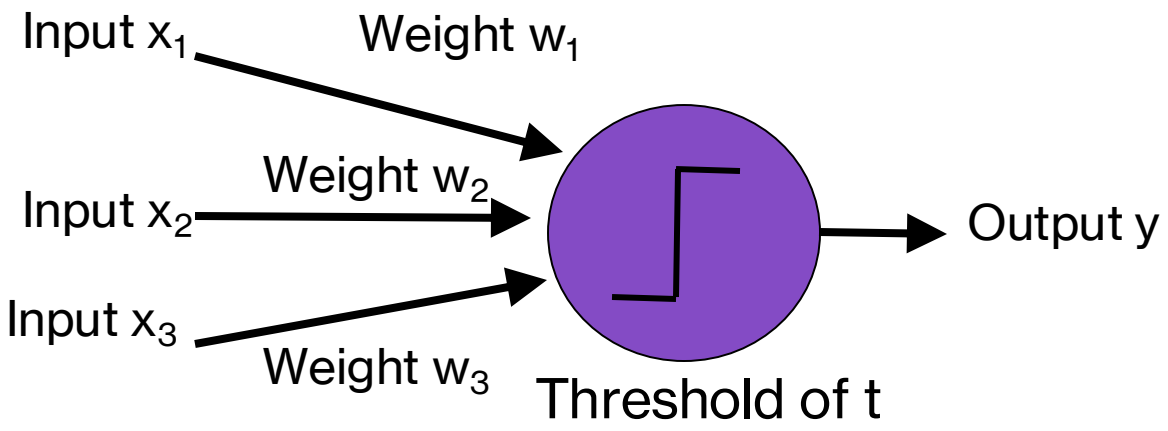
Perceptron learning algorithm



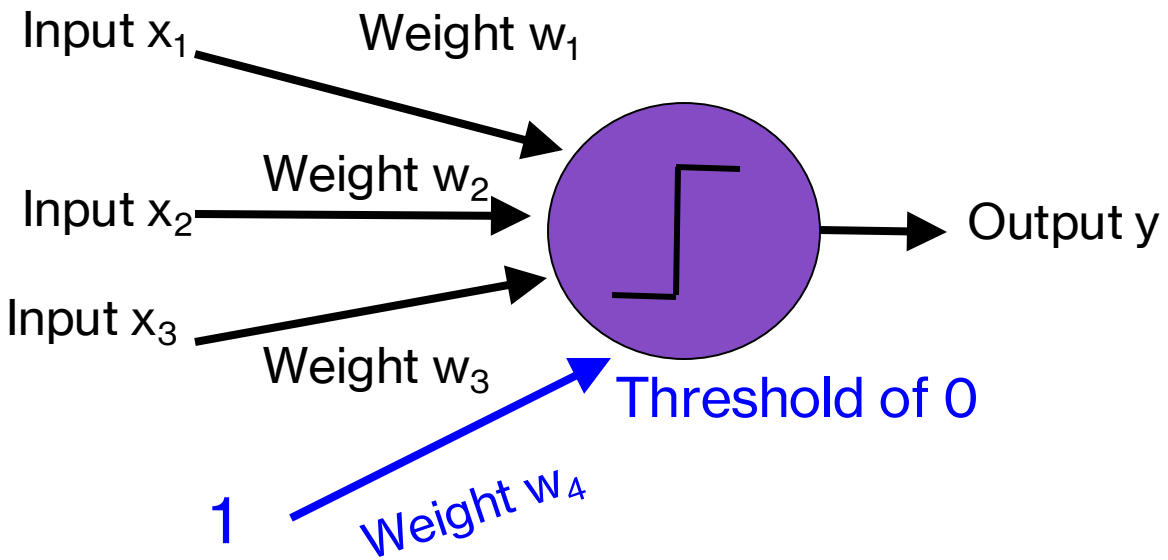
$$w_i = w_i + \Delta wi$$

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

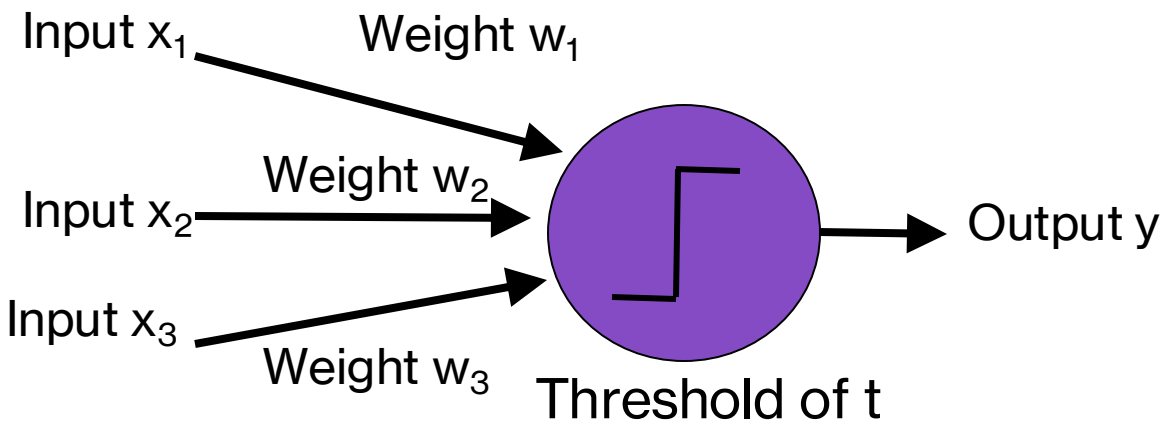
What about the threshold?



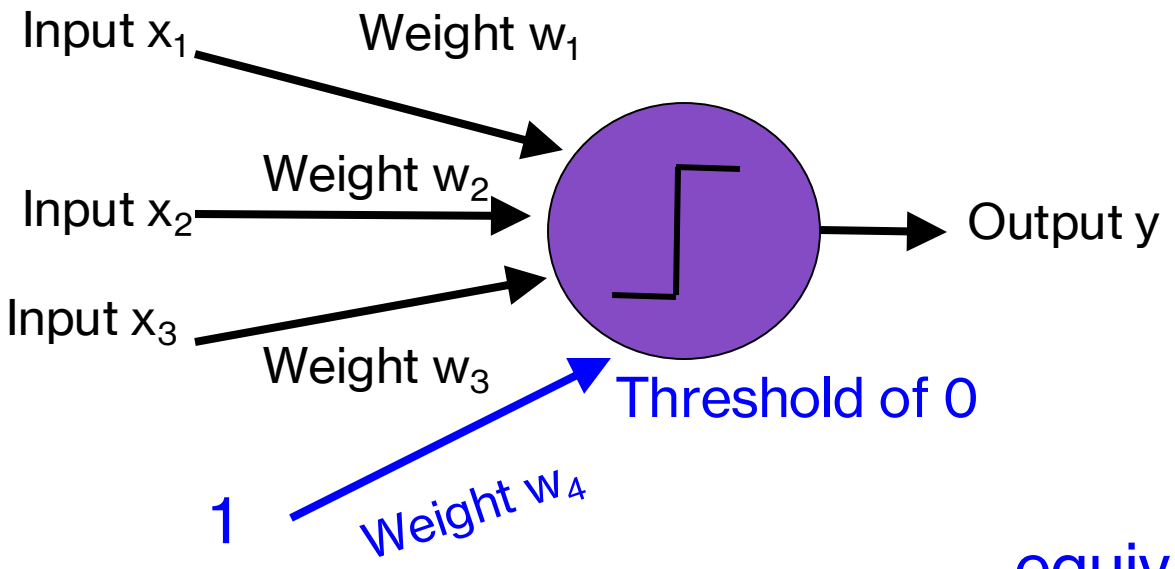
$$1 \text{ if } \sum_{i=1}^{3} w_i x_i \geq t$$



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



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



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

equivalent when $w_4 = -t$

Perceptron learning algorithm

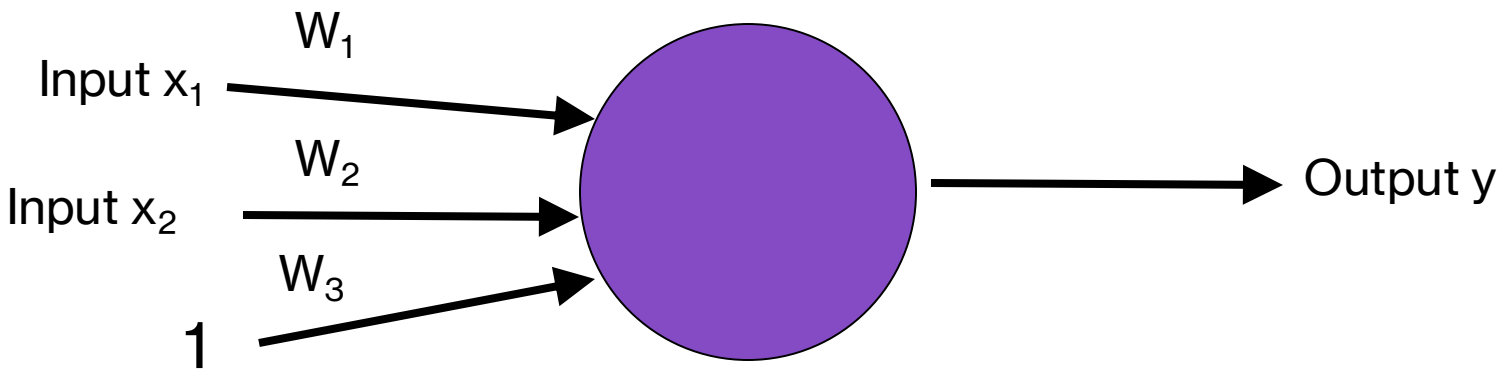
- initialize weights of the model randomly
- repeat until you get all examples right:
- for each “training” example (in a random order):
- calculate current prediction on the example
- if wrong:

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

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

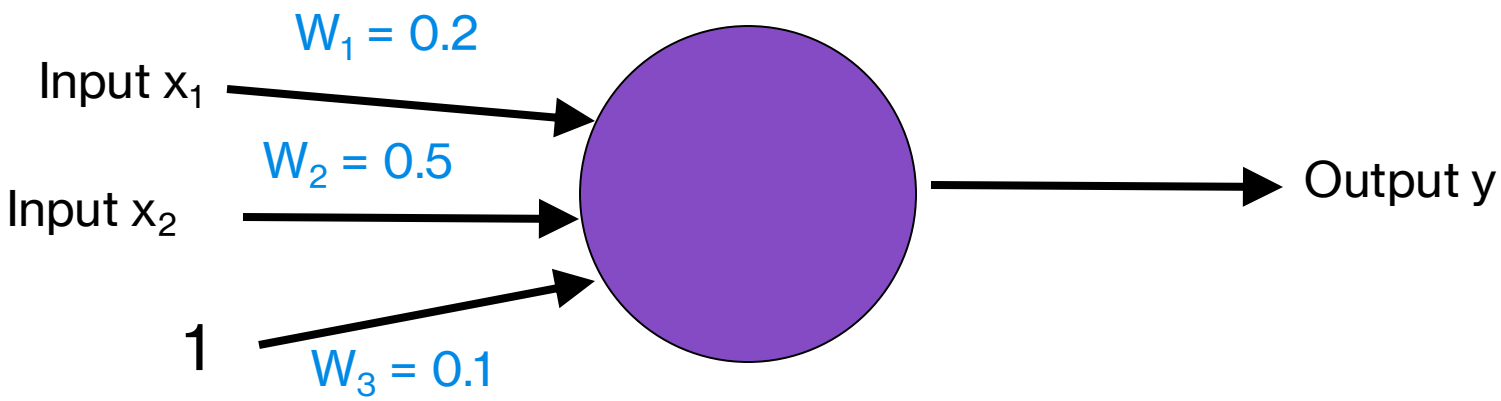
$\lambda = 0.1$

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

$\lambda = 0.1$

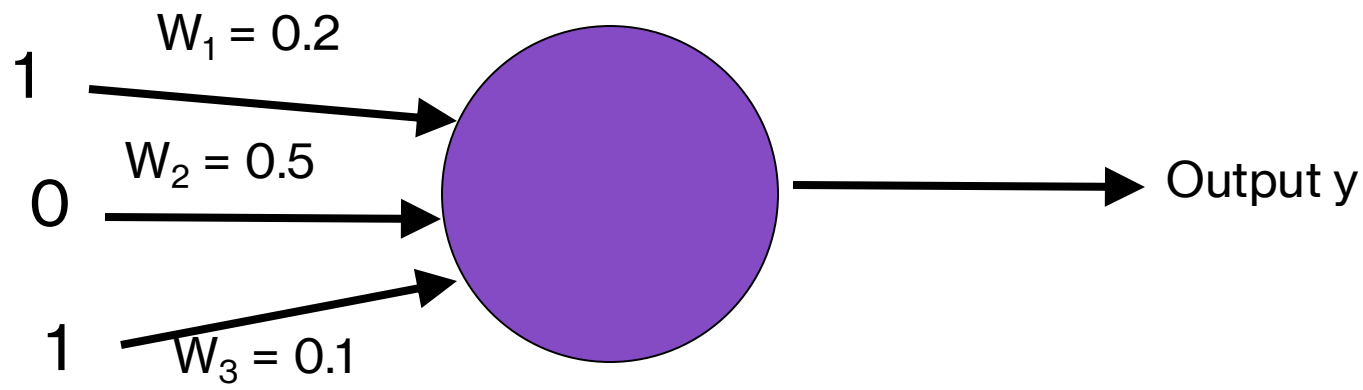


x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

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



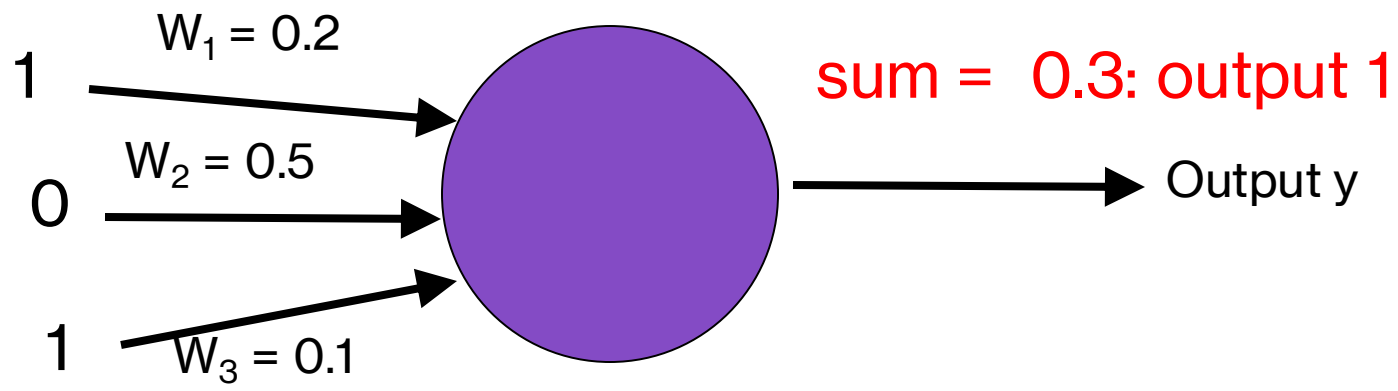
Right or wrong?

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:

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



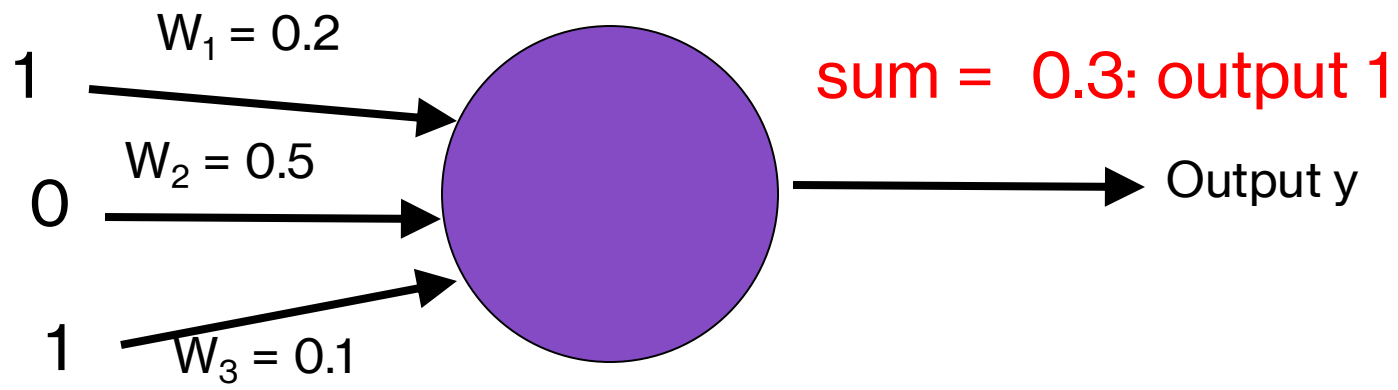
Wrong

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?

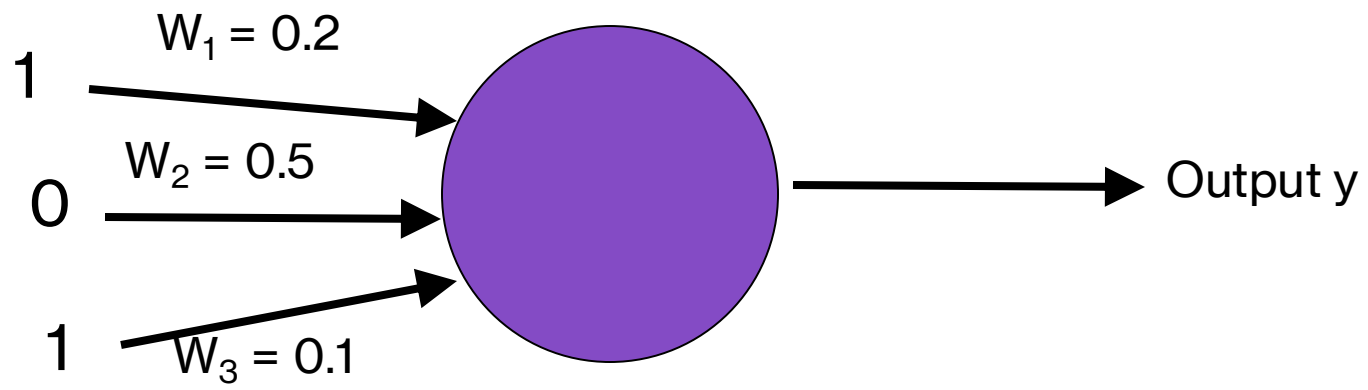
x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

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



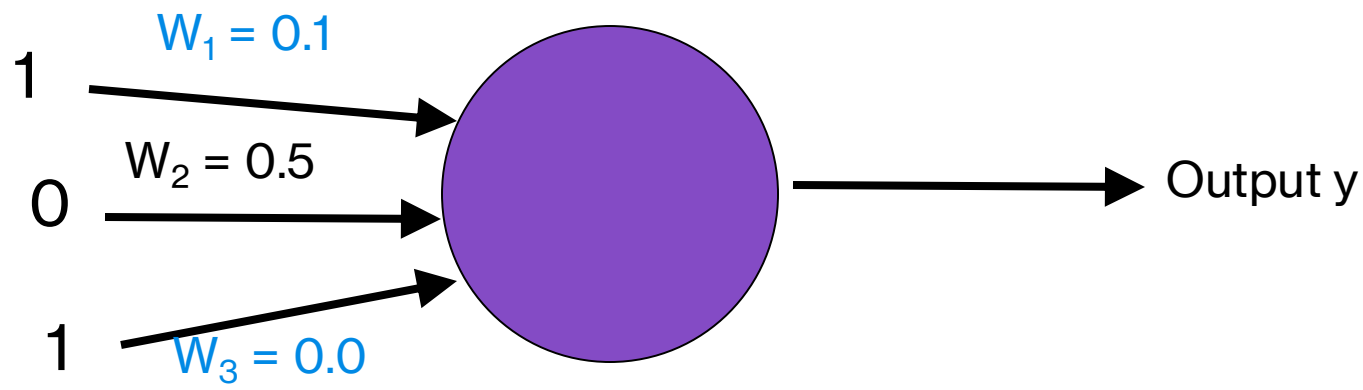
x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

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

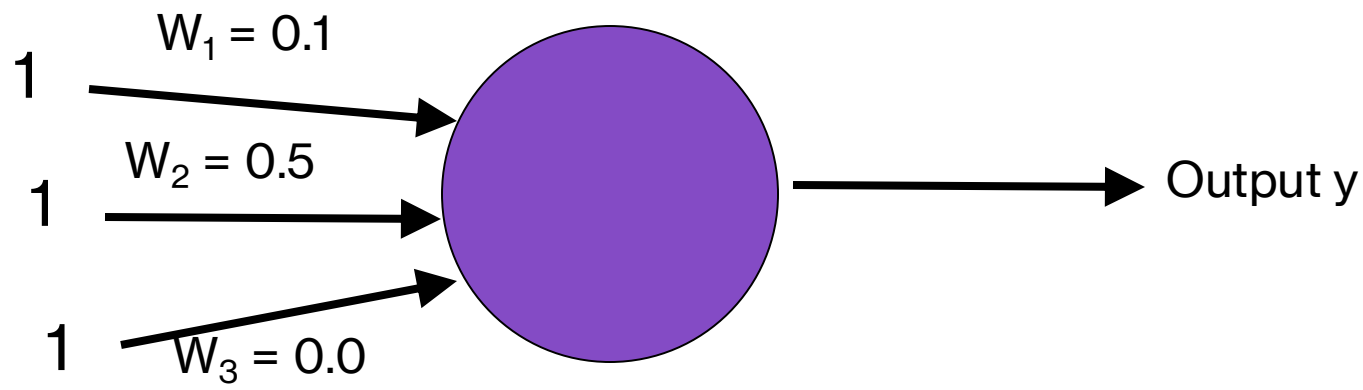


x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

$\lambda = 0.1$

if wrong:

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



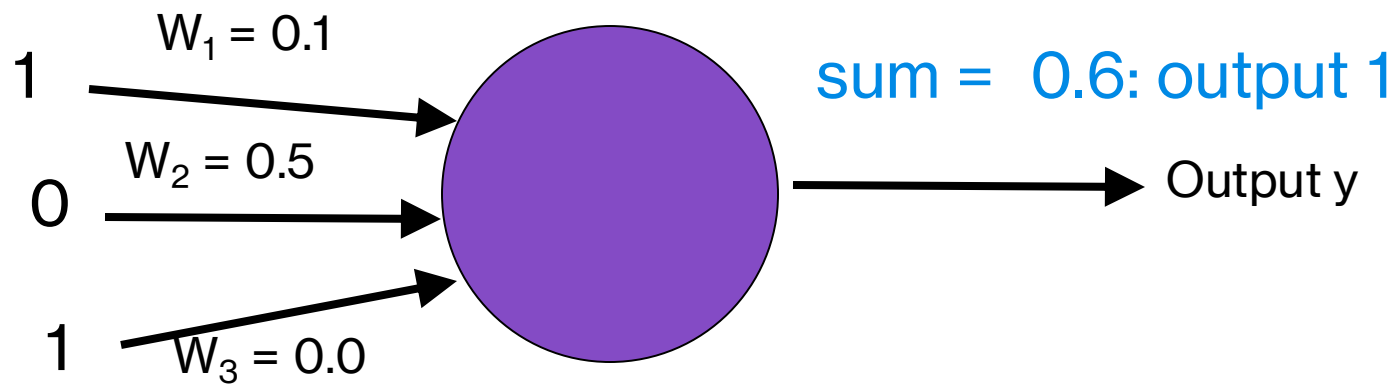
Right or wrong?

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:

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



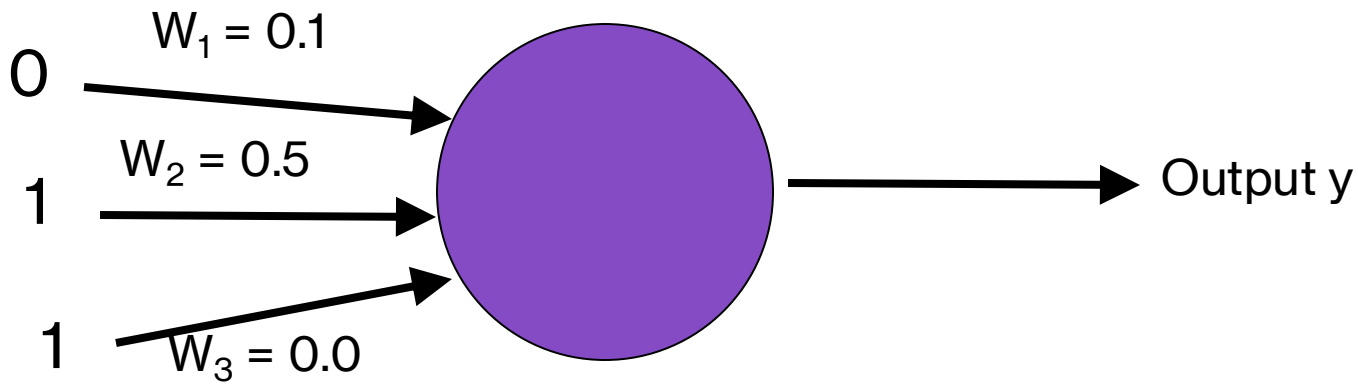
Right. No update!

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:

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



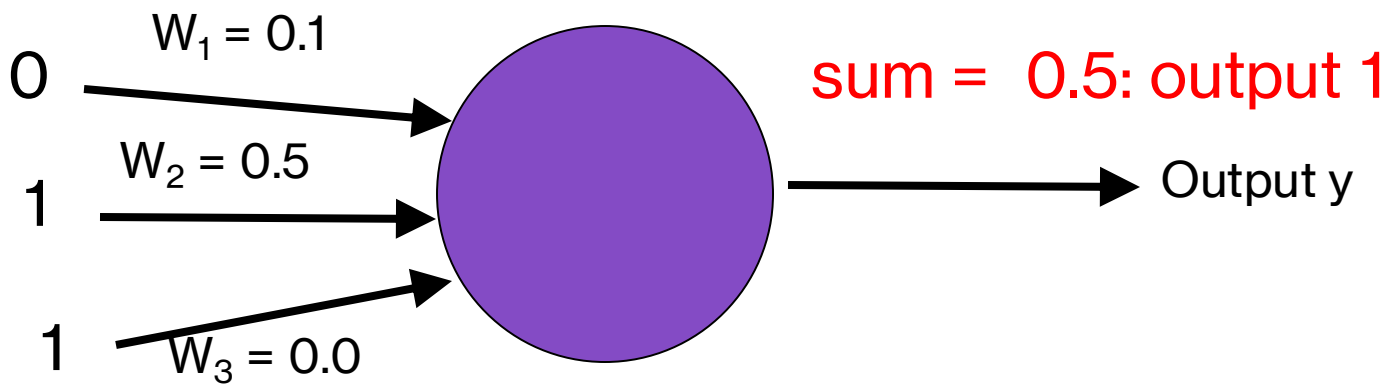
Right or wrong?

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:

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



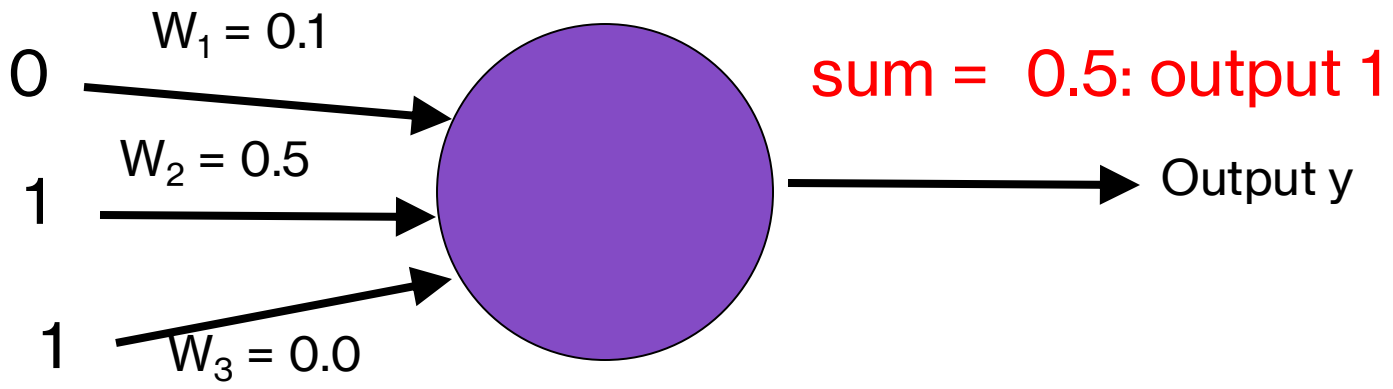
Wrong

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:

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



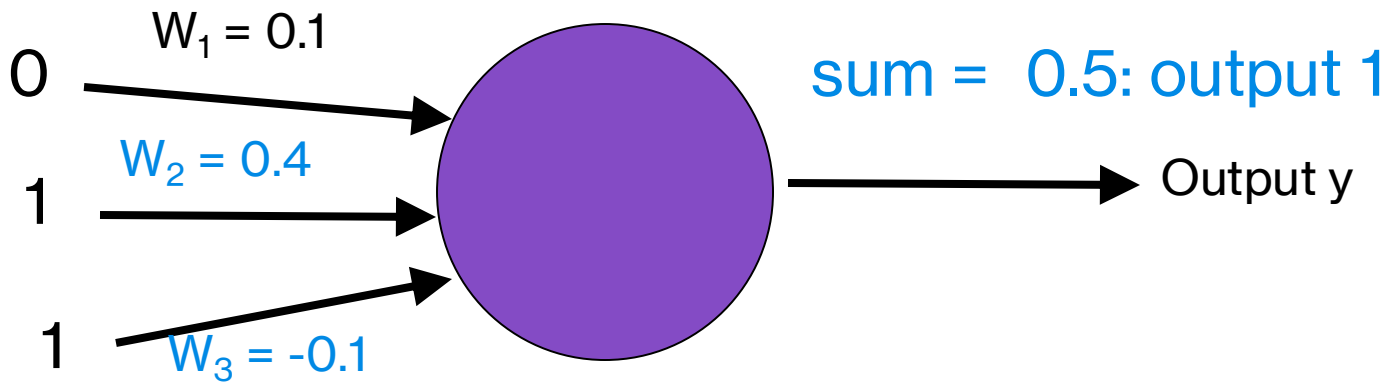
new weights?

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:

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



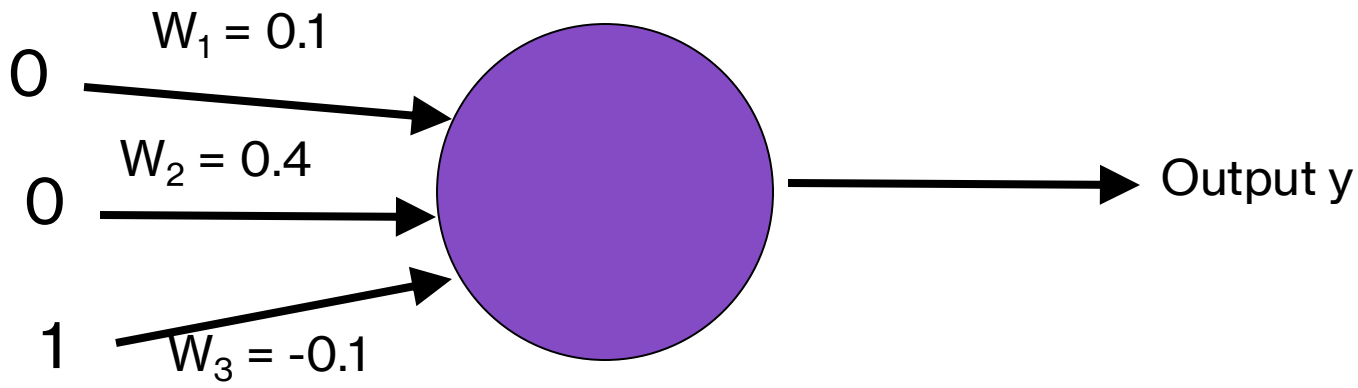
decrease (0-1=-1) all non-zero x_i by 0.1

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:

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



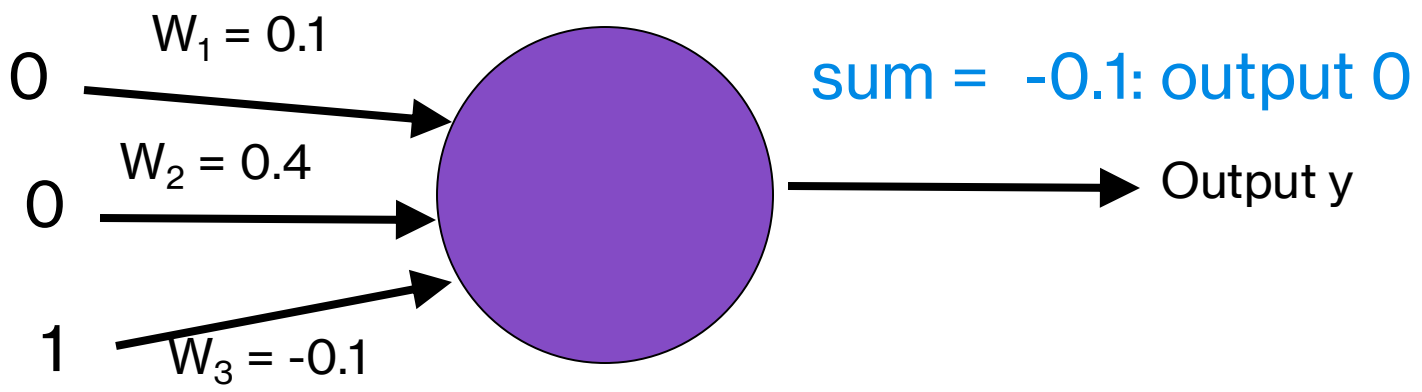
Right or wrong?

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:

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



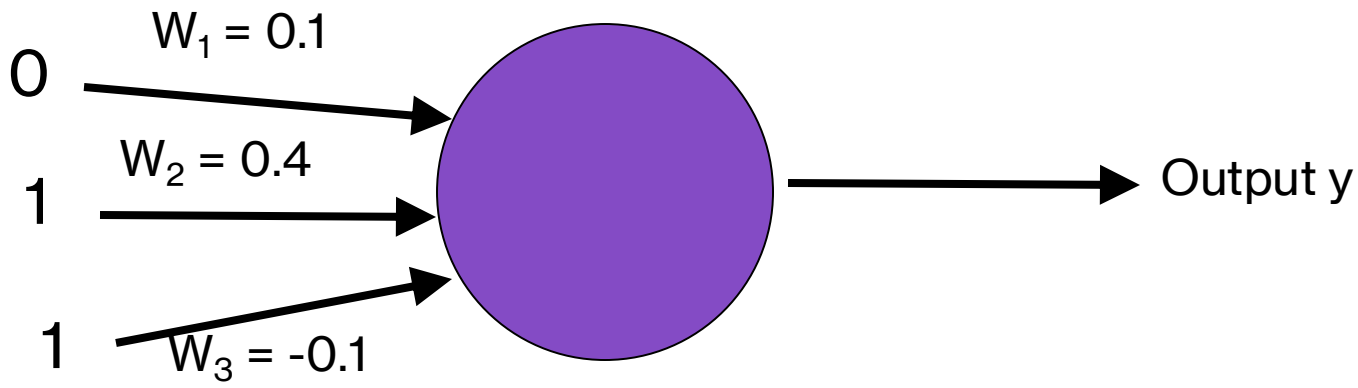
Right. No update!

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:

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



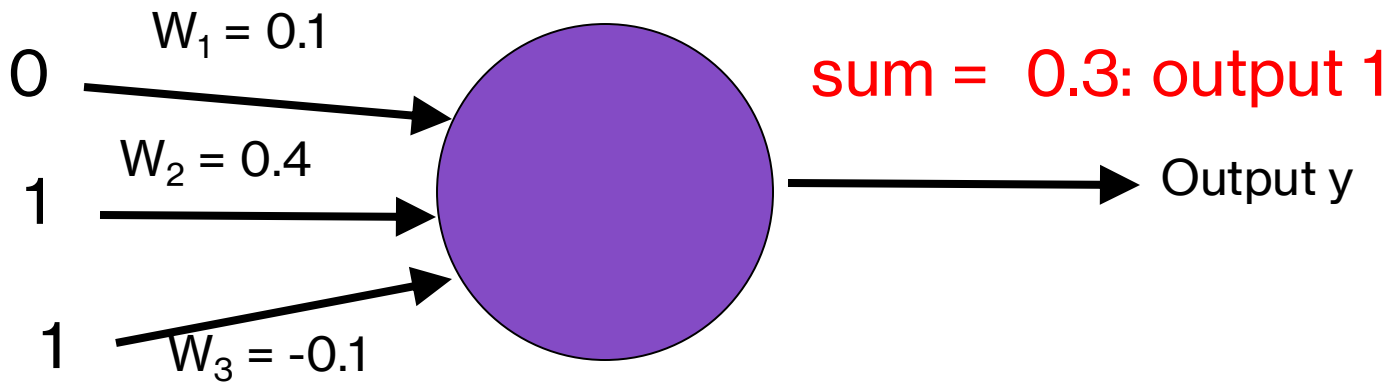
Right or wrong?

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:

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



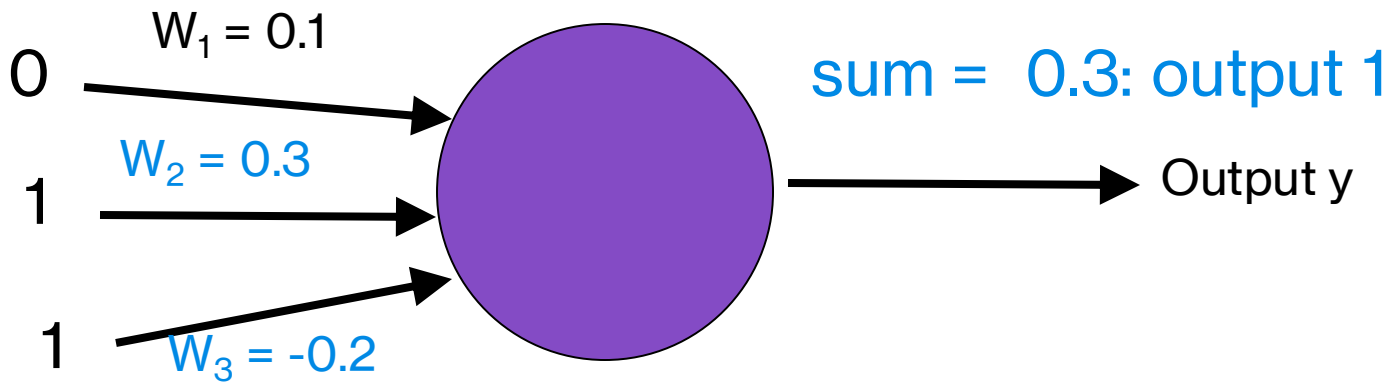
Wrong

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:

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



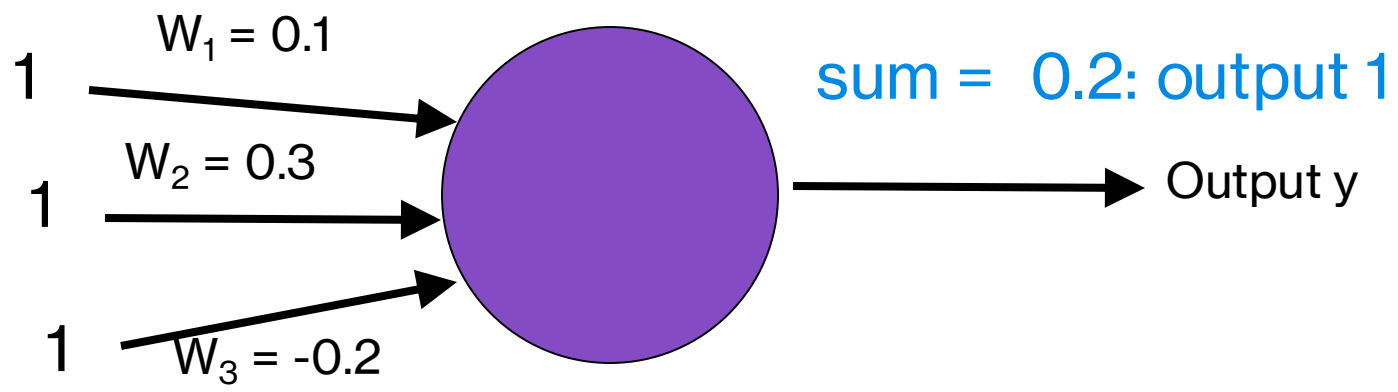
decrease (0-1=-1) all non-zero x_i by 0.1

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:

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



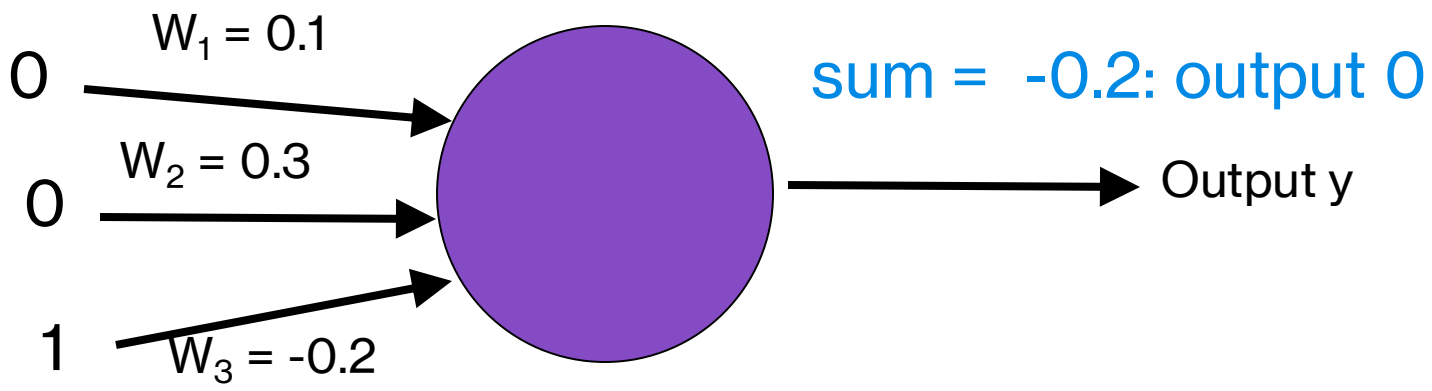
Right. No update!

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:

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



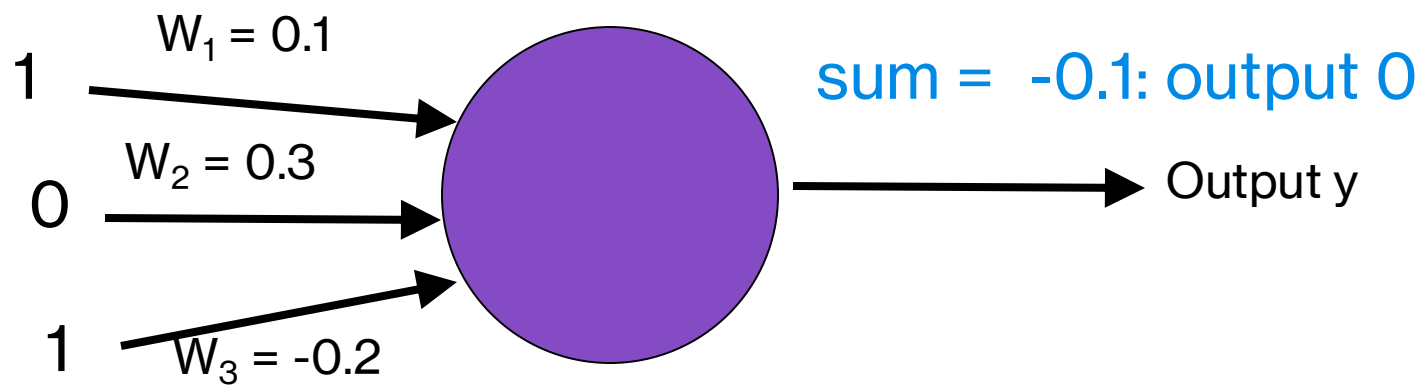
Right. No update!

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:

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



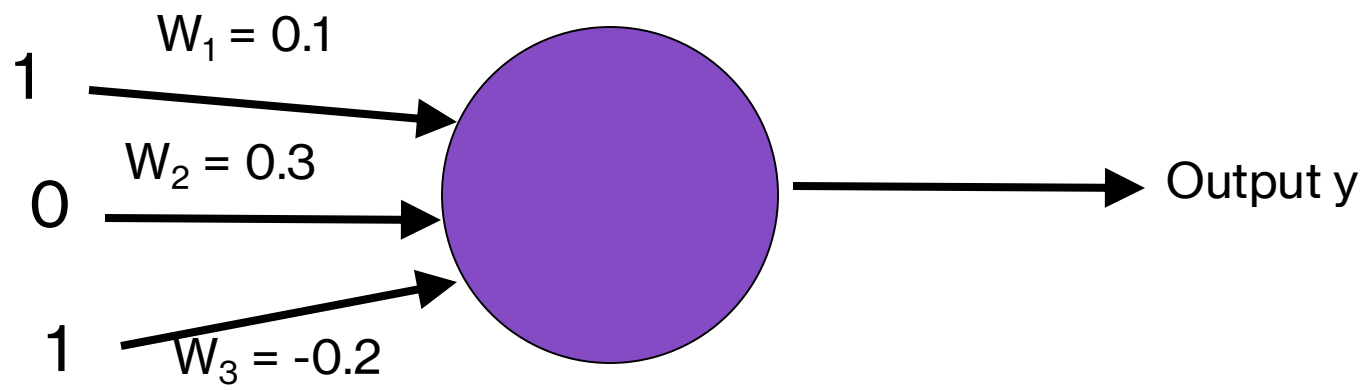
Right. No update!

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:

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



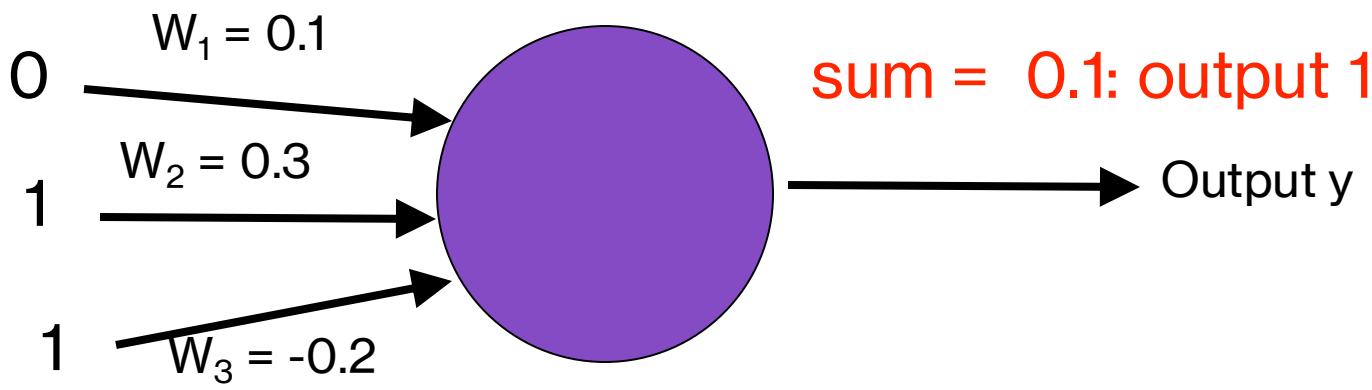
Are they all right?

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:

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



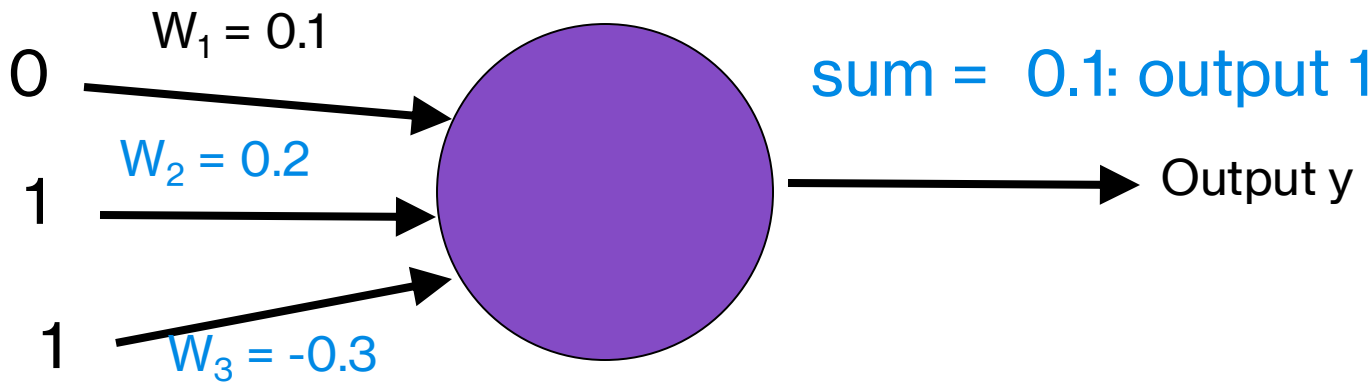
Wrong

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:

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



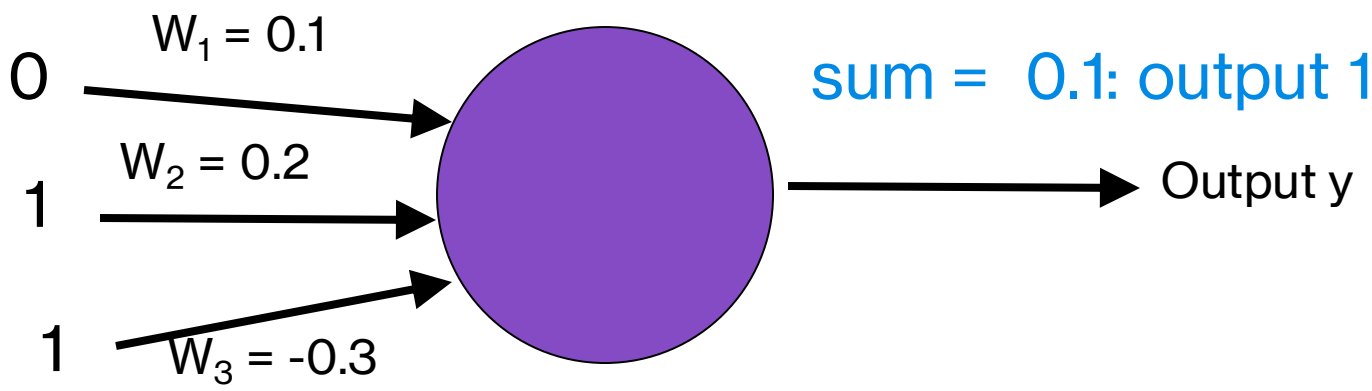
decrease (0-1=-1) all non-zero x_i by 0.1

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:

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



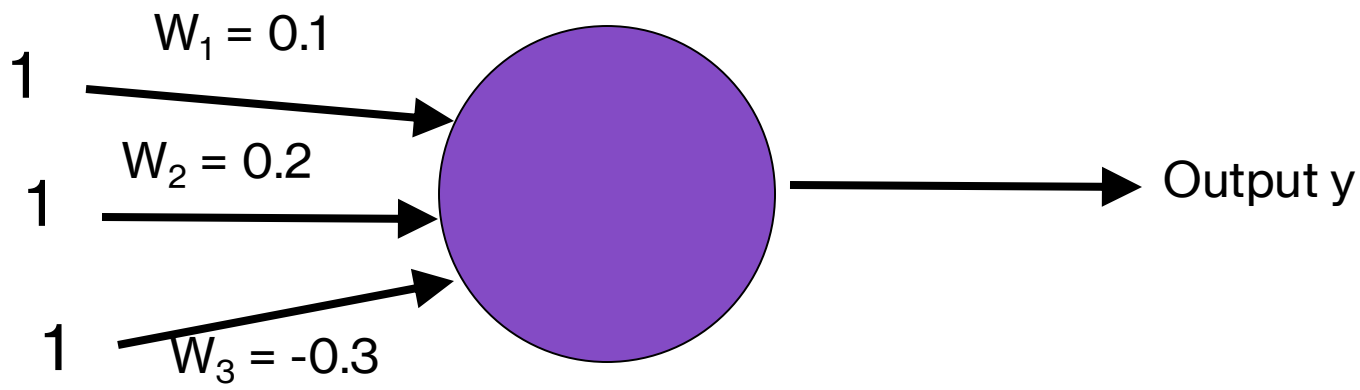
Are they all right?

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)

Final exam

- Friday, Dec. 12 at 2:00-5:00pm (in our classroom)
- Can bring **two** double-sided sheets of hand-written notes
- Cumulative for the entire semester
- Go over all slides and accompanying code/circuits/automata.
- Try to solve the practice time problems before you see the solution.
- Make a pass over all assignments
- If you still have questions, come to special office hours (see slack for announcements)
- Use the following guide as a (non-exhaustive) guide

History and Ethics

- Similar to checkpoint 1

Boolean Algebra

- NOT, AND, OR, XOR opetors
- De Morgan's laws
- DNFs and CNFs
- Minterms and maxterms
- Boolean Algebra in Python

Numeral Systems

- Decimal numbers
- Binary numbers (MSB and LSB)
- Hexadecimal numbers
- Equivalence across all three number systems

Binary Arithmetic and Representing Information

- Addition in decimal, binary, hexadecimal
- Overflow
- Unsigned, signed numbers(magnitude and two's complement representation)
- Equivalence across systems
- Sign extension
- Shifting: logical and arithmetic right, left shifting
- Bitwise operators

Combinational circuits

- Gates and truth tables for NOT, AND, OR, XOR, NAND, NOR
- minterm expansion to find DNF
- Karnaugh maps (see relevant assignment)
- Half- and full-adders
- Decoders
- 7-segment displays

Sequential circuits

- SR-latches
- Registers
- units of memory
- Calculating the size of memory
- Types of memory

Central Processing Unit

- How CPU coordinates with the different registers to read program instructions
- fetch-decode-execute cycle

Call stack

- Know how to visualize the call stack and how different stack frames are pushed and popped (including for recursion later on)

Introduction to Proofs

- Atomic and compound propositions
- Logical connectives including if then and if and only if
- Precedence
- Tautologies
- Satisfiability
- Logical equivalence

Loop invariants

- Know how to state pre- and post-conditions
- How to write a loop invariant
- How to prove correctness of an iterative algorithm:
  - Initialization
  - Maintenance
  - Postcondition
  - Termination

Proof by induction

- Revisit summation notation
- Know how to apply template for proof by induction

Proof by strong induction

- Understand the difference between weak and strong induction
- Floor, ceiling, modulo, prime, composite, divisible
- Apply the template for strong induction

Recursion

- Practice with writing recursive functions on integers, lists, strings

Turtle graphics

- Know basic commands and how they manipulate the turtle

Sequences and dictionaries

- Strings, lists, tuples, ranges
- Indexing
- Slicing
- Iteration
- Membership
- Immutable and mutable
- Methods vs functions
- Aliasing
- Call stack
- Dictionaries

Files and errors

- How to open a file to read it line by line or read all of its lines
- How to write in a file
- Structure of dealing with exceptions

Sorting Algorithms

- Bubble-,selection-, insertion sort
- Know how to run them and their basic properties

Asymptotic Running Analysis

- Big O notation
- Common classes of runtime complexities and what they mean in practice when we double the input size of the problem we try to solve
- Ordering classes of runtime complexities from best to worst
- Understanding how to count comparisons in different sorting algorithms to identify the big O for worst, average, and best case performance

Higher order functions

- Understand that functions are objects and can be passed as parameters, be returned from functions, and created on the fly
- Look into higher_order_functions.py and practice calling its different functions
- Understand how the built-in map and filter work
- Anonymous functions using lambda
- sorted function

DFAs

- Understand that DFAs are finite automata that accept or reject words built out of a specific alphabet and which belong to a specific language
- You need a starting state but can have more than 1 accept state
- For every state, you need to be explicit about what transitions need to happen for each character of the alphabet
- Regular expressions using concatenation, Kleene star, and union

NFAs

- Similar to DFAs but for a state we could have no transition for a character (empty set), one or even more
- There is a way to translate an NFA to DFA
- Regular languages can be described by a DFA/NFA

Turing machines

- More powerful than DFA and NFAs
- Be comfortable with transitions

Machine learning

- Supervised (classification, regression, ranking), unsupervised (clustering), reinforcement learning
- Biological idea behind artificial neural networks
- Neuron/perceptron as a unit that receives inputs with certain weights and gets activated/excited/stimulated if the weighted sum is above threshold

Perceptron training

- Add one extra 1-input, assign random weights to the n+1 inputs, threshold is set at 0.
- Fix learning rate between 0 and 1
- For each training point, if the prediction was wrong, adjust the non-zero input weights by delta.