CPSC 322
Introduction to Artificial Intelligence

December 3, 2004

Things...

Slides for the last couple of weeks will be up this weekend.

Did I mention that you have a final exam at noon on Friday, December 10, in MCML 166?

Check the announcements part of the web page occasionally between now and then in case anything important comes up.

This should be fun

Artificial Intelligence and Interactive Digital Entertainment
First Annual Conference
June 1-3, 2005 in Marina Del Rey, California
www.aiide.org

Sponsors:

AAAI

ELECTRONIC ARTS
Electronic Arts

About the Event

AIIDE is intended to be the definitive point of interaction between entertainment software developers interested in AI and academic and industrial AI researchers. Sponsored by the American Association for Artificial Intelligence (AAAI), the conference is targeted at both the research and commercial communities, promoting AI research and practice in the context of interactive digital entertainment systems with an emphasis on commercial computer and video games.

Reinforcement learning

Learning operator sequences based on reward or punishment

Let's say you want your robot to learn how to vacuum the living room

iRobot Roomba 4210 Discovery Floorvac Robotic Vacuum $249.99
It's the ideal Christmas gift!

Reinforcement learning

Learning operator sequences based on reward or punishment

Let's say you want your robot to learn how to vacuum the living room

```
goto livingroom
vacuum floor
goto trashcan
empty bag
```

Good Robot!

```
goto trashcan
vacuum floor
goto livingroom
empty bag
```

Bad Robot!

(actually, there are no bad robots, there is only bad behavior...and Roomba can't really empty its own bag)

Reinforcement learning

Should the robot learn from success? How does it figure out which part of the sequence of actions is right (credit assignment problem)?

```
goto livingroom
vacuum floor
goto trashcan
empty bag
```

Good Robot!

```
goto trashcan
vacuum floor
goto livingroom
empty bag
```

Bad Robot!

Reinforcement learning

Should the robot learn from failure? How does it figure out which part of the sequence of actions is wrong (blame assignment problem)?

```
goto livingroom
vacuum floor
goto trashcan
empty bag
```

Good Robot!

```
goto trashcan
vacuum floor
goto livingroom
empty bag
```

Bad Robot!

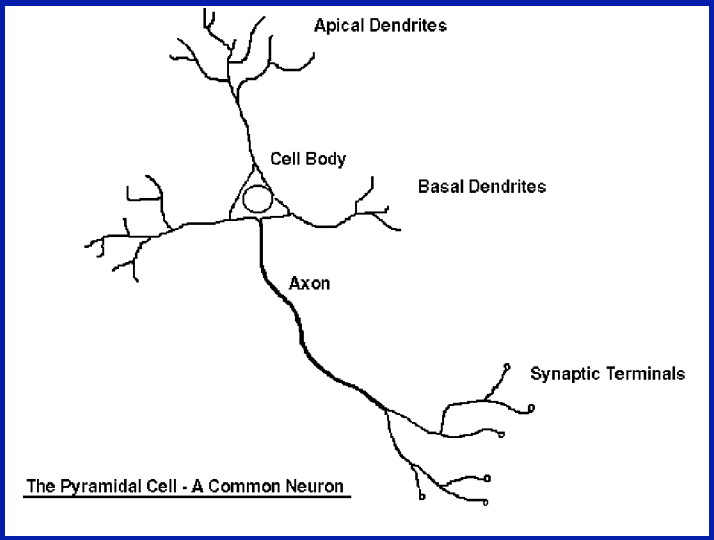
Reinforcement learning

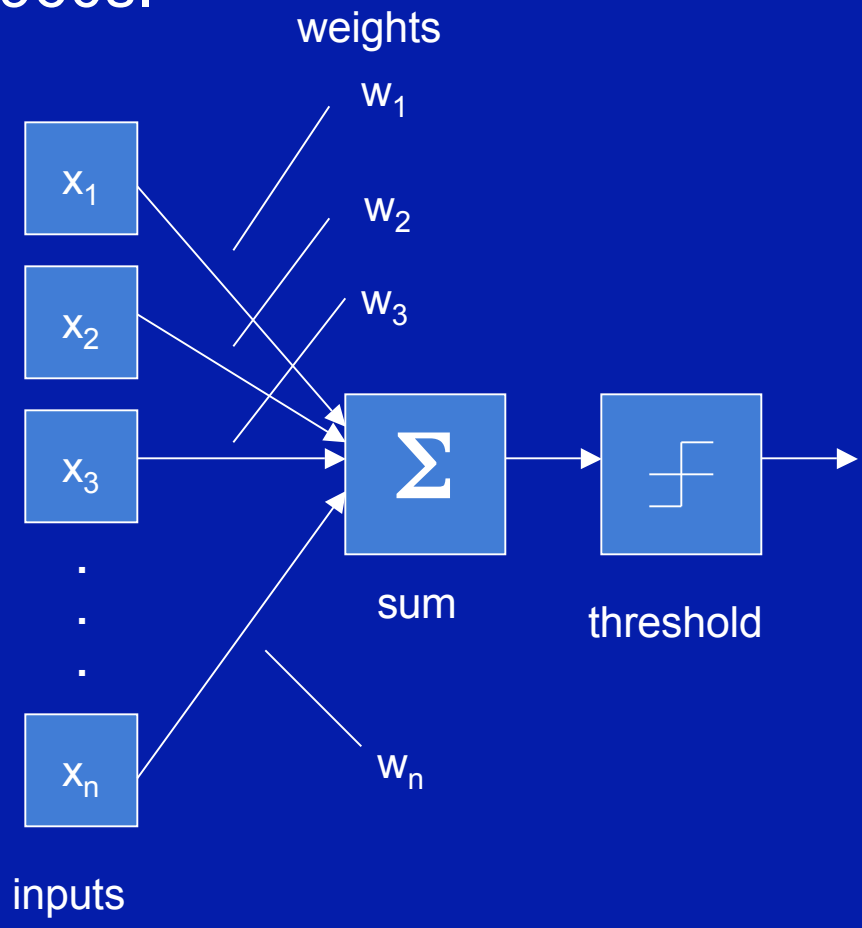
However you answer those questions, the learning task again boils down to the search for the best representation.

Here's another type of learning that searches for the best representation, but the representation is very different from what you've seen so far.

Learning in neural networks

The perceptron is one of the earliest neural network models, dating to the early 1960s.

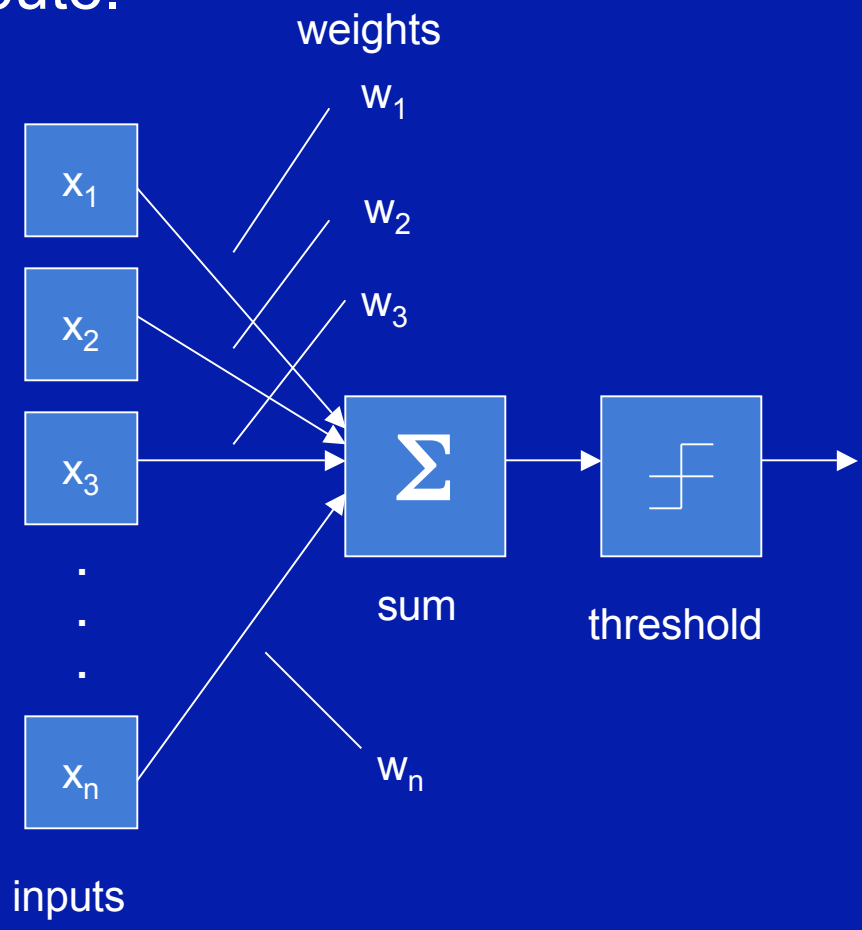




Learning in neural networks

The perceptron can't compute everything, but what it can compute it can learn to compute.
Here's how it works.

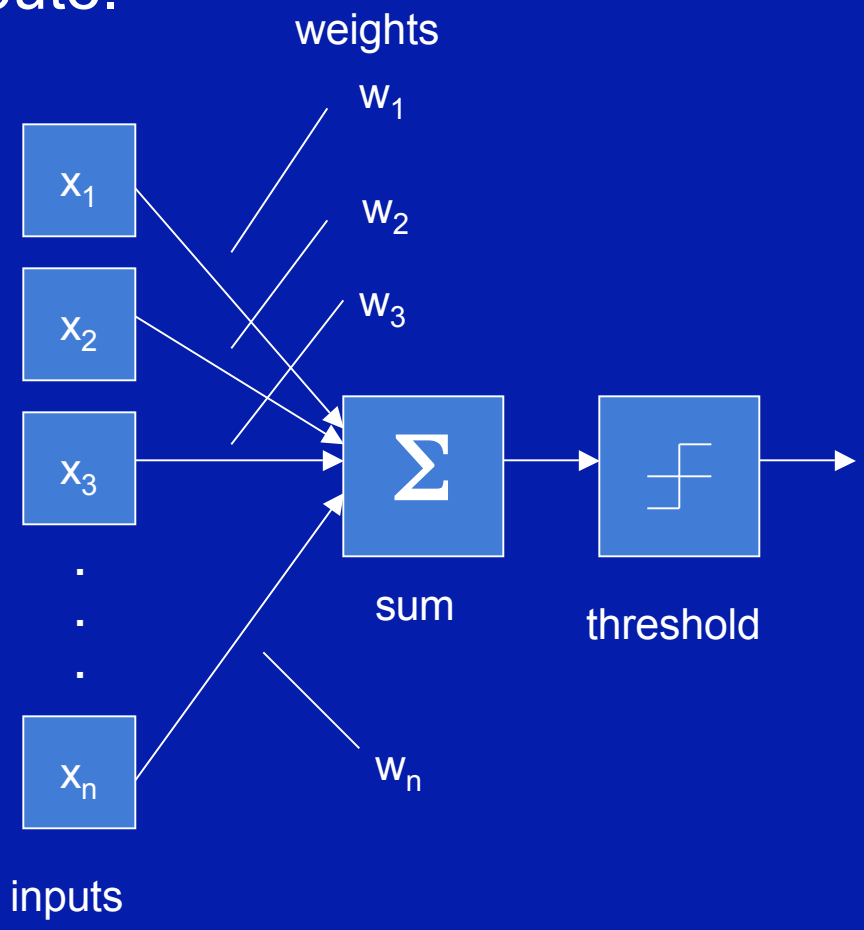
Inputs are 1 or 0.
Weights are reals (-n to +n).
Each input is multiplied by its corresponding weight.
If the sum of the products is greater than the threshold, then the perceptron outputs 1, otherwise the output is 0.



Learning in neural networks

The perceptron can't compute everything, but what it can compute it can learn to compute. Here's how it works.

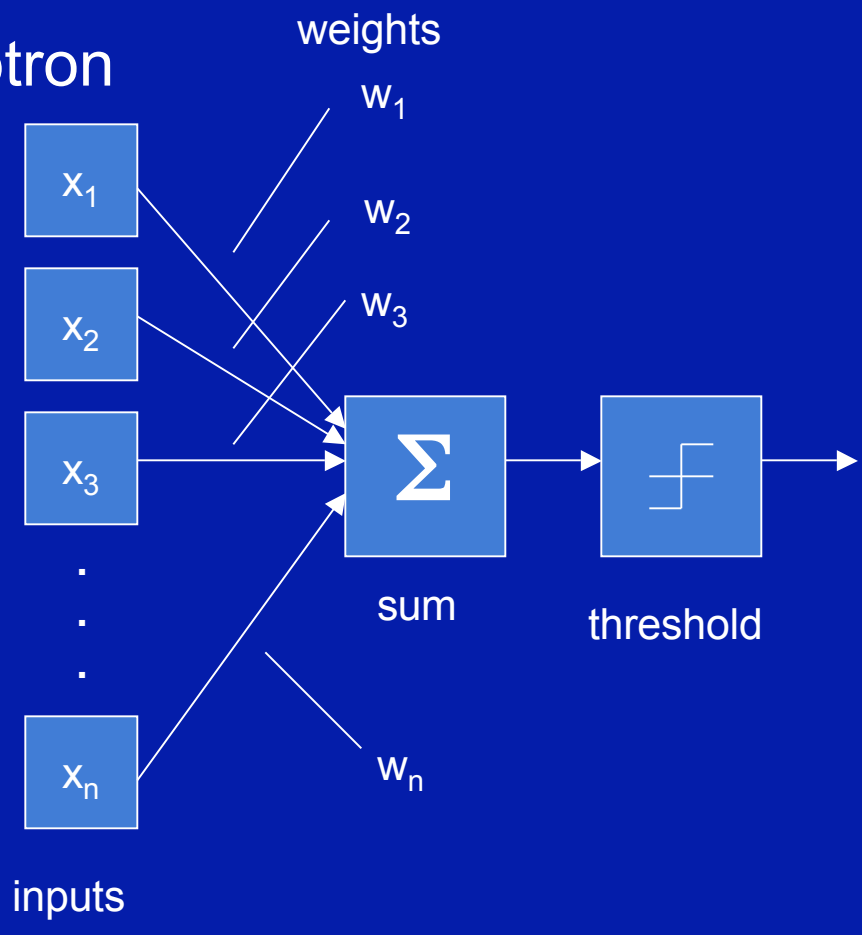
The output, 1 or 0, is a guess or prediction about the input: does it fall into the desired classification (output = 1) or not (output = 0)?



Learning in neural networks

That's it? Big deal. No, there's more to it....

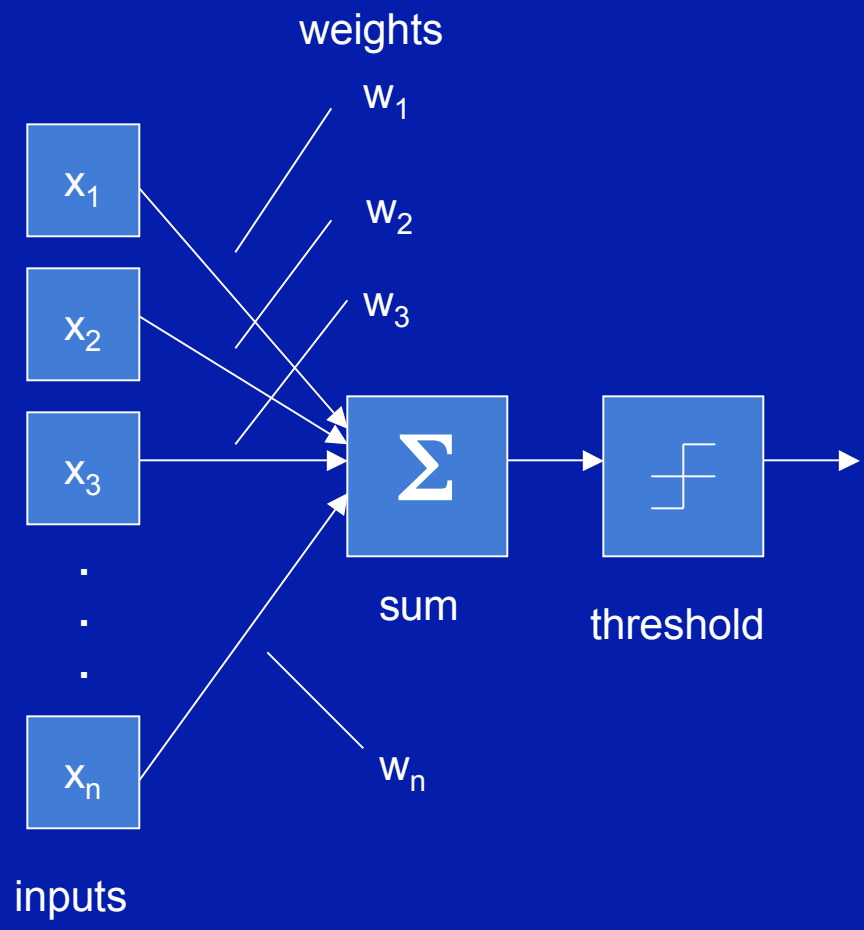
Say you wanted your perceptron to classify arches.
That is, you present inputs representing an arch, and the output should be 1.
You present inputs not representing an arch, and the output should be 0. If your perceptron does that correctly for all inputs, it knows the concept of arch.



Learning in neural networks

But what if you present inputs for an arch, and your perceptron outputs a 0??? What could be done to make it more likely that the output will be 1 the next time the ‘tron sees those same inputs?

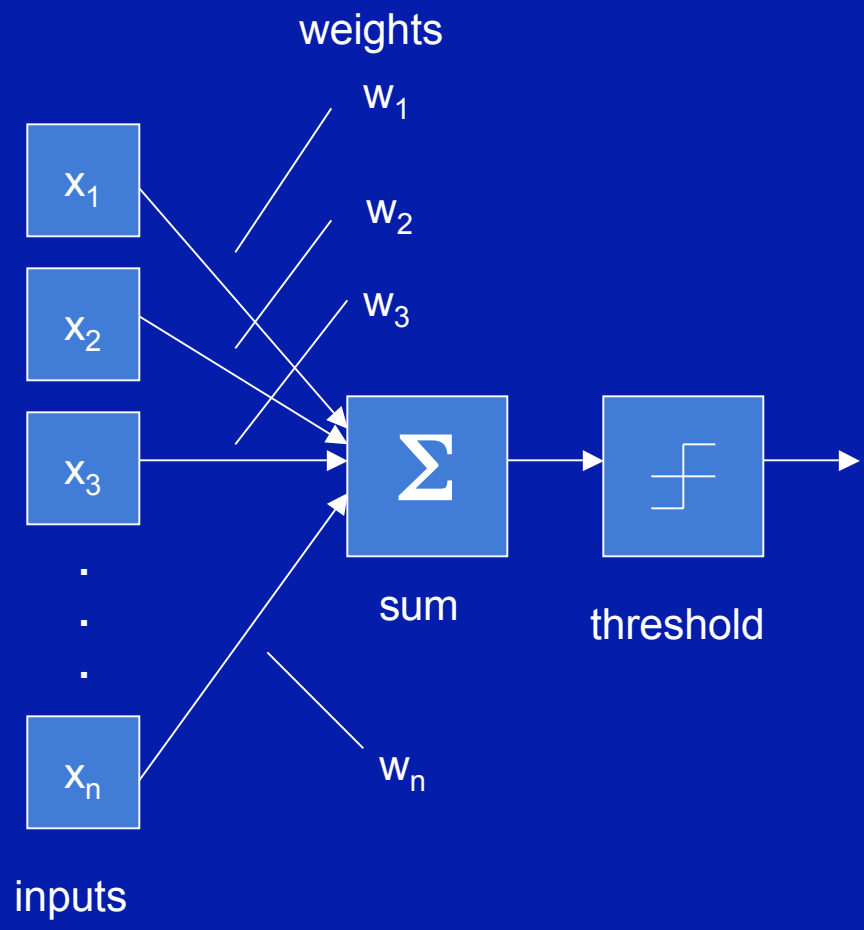
You increase the weights. Which ones? How much?



Learning in neural networks

But what if you present inputs for not an arch, and your perceptron outputs a 1? What could be done to make it more likely that the output will be 0 the next time the ‘tron sees those same inputs?

You decrease the weights. Which ones? How much?



Let’s make one...

First we need to come up with a representation language.
We’ll abstract away most everything to make it simple.

Let’s make one...

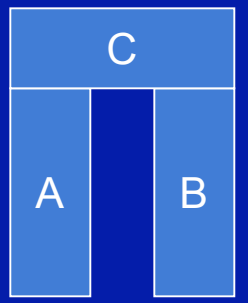
First we need to come up with a representation language. We’ll abstract away most everything to make it simple.

All training examples have three blocks.
A and B are upright blocks. A is always left of B.
C is a sideways block. Our language will assume those things always to be true. The only things our language will represent are the answers to these five questions...

Let’s make one...

yes = 1, no = 0

Does A support C?	1
Does B support C?	1
Does A touch C?	1
Does B touch C?	1
Does A touch B?	0

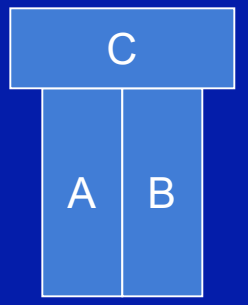


arch

Let’s make one...

yes = 1, no = 0

Does A support C?	1
Does B support C?	1
Does A touch C?	1
Does B touch C?	1
Does A touch B?	1

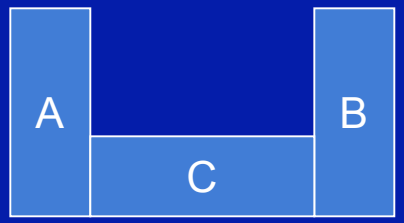


not arch

Let’s make one...

yes = 1, no = 0

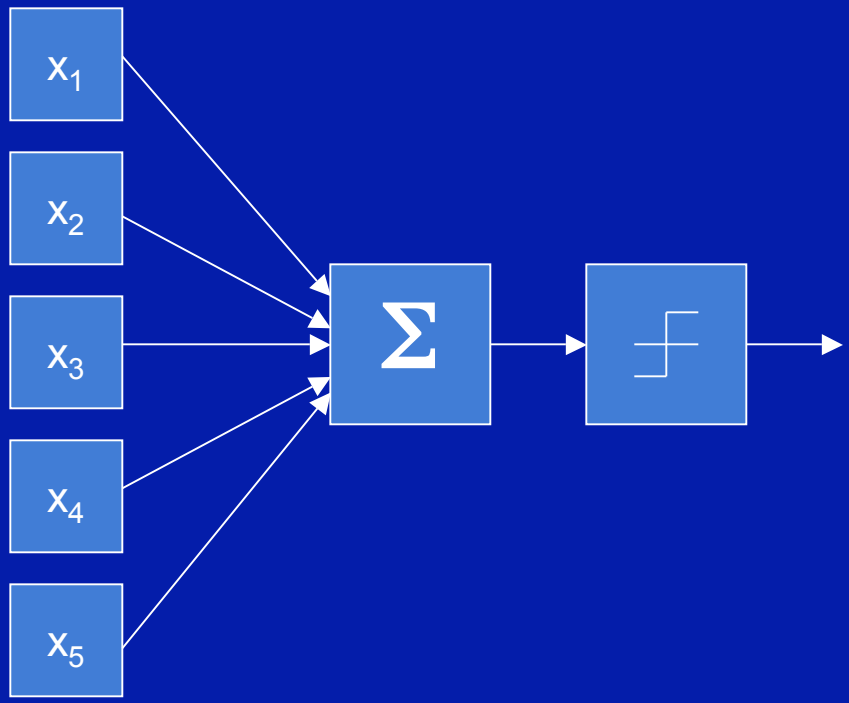
Does A support C?	0
Does B support C?	0
Does A touch C?	1
Does B touch C?	1
Does A touch B?	0



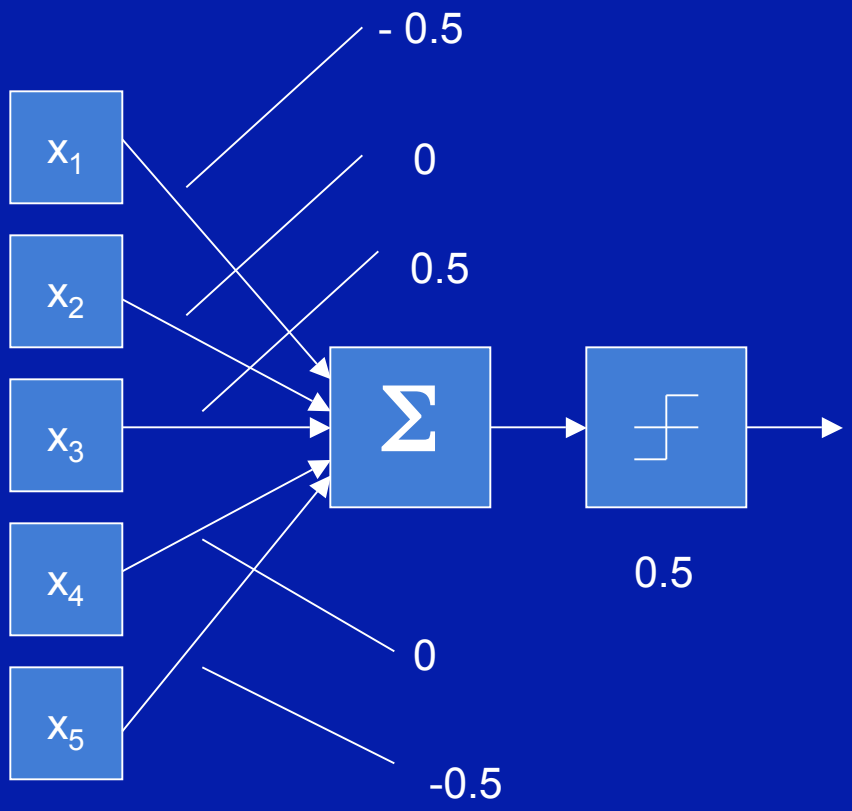
not arch

and so on.....

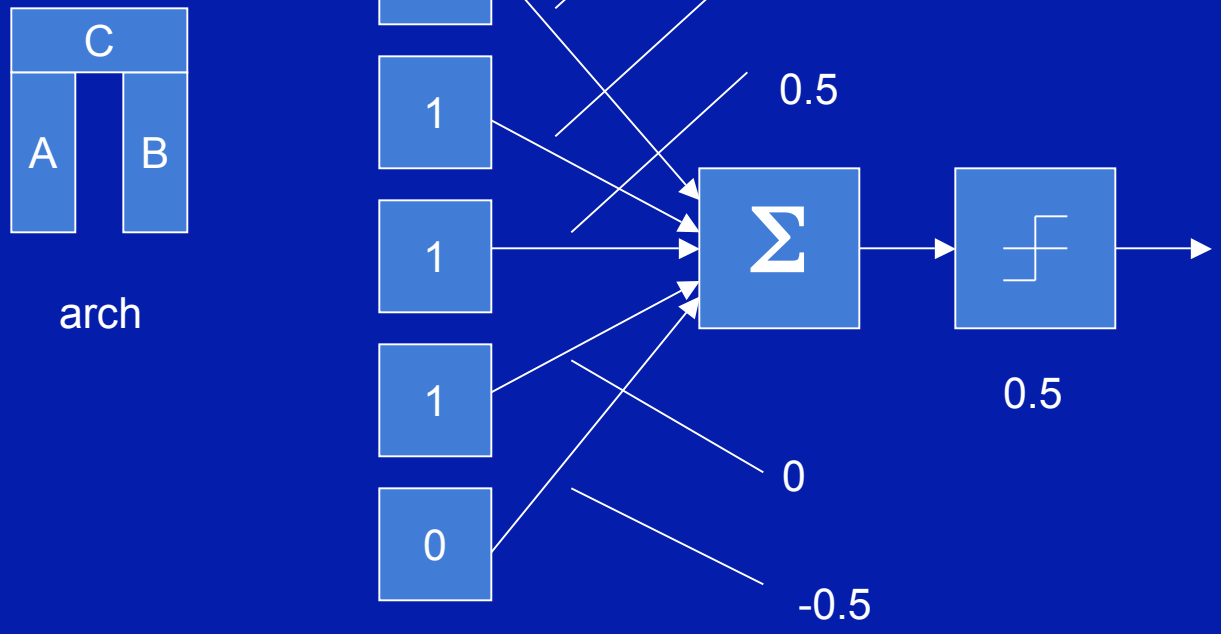
Our very simple arch learner



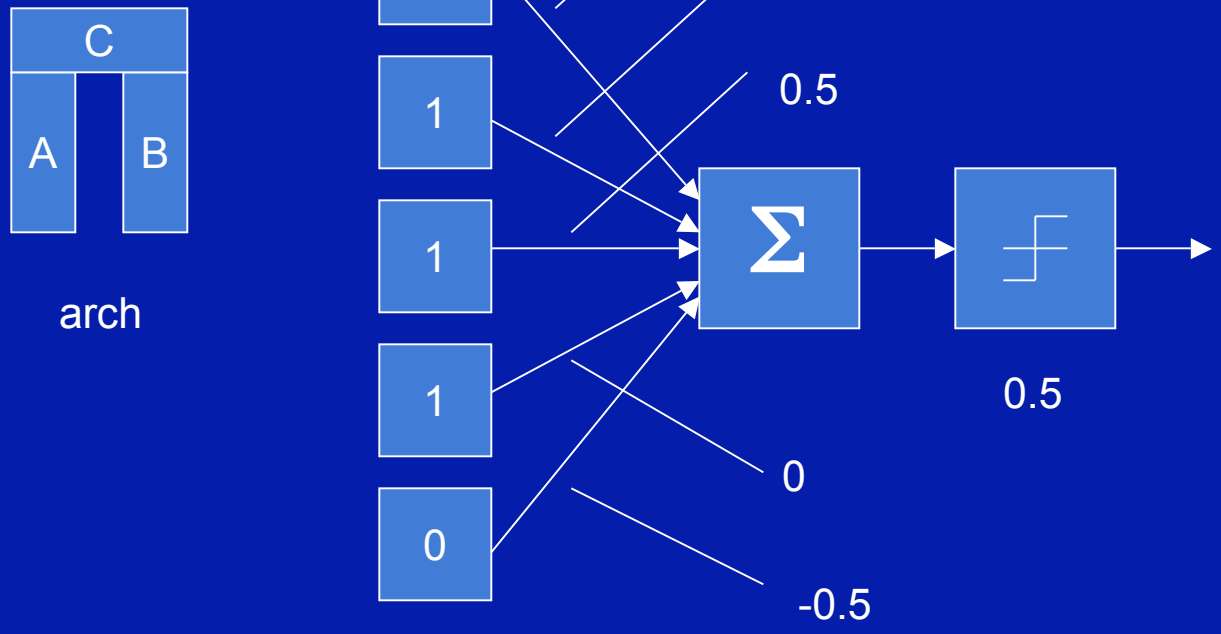
Our very simple arch learner



Our very simple arch learner

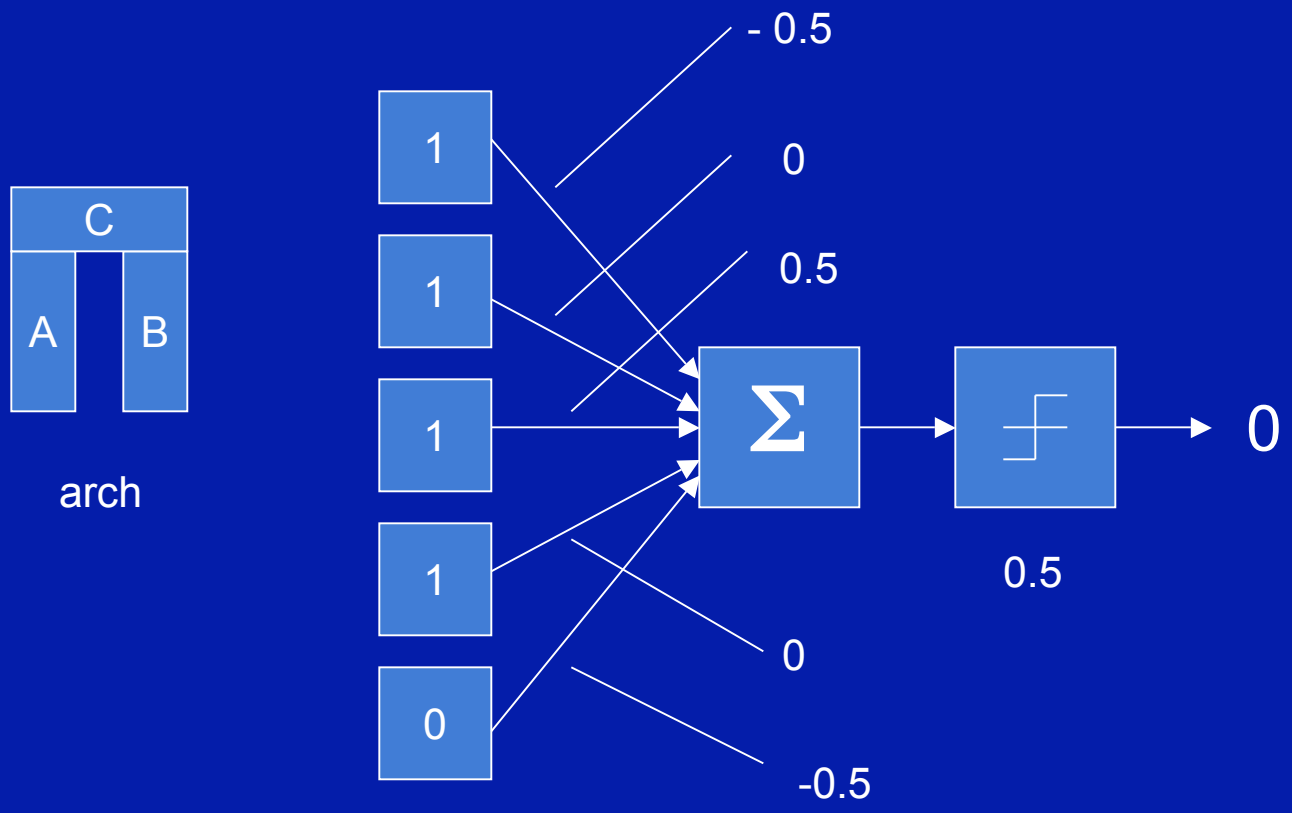


Our very simple arch learner



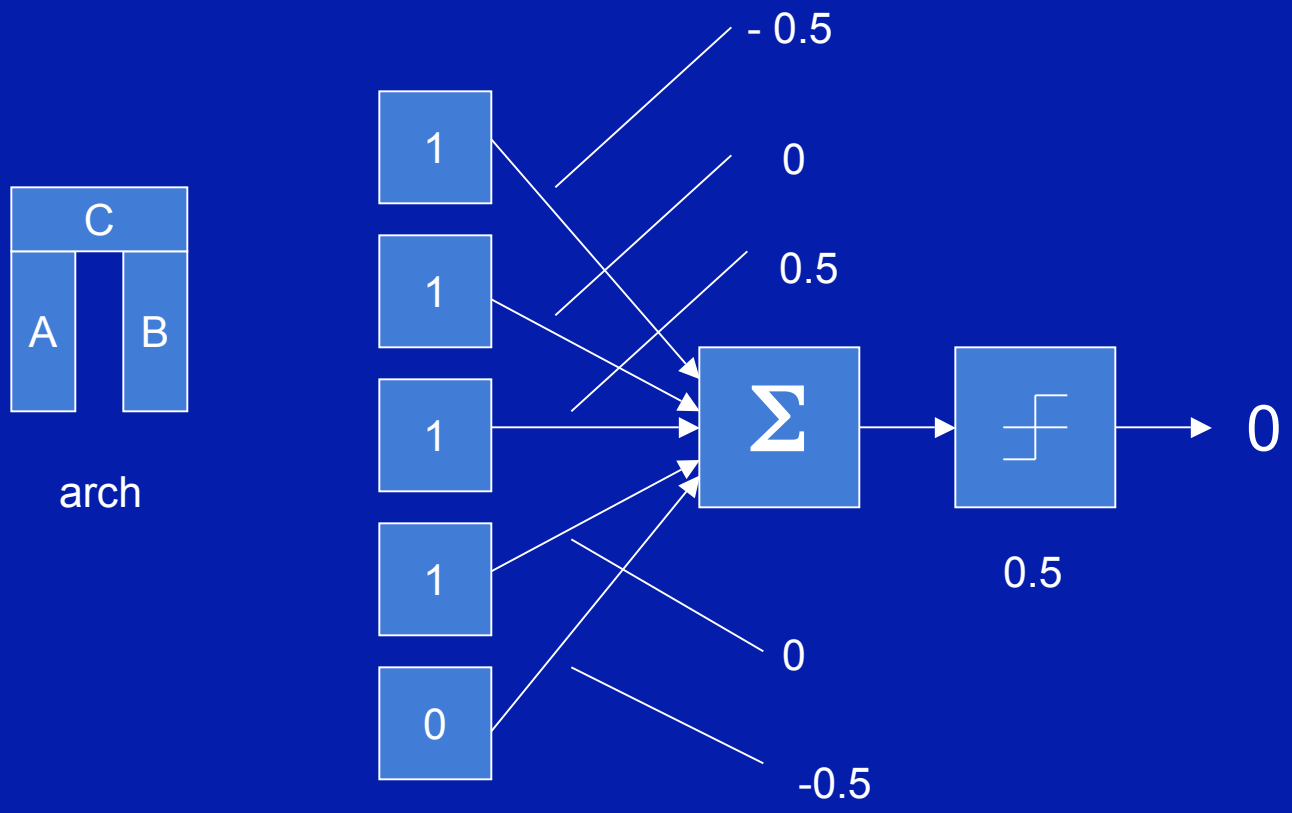
sum = 1*-0.5 + 1*0 + 1*0.5 + 1*0 + 0*0.5

Our very simple arch learner



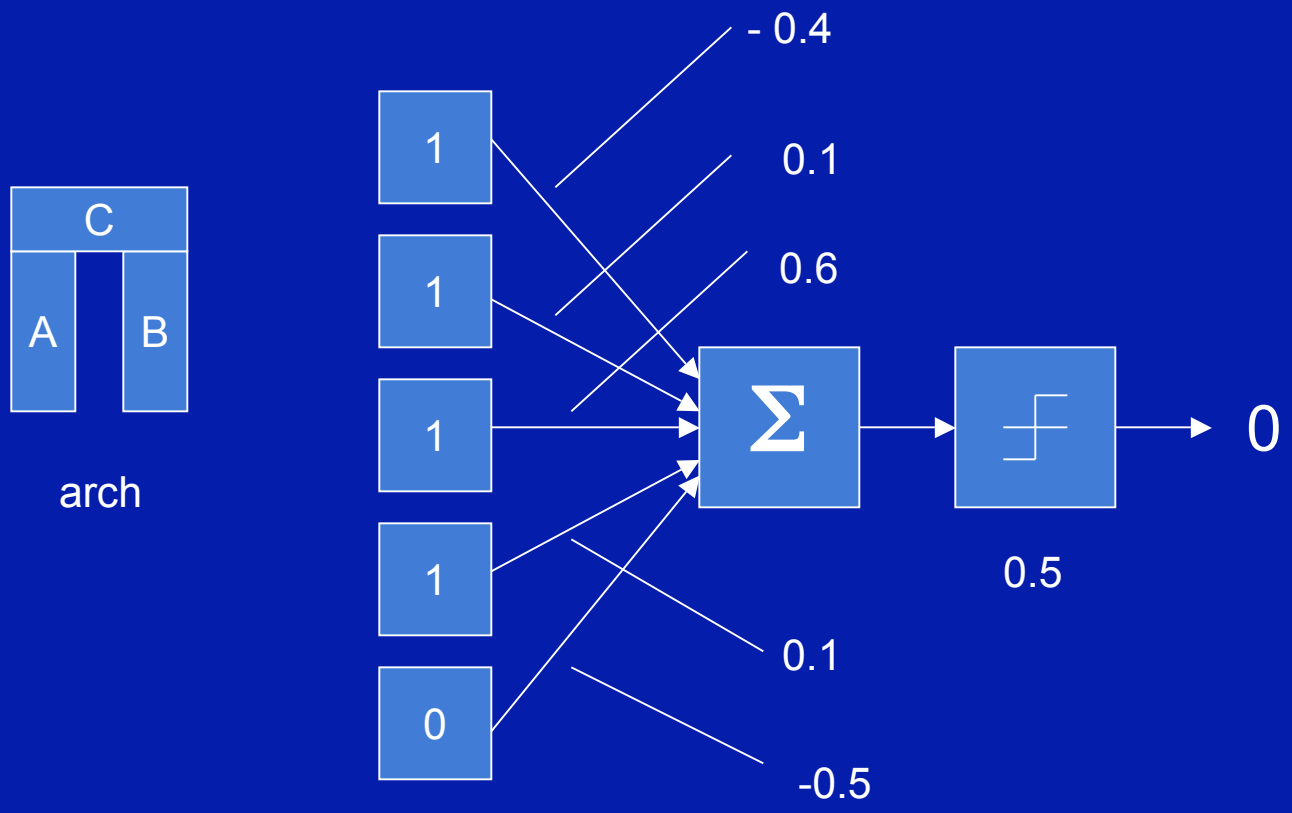
sum = -0.5 + 0 + 0.5 + 0 + 0 = 0 which is not > threshold so output is 0

Our very simple arch learner



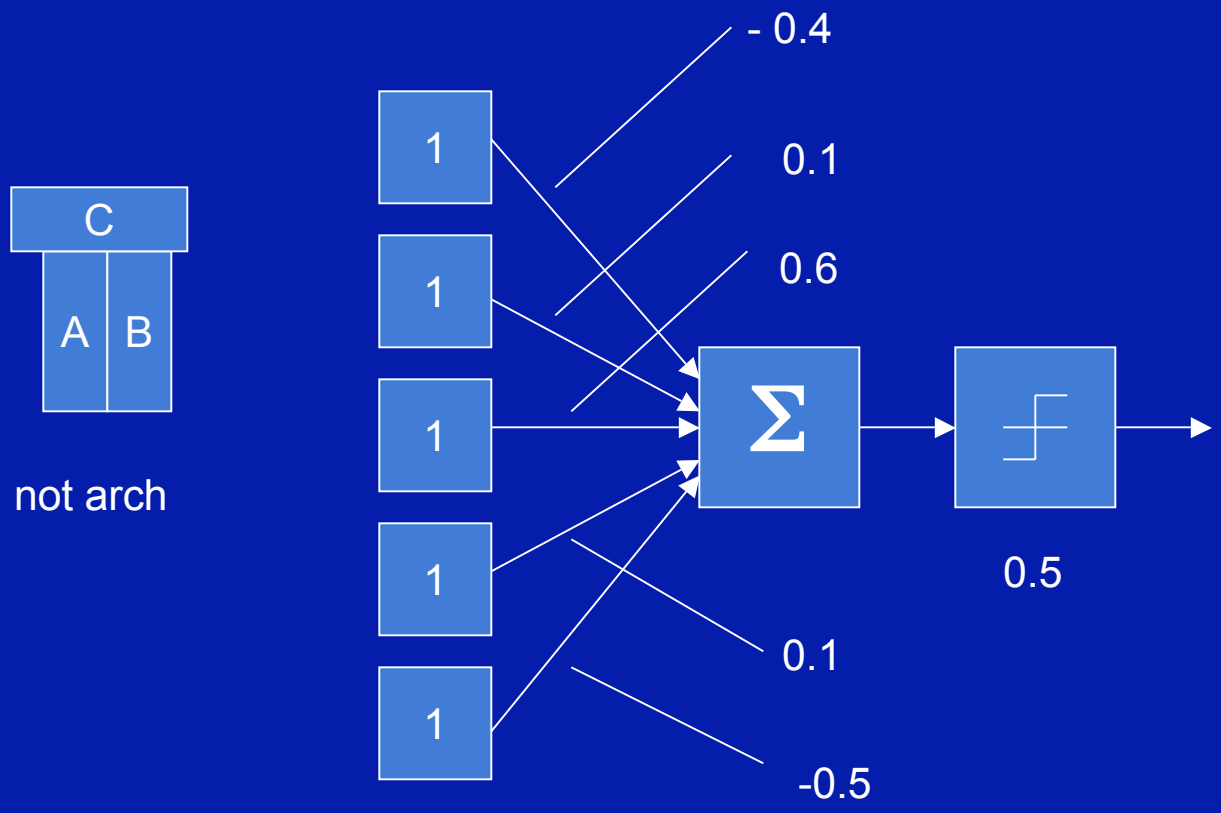
sum = -0.5 + 0 + 0.5 + 0 + 0 = 0 which is not > threshold so output is 0
‘tron said no when it should say yes so increase weights where input = 1

Our very simple arch learner



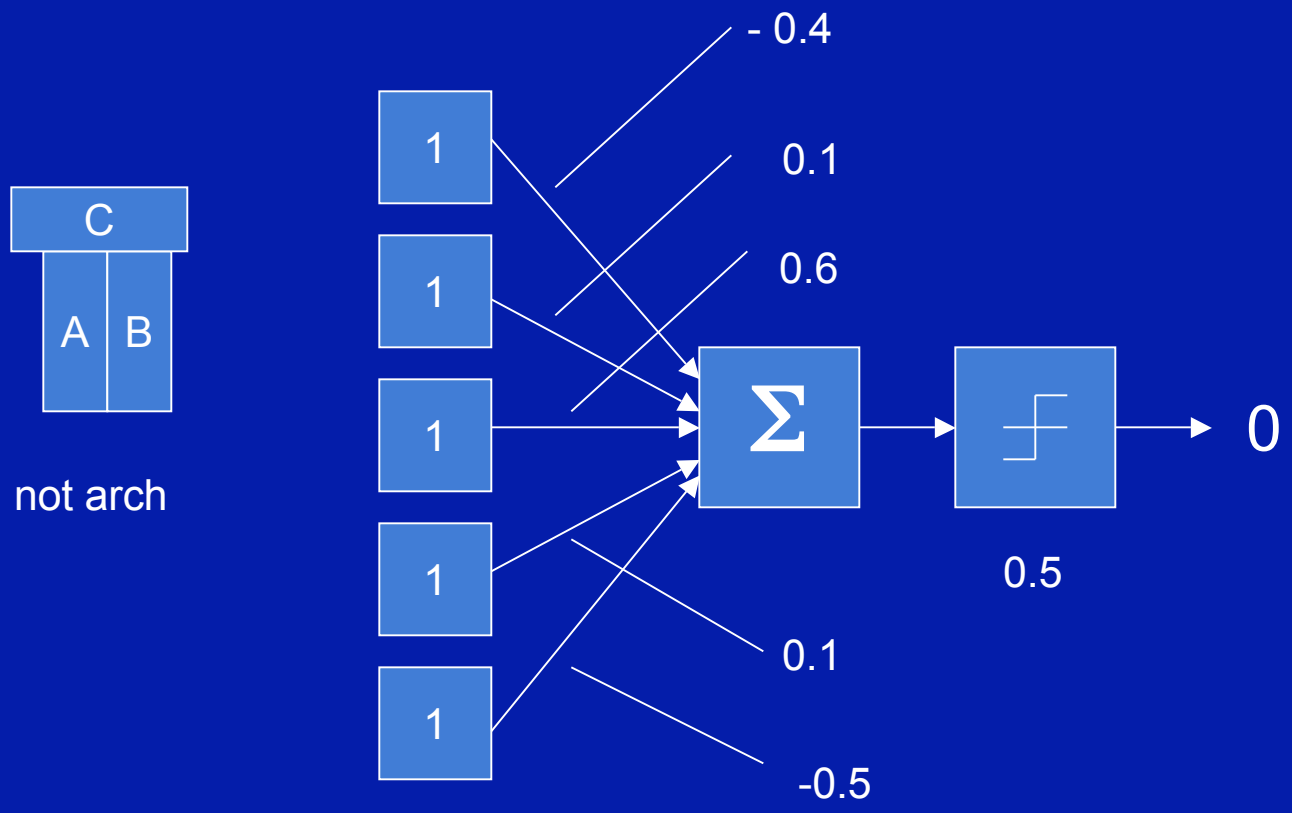
sum = -0.5 + 0 + 0.5 + 0 + 0 = 0 which is not > threshold so output is 0
so we increase the weights where the input is 1

Our very simple arch learner



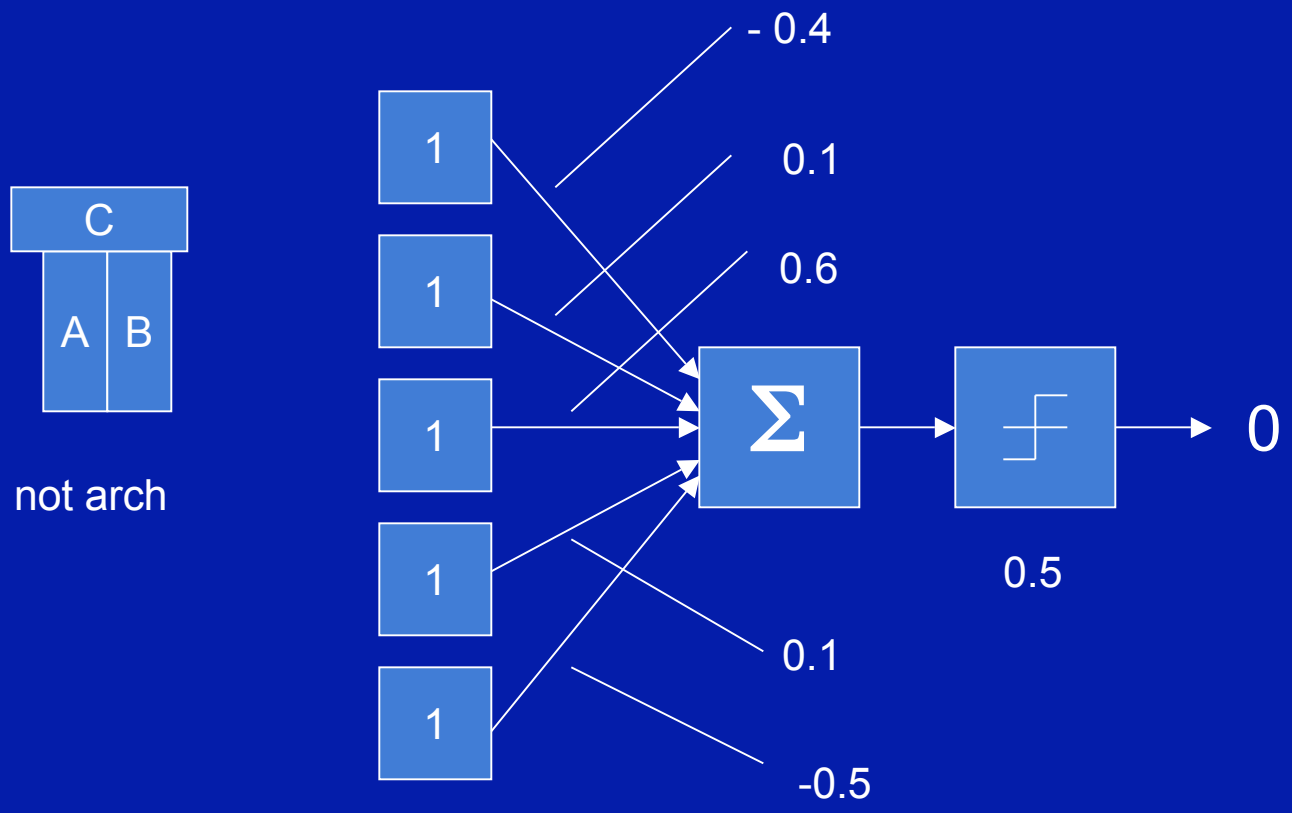
now we look at the next example

Our very simple arch learner



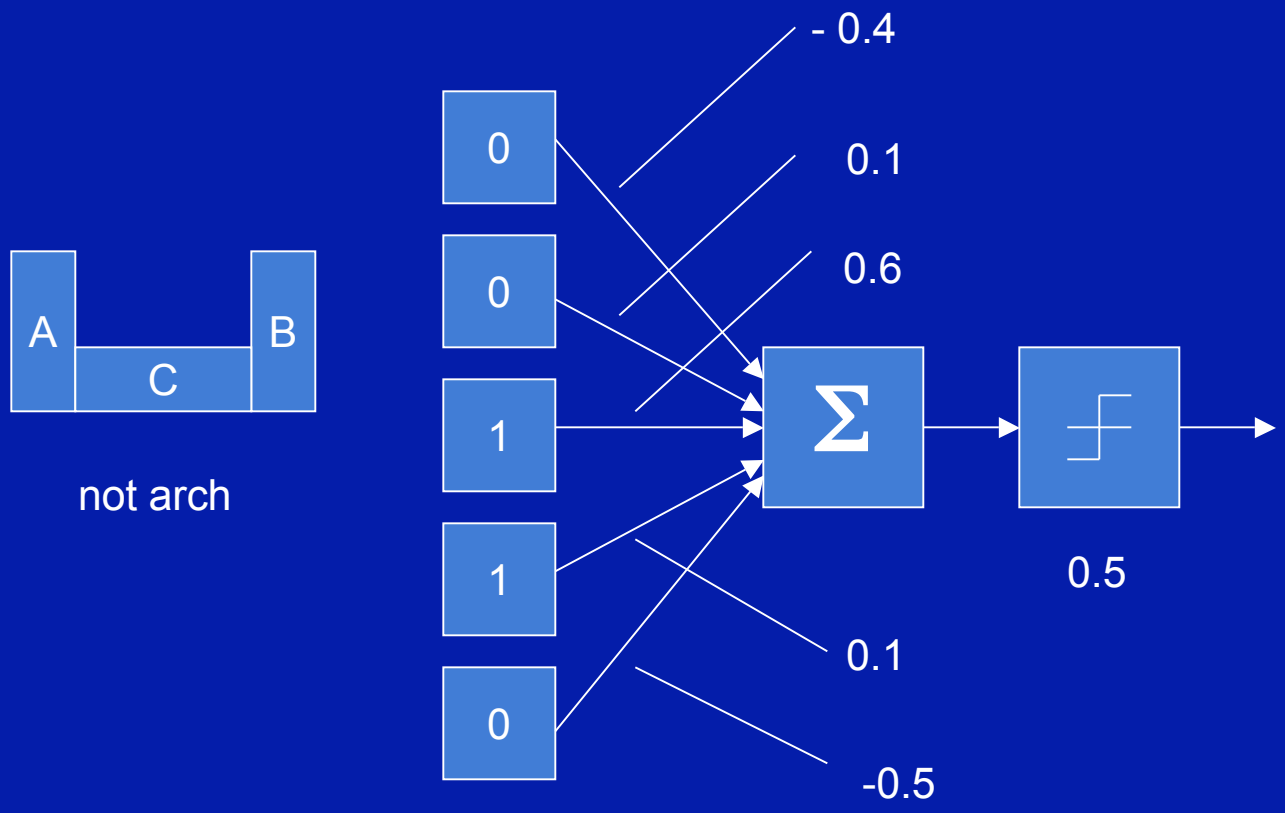
sum = -0.4 + 0.1 + 0.6 + 0.1 - 0.5 = -0.1 which is not > 0.5 so output is 0

Our very simple arch learner



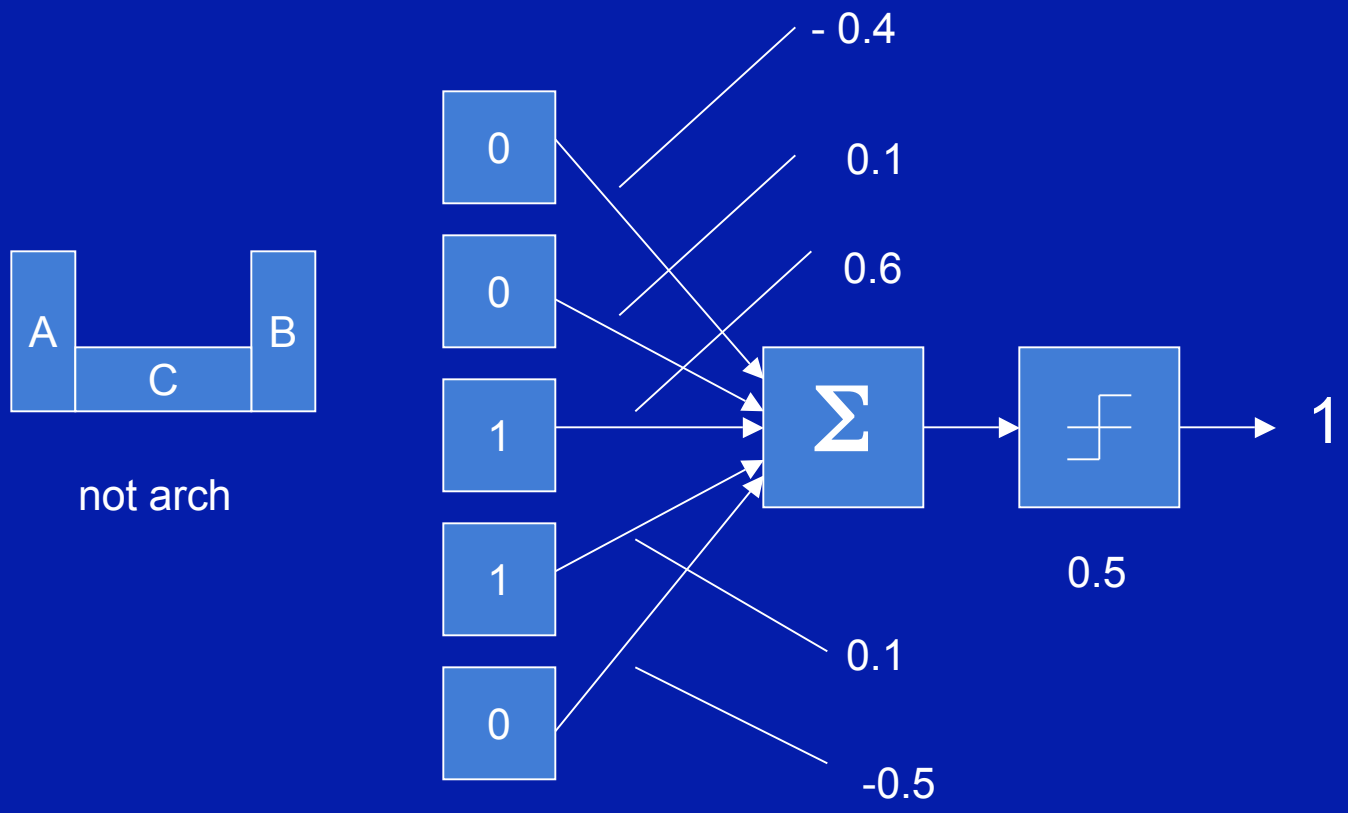
sum = -0.4 + 0.1 + 0.6 + 0.1 - 0.5 = -0.1 which is not > 0.5 so output is 0
that's the right output for this input, so we don't touch the weights

Our very simple arch learner



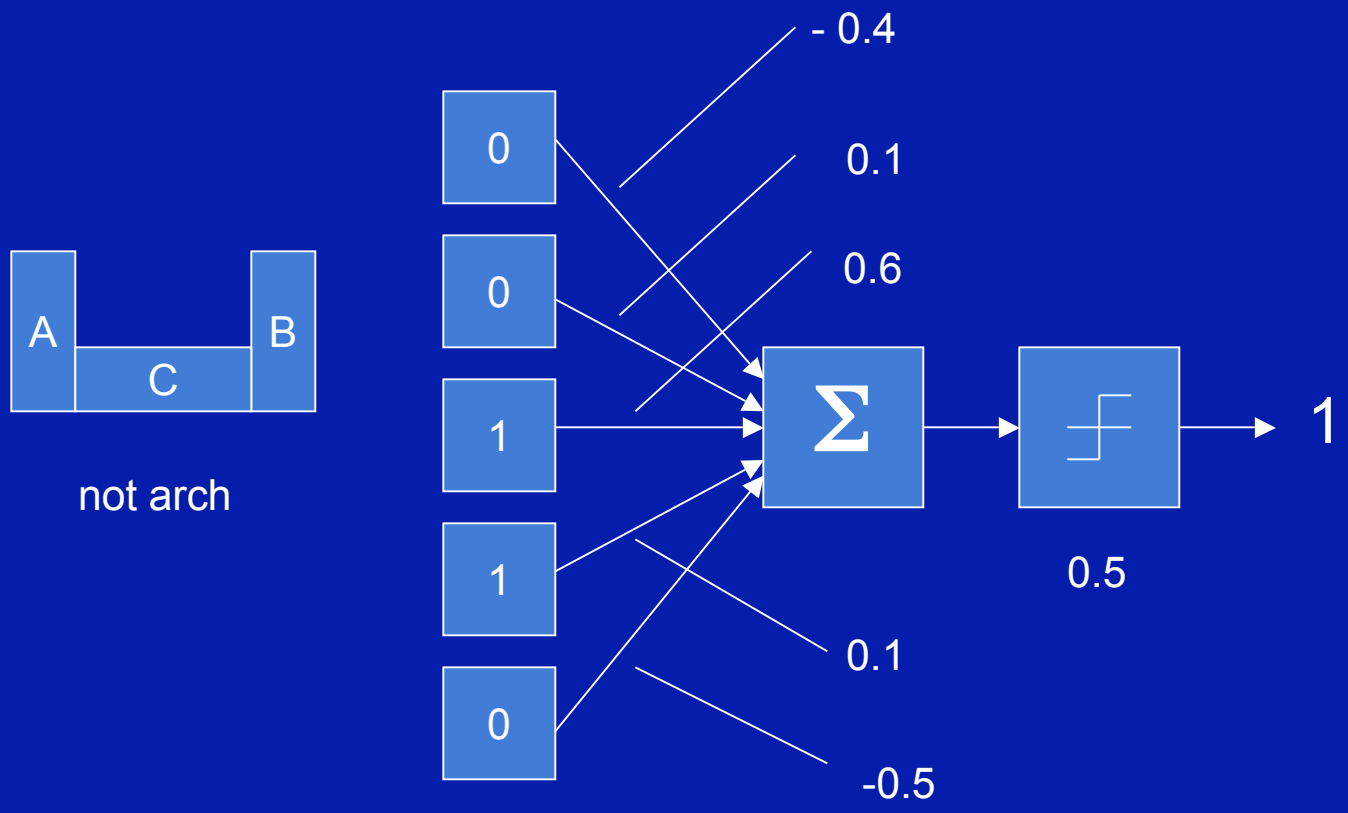
now we look at the next example

Our very simple arch learner



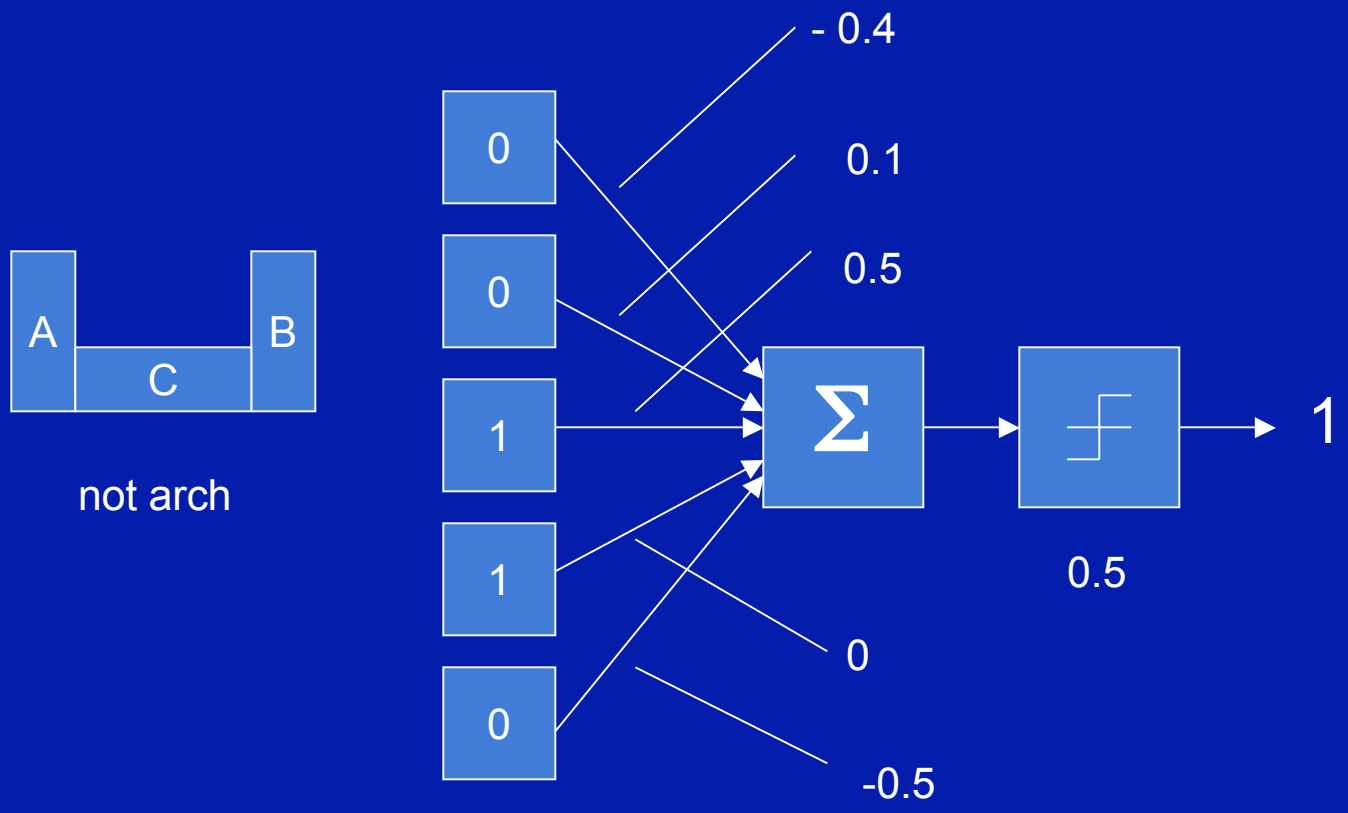
sum = 0 + 0 + 0.6 + 0.1 + 0 = 0.7 which is > 0.5 so output = 1

Our very simple arch learner



the ‘tron said yes when it should have said no, so we decrease the weights where the inputs = 1

Our very simple arch learner



the ‘tron said yes when it should have said no, so we decrease the weights where the inputs = 1

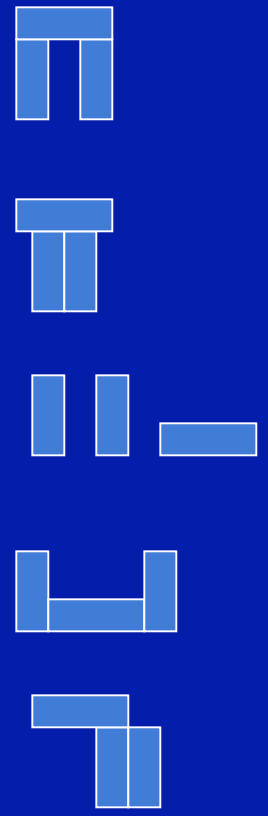
We could do this for days...

...but let’s have the computer do it all for us.

First, take a look at the training examples we’ve constructed...

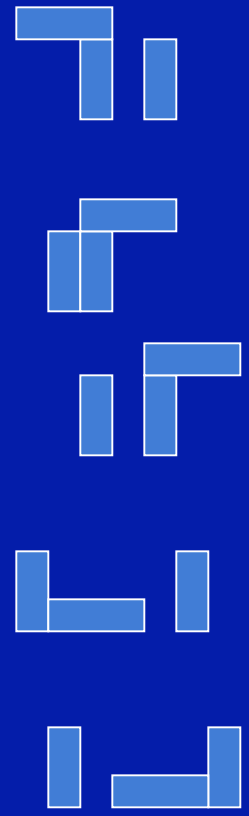
Training Set

in class?	a supports c	b supports c	a touches c	b touches c	a touches b
yes	1	1	1	1	0
no	1	1	1	1	1
no	0	0	0	0	0
no	0	0	1	1	0
no	1	0	1	0	1



Training Set

in class?	a supports c	b supports c	a touches c	b touches c	a touches b
no	1	0	1	0	0
no	0	1	0	1	1
no	0	1	0	1	0
no	0	0	1	0	0
no	0	0	0	1	0



Now let's look at the program

"By the almighty powers of Scheme, I command thee to learn the concept of an arch!"

What's going on?

The perceptron goes through the training set, making a guess for each example and comparing it to the actual answer.

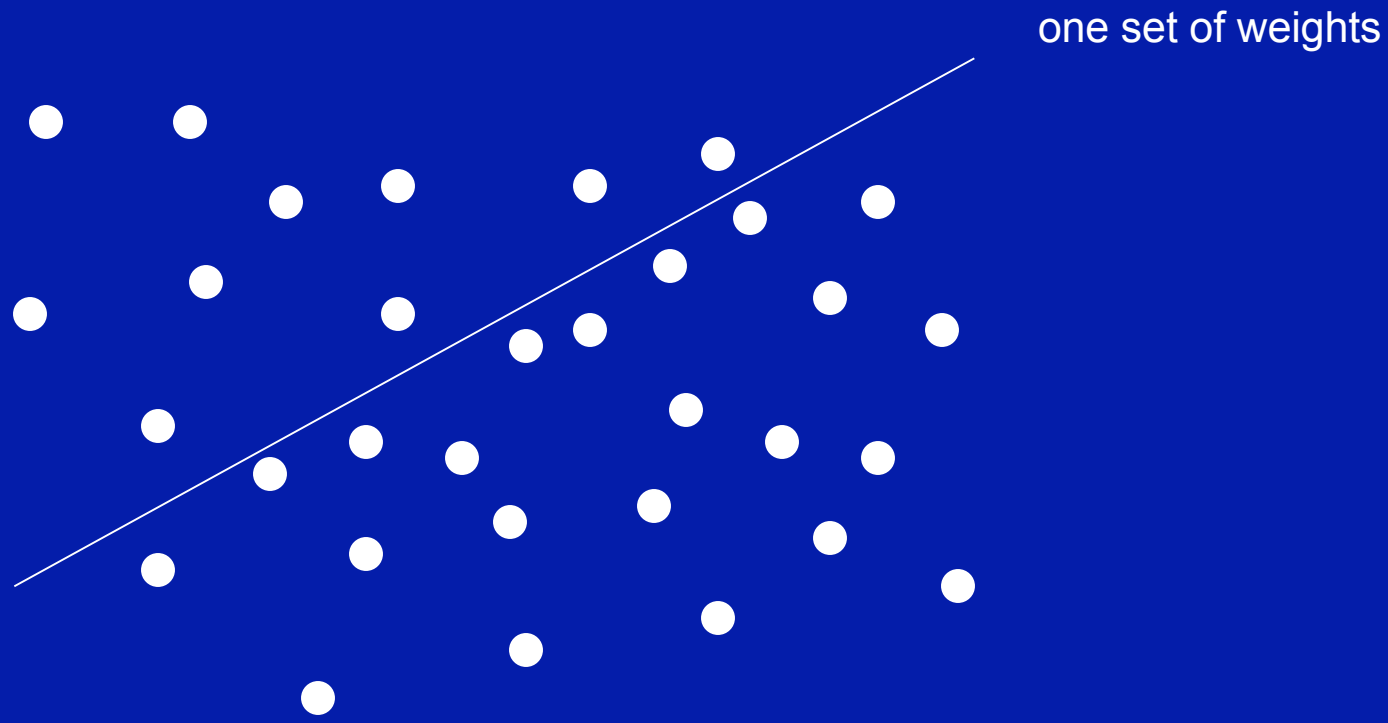
Based on that comparison, the perceptron adjusts weights up, down, or not at all.

For some concepts, the process converges on a set of weights such that the perceptron guesses correctly for every example in the training set -- that's when the weights stop changing.

What's going on?

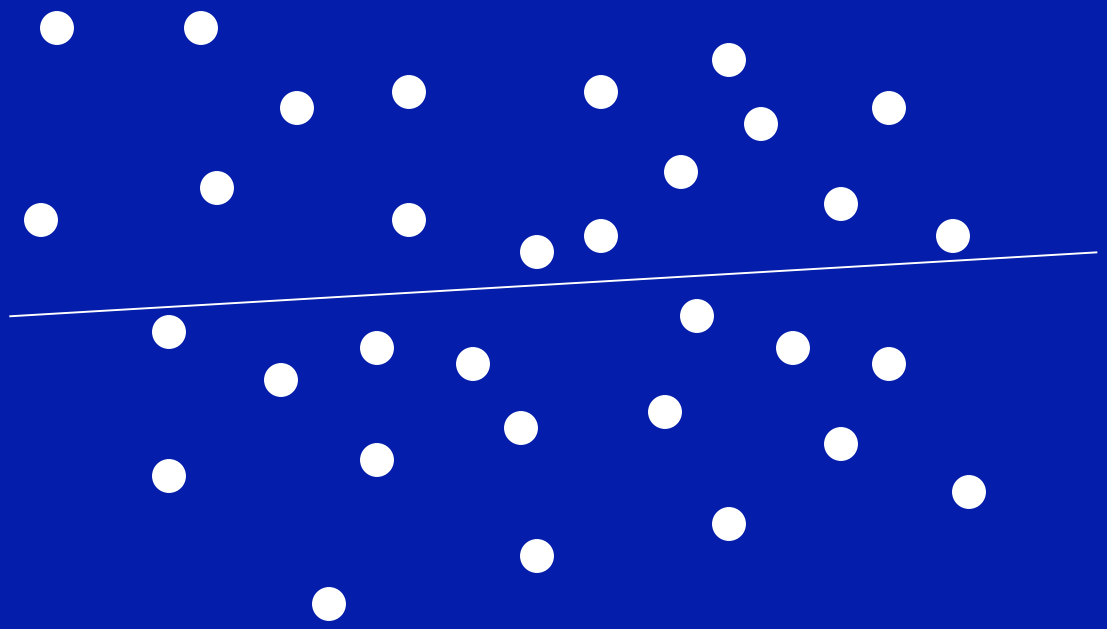
Another way of looking at it:
Each of the possible inputs (2^5 in our case) maps onto a 1 or a 0. The perceptron is trying to find a set of weights such that it can draw a line through the set of all inputs and say "these inputs belong on one side of the line (output = 1) and those belong on the other side (output = 0)".

What's going on?



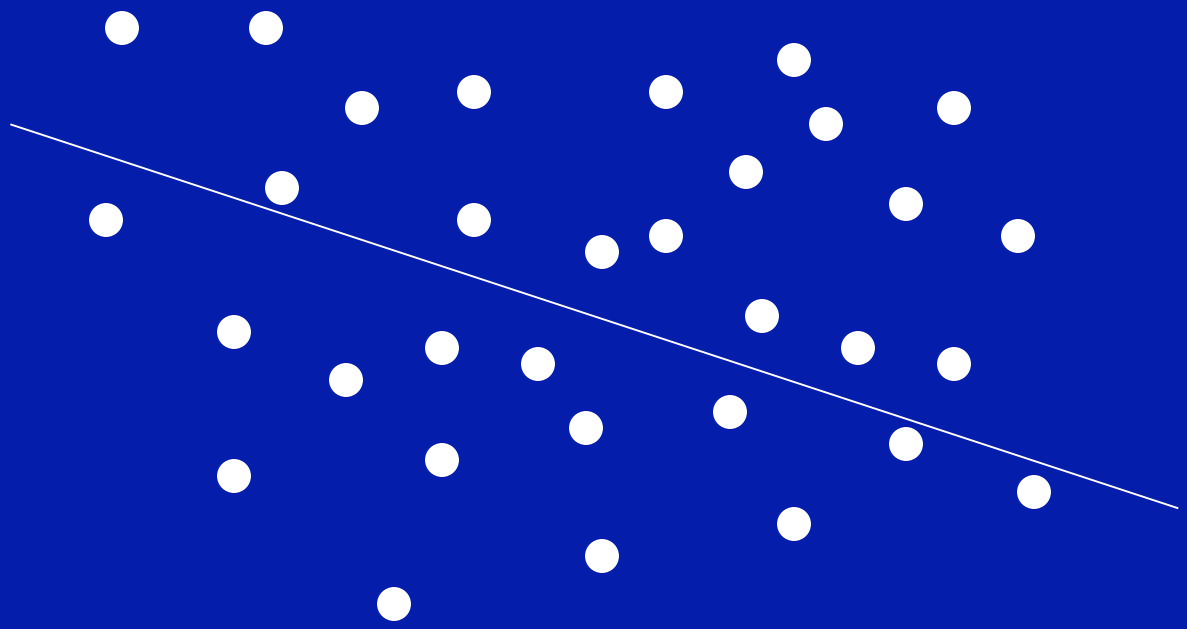
What's going on?

another set of weights



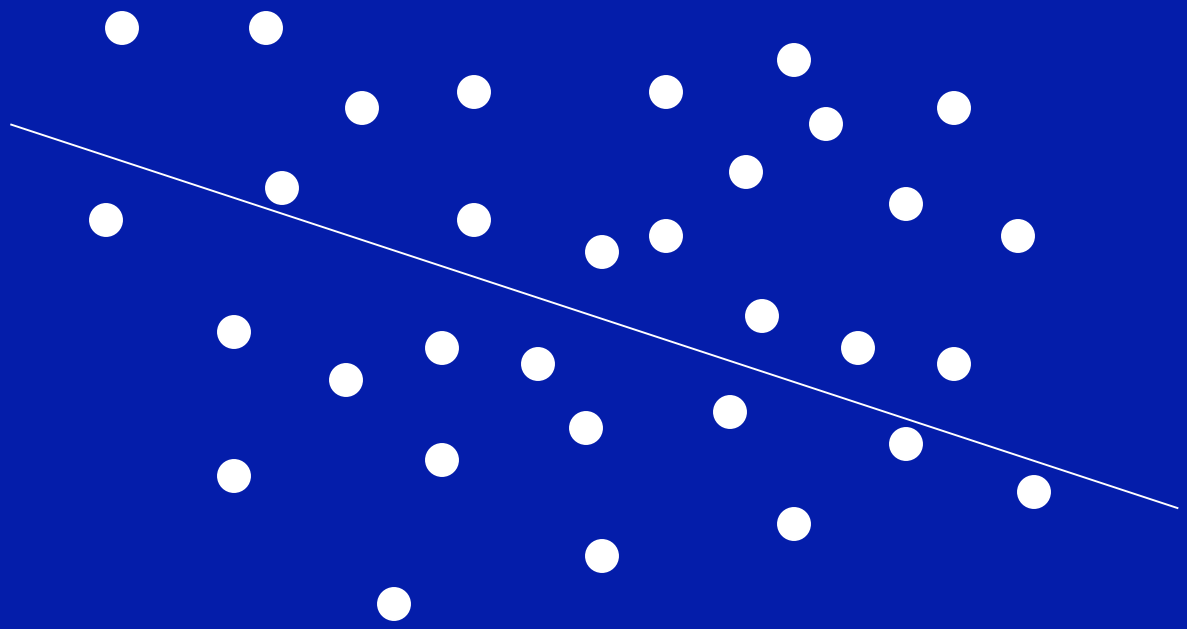
What's going on?

still another set of weights



What's going on?

still another set of weights



The perceptron looks for linear separability. That is, in the n-space defined by the inputs, it's looking for a line or plane that divides the inputs.

Observations

This perceptron can learn concepts involving “and” and “or”

This perceptron can’t learn “exclusive or” (try ex3 in the Scheme code). Not linearly separble... it won’t converge

Even a network of perceptrons arranged in a single layer can’t compute XOR (as well as other things)

Observations

The representation for the learned concept (e.g., the arch concept) is just five numbers. What does that say about the physical symbol system hypothesis?

However, if you know which questions or relations each individual weight is associated with, you still have a sense of what the numbers/weights mean.

Observations

This is another example of intelligence as search for the best representation.

The final set of weights that was the solution to the arch-learning problem is not the only solution. In fact, there are infinitely many solutions, corresponding to the infinitely many ways to draw the line or plane.

Beyond perceptrons

Perceptrons were popular in the 1960s, and some folks thought they were the key to intelligent machines.

But you needed multiple-layer perceptron networks to compute some things, and to make those work you had to build them by hand. Multiple-layer perceptron nets don't necessarily converge, and when they don't converge, they don't learn. Building big nets by hand was too time consuming, so interest in perceptrons died off in the 1970s.

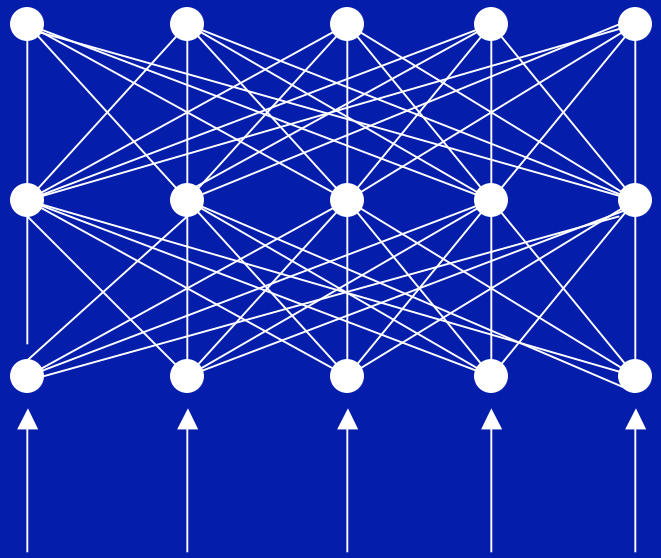
The return of the perceptron

In the 1980s, people figured out that with some changes, multiple layers of perceptron-like units could compute anything.

First, you get rid of the threshold -- replace the step function that generated output with a continuous function.

Then you put hidden layers between the inputs and the output(s).

The return of the perceptron



output layer

hidden layer

input layer

The return of the perceptron

Then for each input example you let the activation feed forward to the outputs, compare outputs to desired outputs, and then backpropagate the information about the differences to inform decisions about how to adjust the weights in the multiple layers of network. Changes to weights give rise to immediate changes in output, because there's no threshold.

The return of the perceptron

This generation of networks is called neural networks, or backpropagation nets, or connectionist networks (don't use perceptron now)

They're capable of computing anything computable
They learn well
They degrade gracefully
They handle noisy input well
They're really good at modelling low-level perceptual processing and simple learning

but...

The return of the perceptron

This generation of networks is called neural networks, or backpropagation nets, or connectionist networks (don't use perceptron now)

They can't explain their reasoning

Neither can we, easily...what the pattern of weights in the hidden layers correspond to is opaque

Not a lot of success yet in modelling higher levels of cognitive behavior (planning, story understanding, etc.)

The return of the perceptron

But this discussion of perceptrons should put you in a position to read Chapter 11.2 and see how those backprop networks came about.

Which leads us to the end of this term...

Things...

Good bye!

Thanks for being nice to the new guy.

We'll see you at noon on Friday, December 10, in MCML 166 for the final exam

Have a great holiday break, and come visit next term.