CS4710: Artificial Intelligence
Intro to Machine Learning

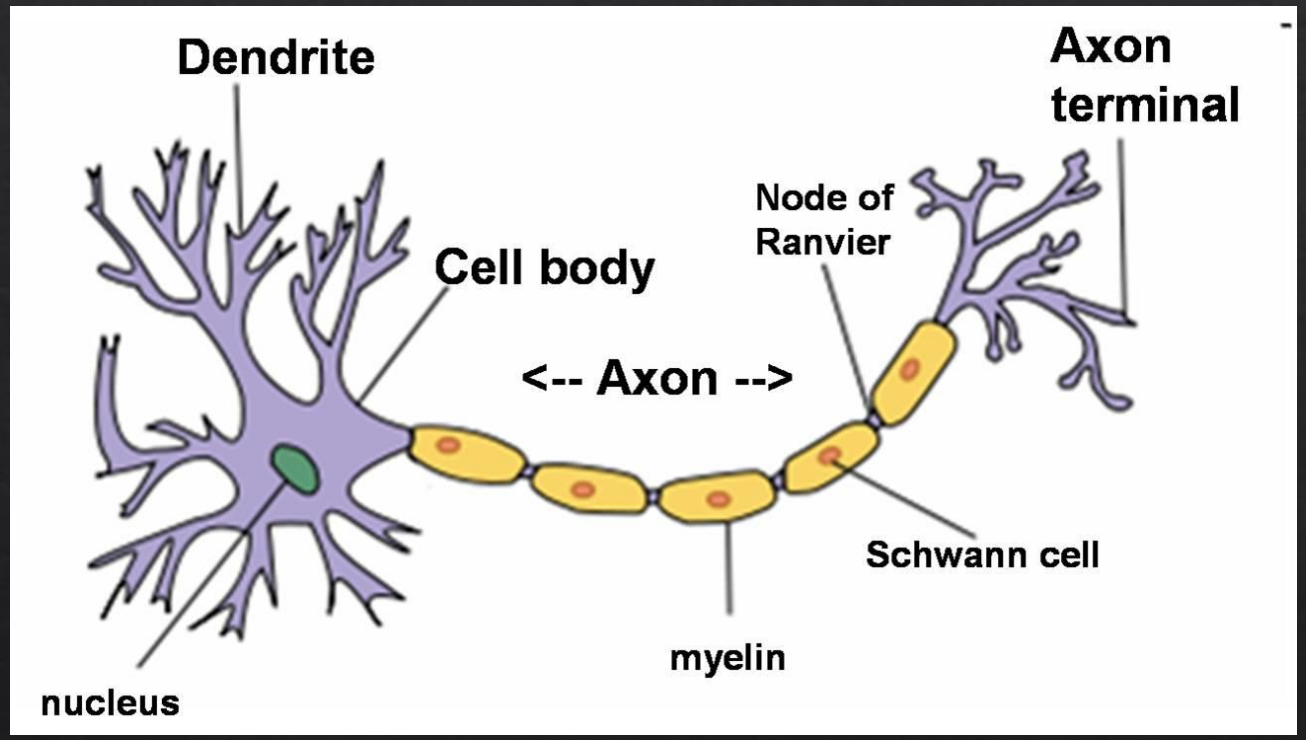
Let's look at a couple more machine learning algorithms!

Artificial Neural Networks

Intro to Neural Networks

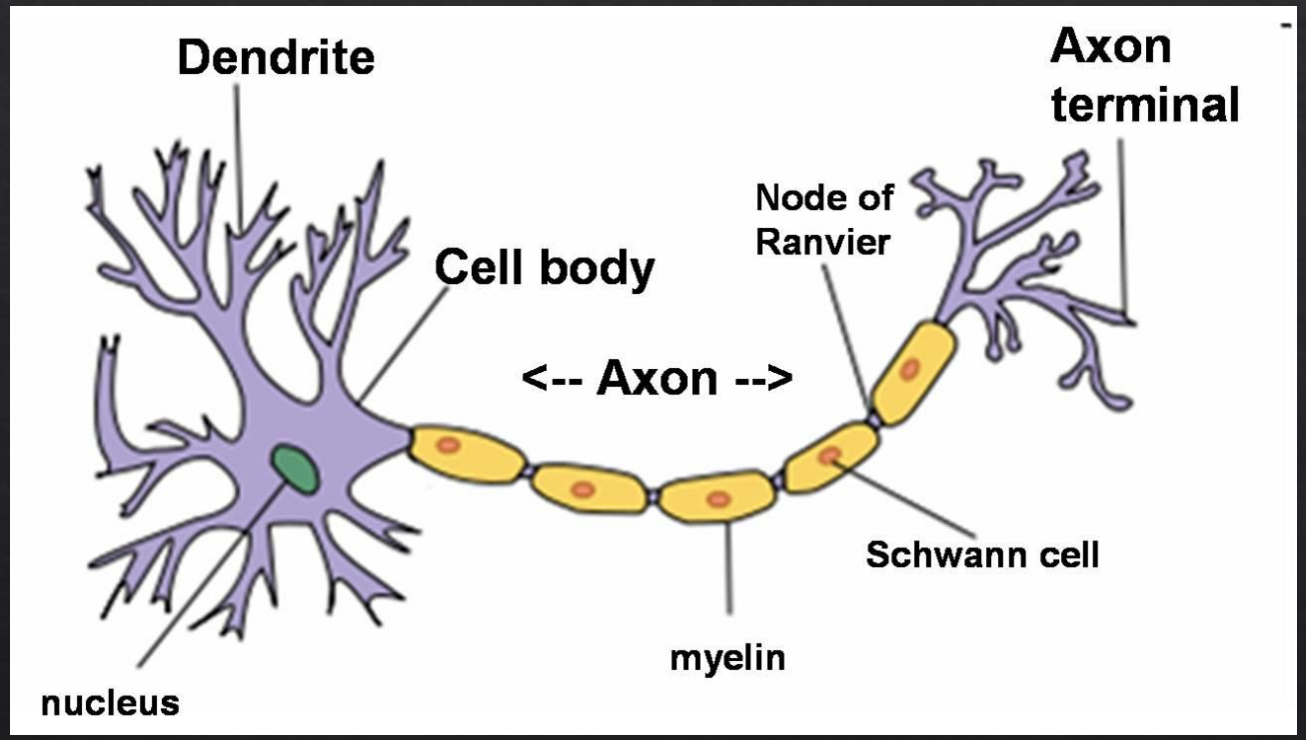
- Used for many pattern recognition tasks
- Biologically inspired
- Must first understand the very basics of how neurons work

Neurons



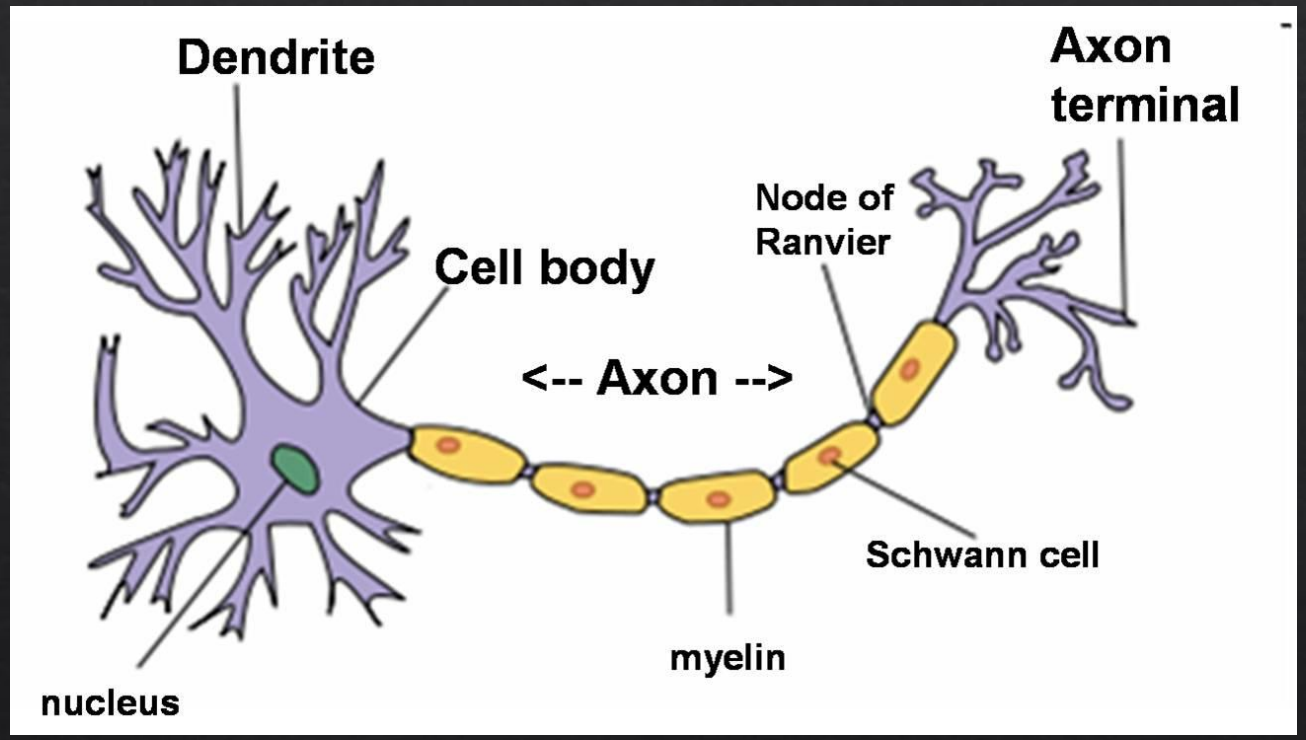
- Dendrite:
  - Extend from the cell body
  - Receive messages from other neurons
- Axon:
  - Where signal is outputted from the neuron
  - Electrical signal moves across axon
- Synapse (Axon terminal in image here):
  - Connects this neuron to Dendrite of another
  - Allows only a certain amount of the “signal” through (via chemical processes)

Neurons



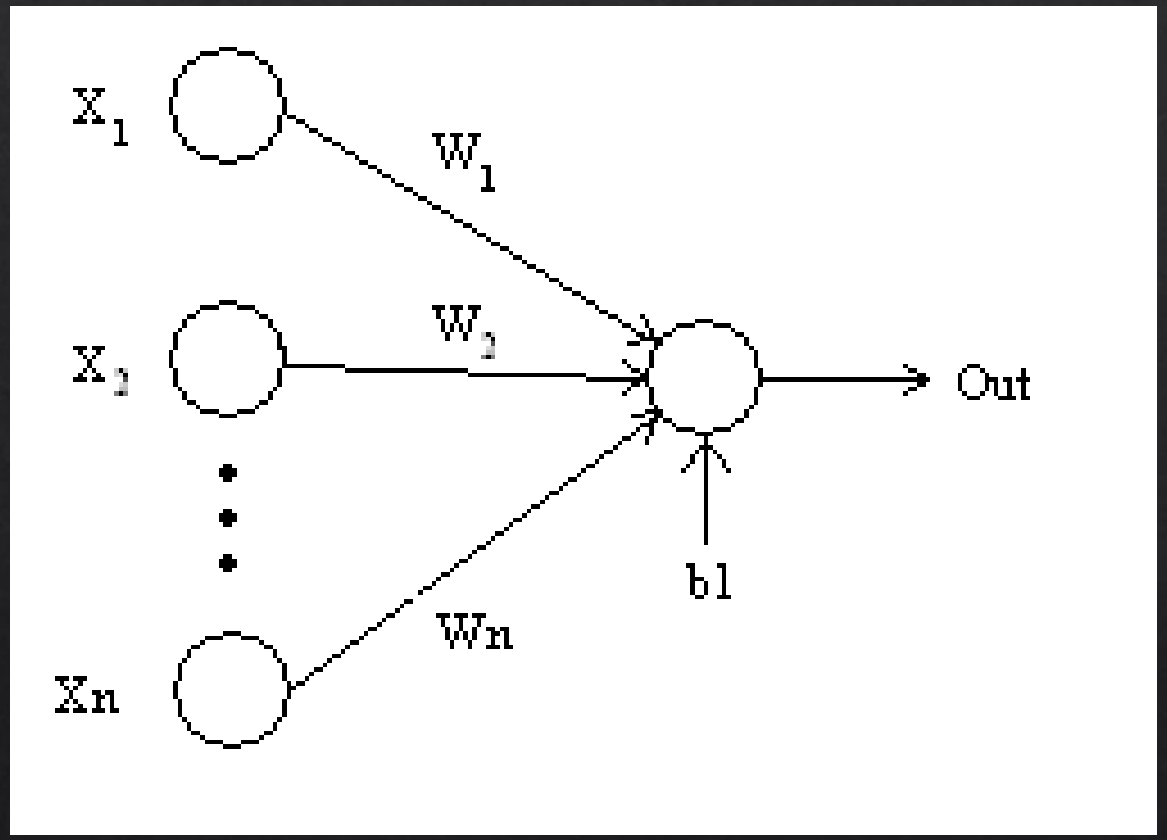
- Basic functionality of neuron:
  - Electrical signal arrives at ***Dendrites*** (from other neurons)
  - ***Axon*** transmits the signal along to the ***synapses***
  - At the synapse, a molecule called a ***neurotransmitter*** is released, this chemical stimulates the next neuron, creating an electrical signal at the next Dendrite
- So the amount of neurotransmitters that are released controls the amount of signal that "gets through"

Neurons



- Basic Idea (as a CS person):
  - A neuron receives a bunch of electrical signals from other neurons.
  - There is some function that determines how much of that signal gets through to the connecting neurons

Perceptron

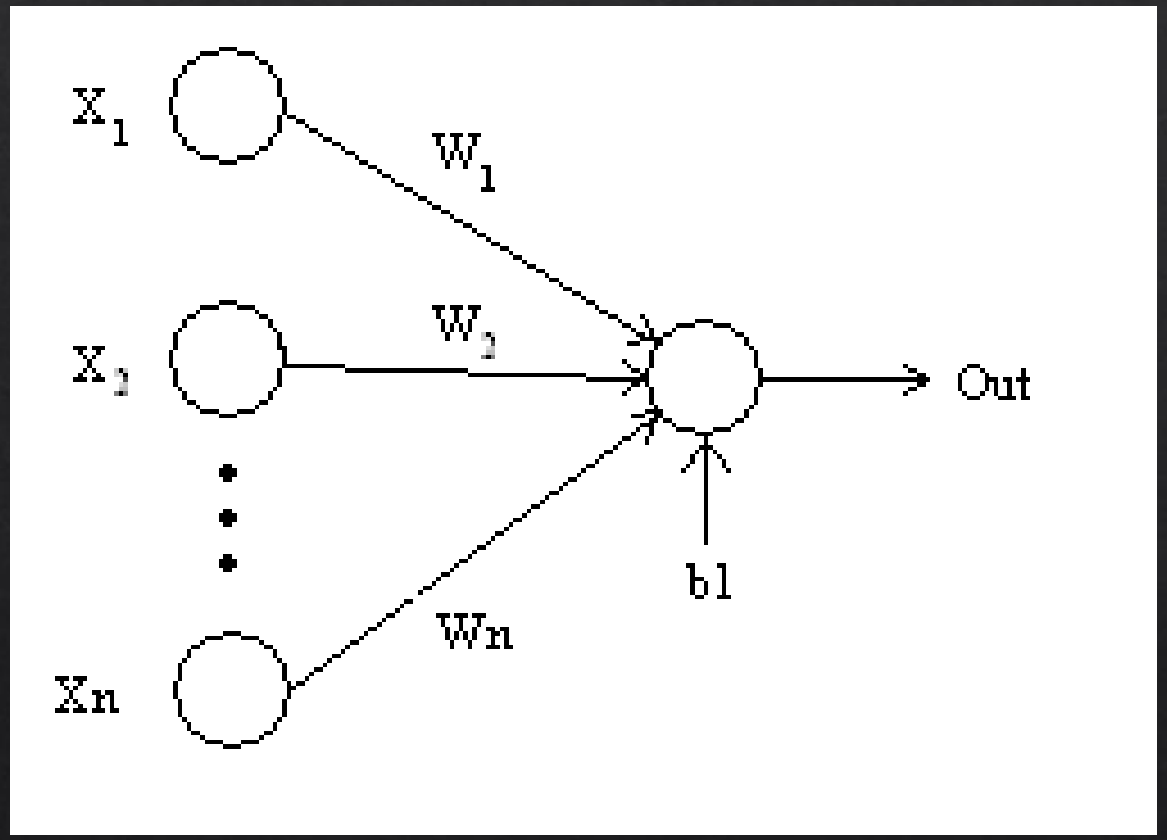


A computational model that simulates a neuron

Captures the basic ideas of a neuron and its behavior

We will start by looking at networks with only one perceptron or only one layer of “neurons”

Perceptron



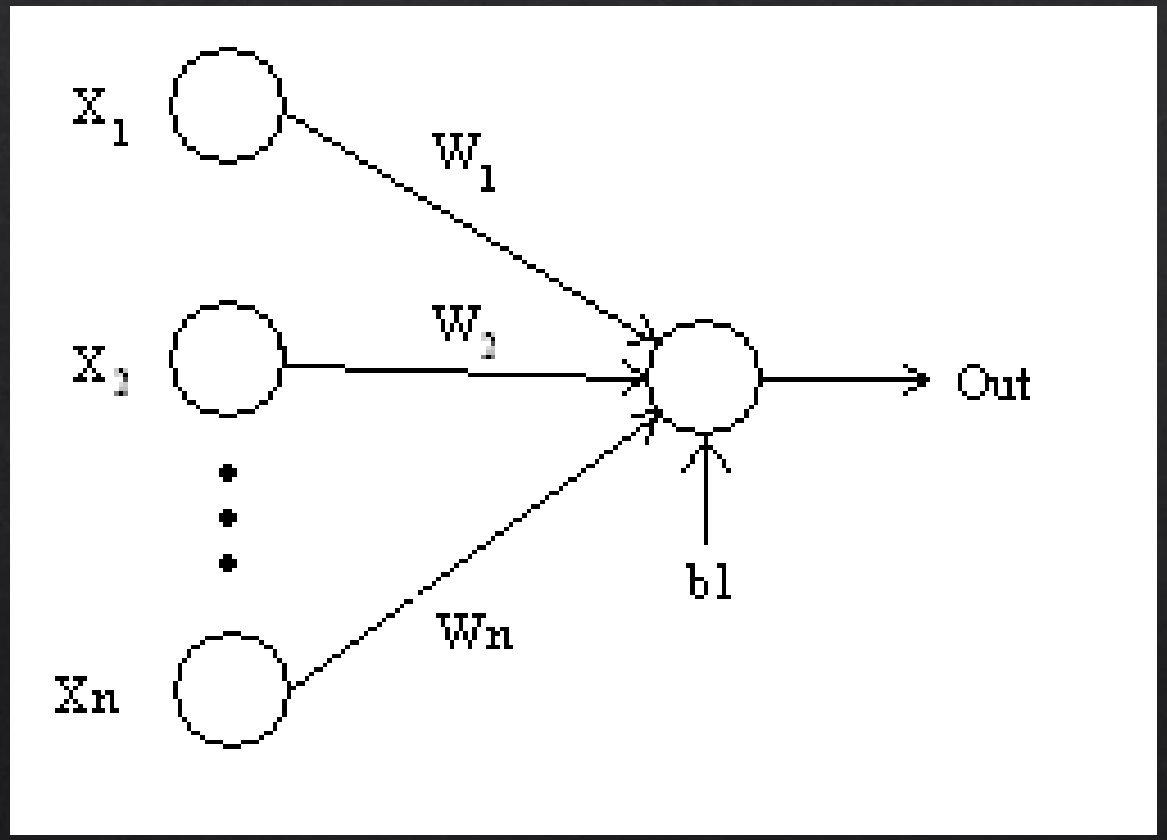
X1…Xn : 0 or 1, represents inputs to neuron

W1…Wn : 0-1, represents weight for that input

B1 : Threshold

Out : 0 if neuron not firing, otherwise 1

Perceptron: Basic Functionality



If X1*W1 + X2*W2 … > b1 (Threshold){

    Output = 1

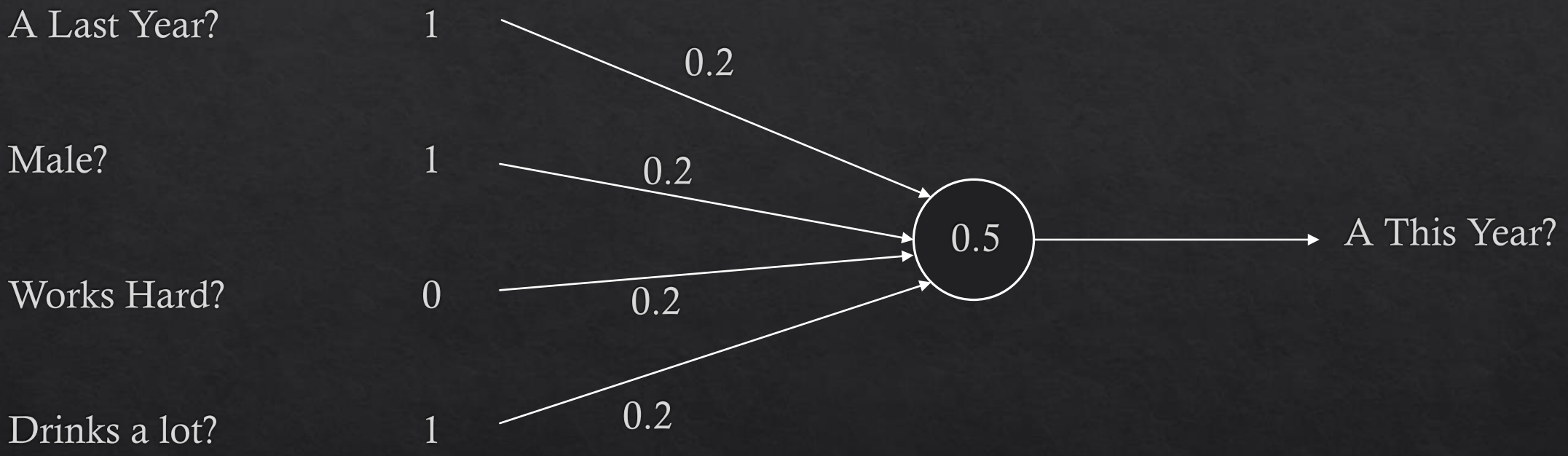
}

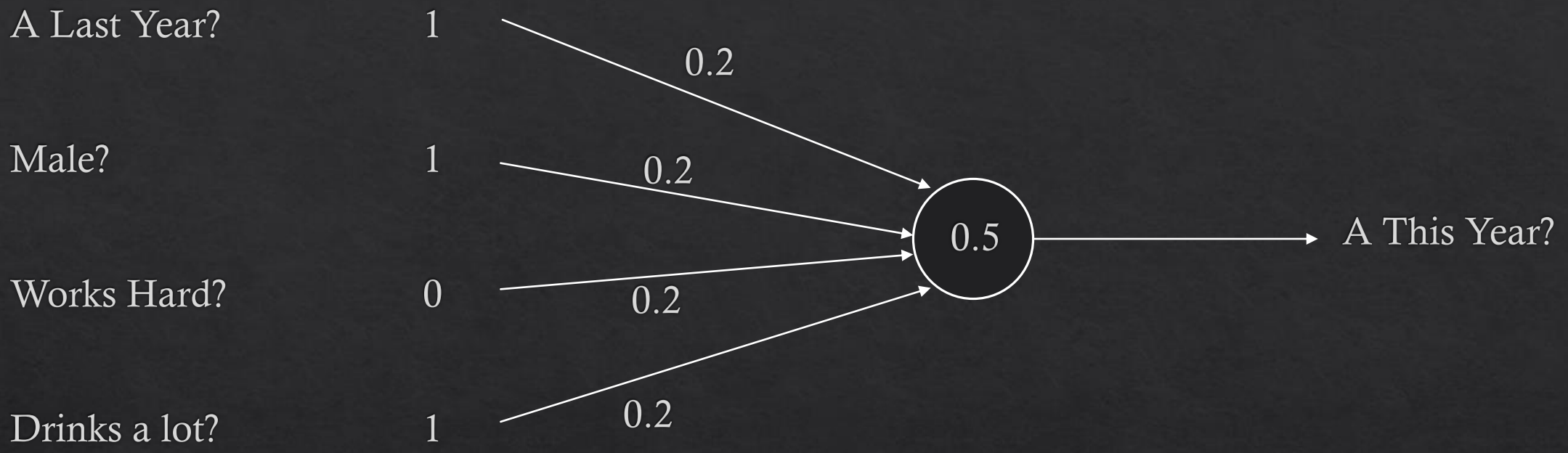
Else{

    Output = 0

}

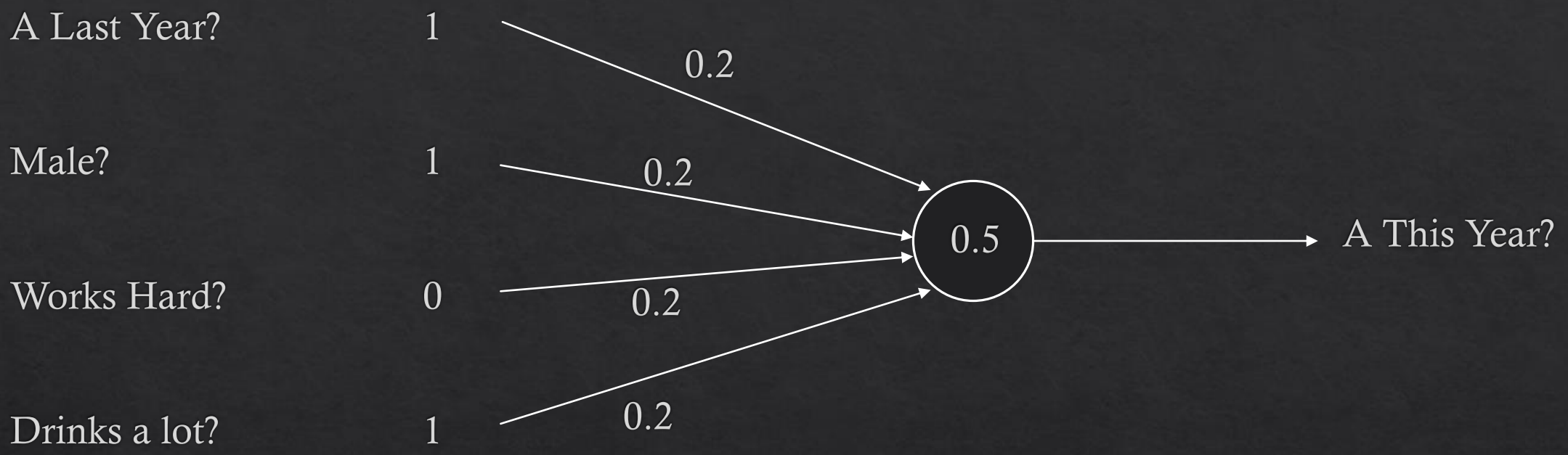
*This is called a ***threshold activation function***.
We will see different versions of this later


A Last Year?
1
0.2
Male?
1
0.2
0.5
A This Year?
Works Hard?
0
0.2
Drinks a lot?
1
0.2



Ok! But how is this learning?

Well, we want to 1) test the network on every example we have available to us and 2) adjust the weights on the edges until the network correctly classifies every single example.



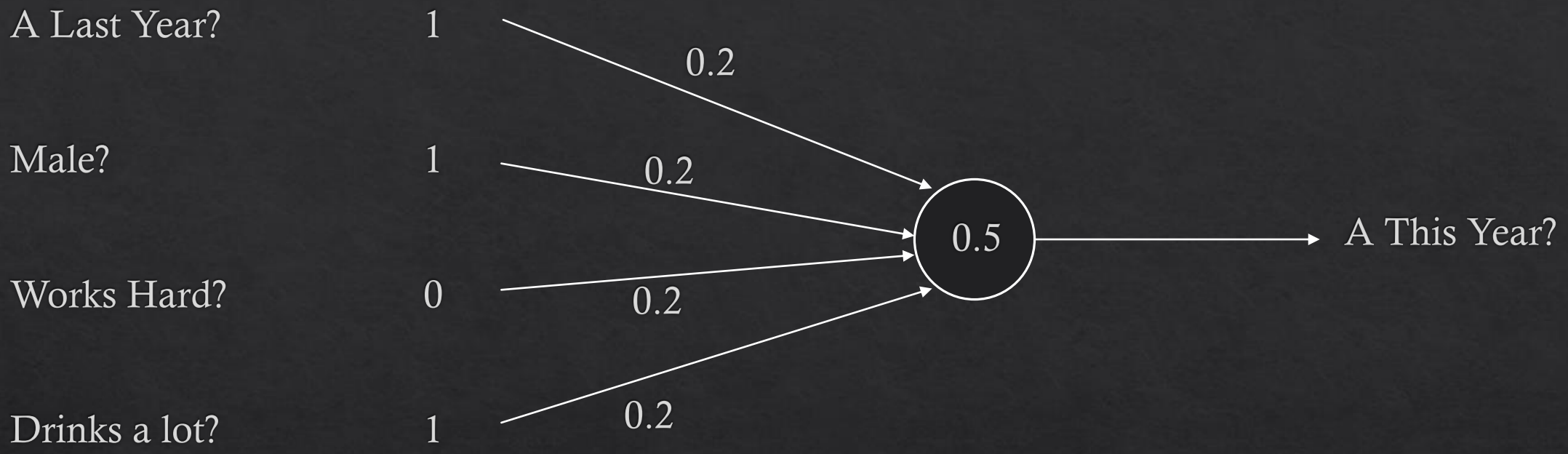
Let d = learning rate (float)

Until every example produces the correct output

For each example in training set:

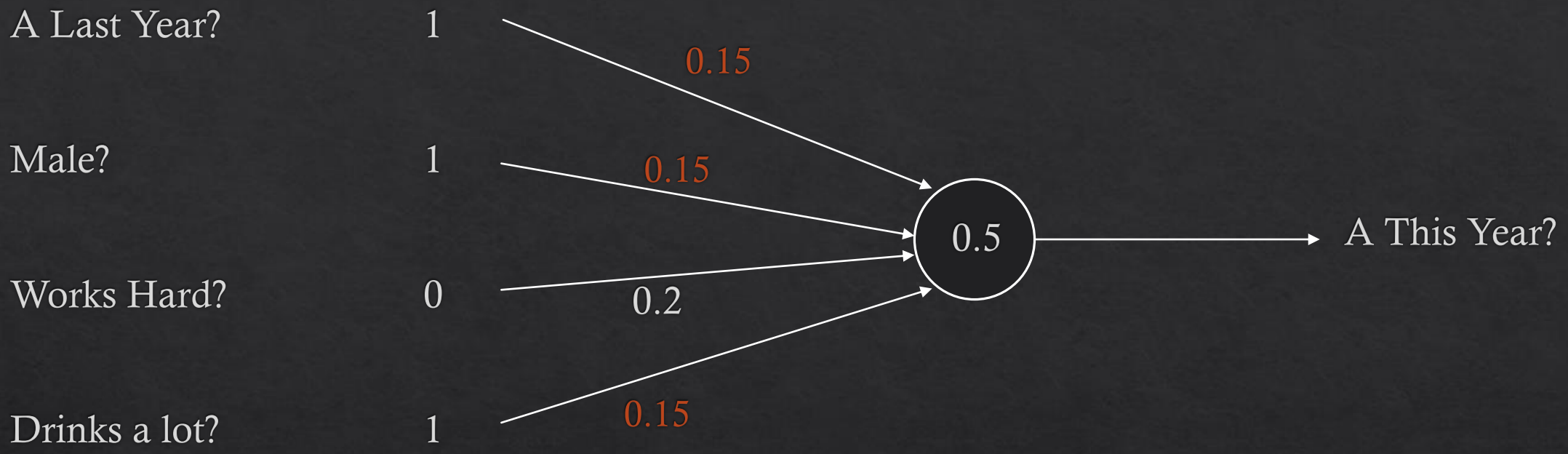
If output is 0 but should be 1: Raise weights on active connections by d

If output is 1 but should be 0: Lower weights on active connections by d



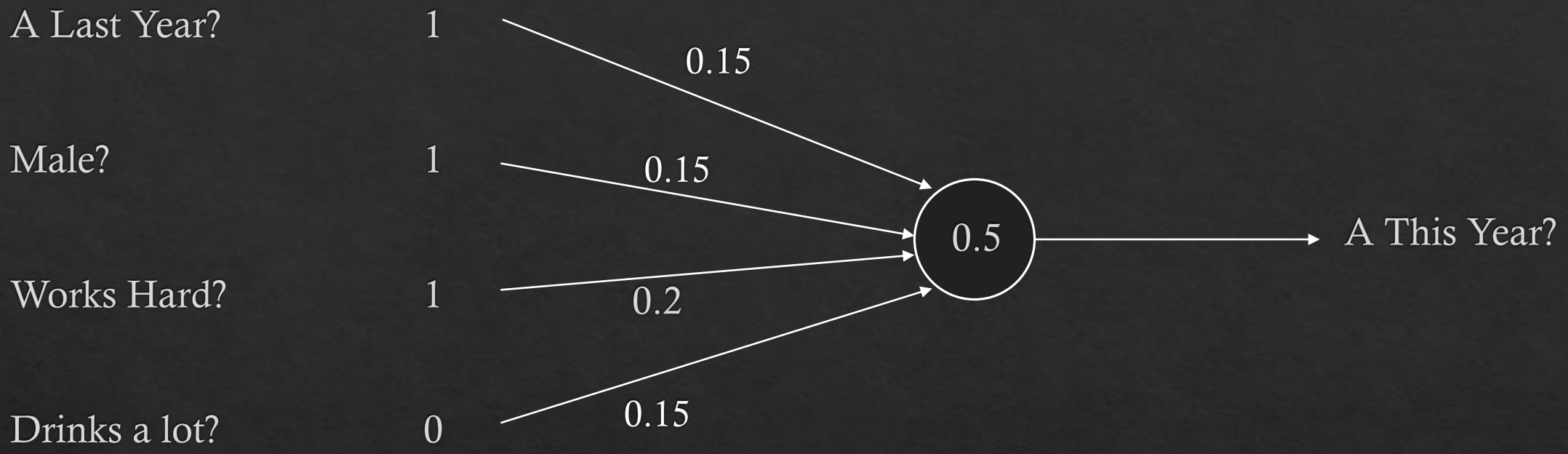
- Let d = learning rate (float)
- Until every example produces the correct output
- For each example in training set:
  - If output is 0 but should be 1: Raise weights on active connections by d
  - **If output is 1 but should be 0: Lower weights on active connections by d**

Student	A last year?	Male?	Works Hard?	Drinks?	A this year?
Richard	**Yes**	**Yes**	**No**	**Yes**	**No**
Allen	Yes	Yes	Yes	No	Yes
Alison	No	No	Yes	No	No
Jeff	No	Yes	No	Yes	No
Gail	Yes	No	Yes	Yes	Yes
Simon	No	Yes	Yes	Yes	No



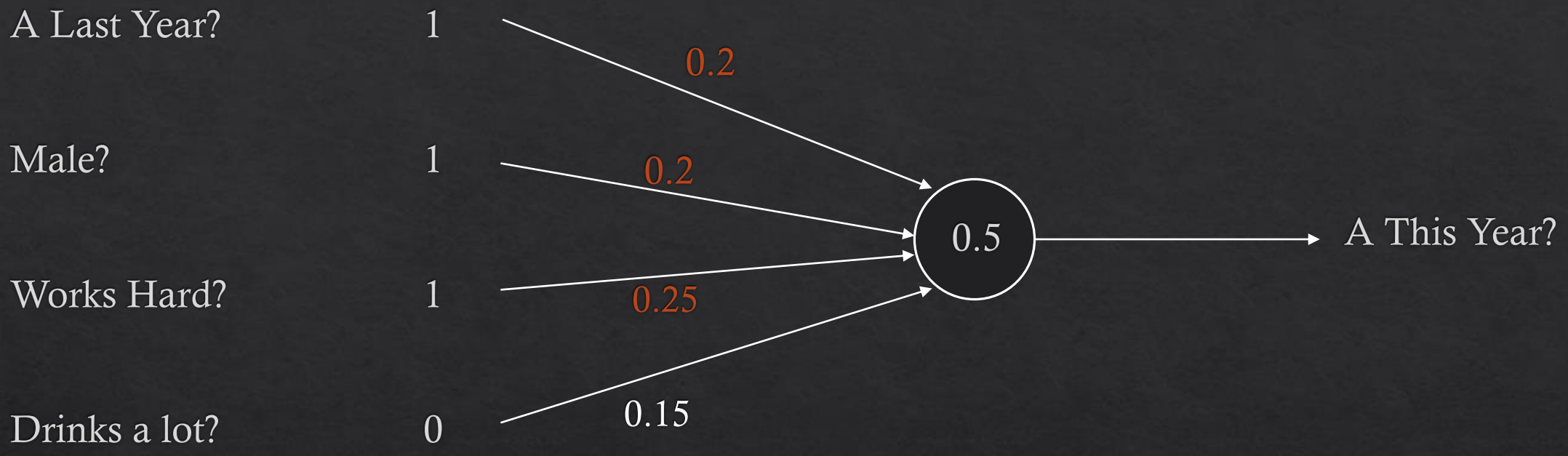
- Let d = learning rate (float)
- Until every example produces the correct output
- For each example in training set:
  - If output is 0 but should be 1: Raise weights on active connections by d
  - **If output is 1 but should be 0: Lower weights on active connections by d**

Student	A last year?	Male?	Works Hard?	Drinks?	A this year?
Richard	**Yes**	**Yes**	**No**	**Yes**	**No**
Allen	Yes	Yes	Yes	No	Yes
Alison	No	No	Yes	No	No
Jeff	No	Yes	No	Yes	No
Gail	Yes	No	Yes	Yes	Yes
Simon	No	Yes	Yes	Yes	No



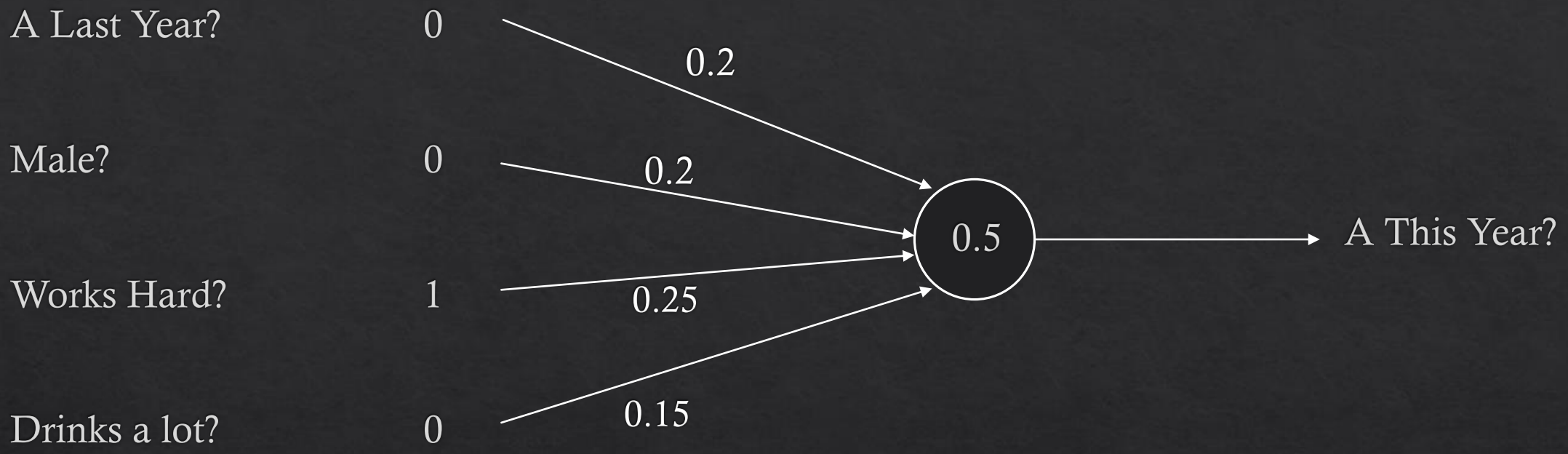
- Let d = learning rate (float)
- Until every example produces the correct output
- For each example in training set:
  - **If output is 0 but should be 1: Raise weights on active connections by d**
  - If output is 1 but should be 0: Lower weights on active connections by d

Student	A last year?	Male?	Works Hard?	Drinks?	A this year?
Richard	Yes	Yes	No	Yes	No
Allen	**Yes**	**Yes**	**Yes**	**No**	**Yes**
Alison	No	No	Yes	No	No
Jeff	No	Yes	No	Yes	No
Gail	Yes	No	Yes	Yes	Yes
Simon	No	Yes	Yes	Yes	No



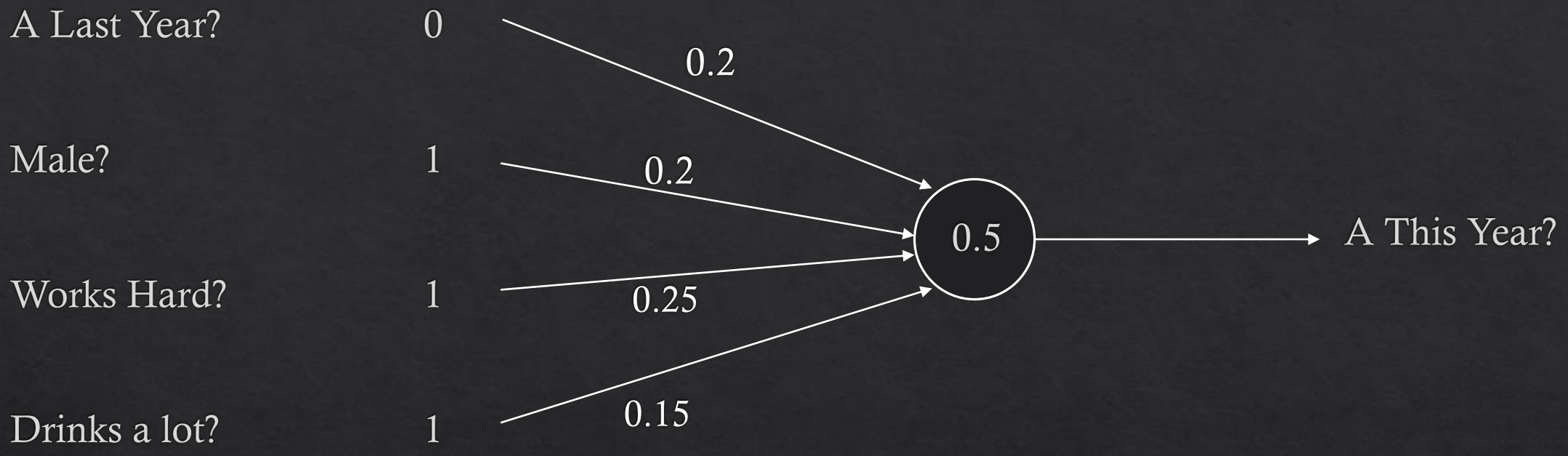
- Let d = learning rate (float)
- Until every example produces the correct output
- For each example in training set:
  - **If output is 0 but should be 1: Raise weights on active connections by d**
  - If output is 1 but should be 0: Lower weights on active connections by d

Student	A last year?	Male?	Works Hard?	Drinks?	A this year?
Richard	Yes	Yes	No	Yes	No
Allen	**Yes**	**Yes**	**Yes**	**No**	**Yes**
Alison	No	No	Yes	No	No
Jeff	No	Yes	No	Yes	No
Gail	Yes	No	Yes	Yes	Yes
Simon	No	Yes	Yes	Yes	No



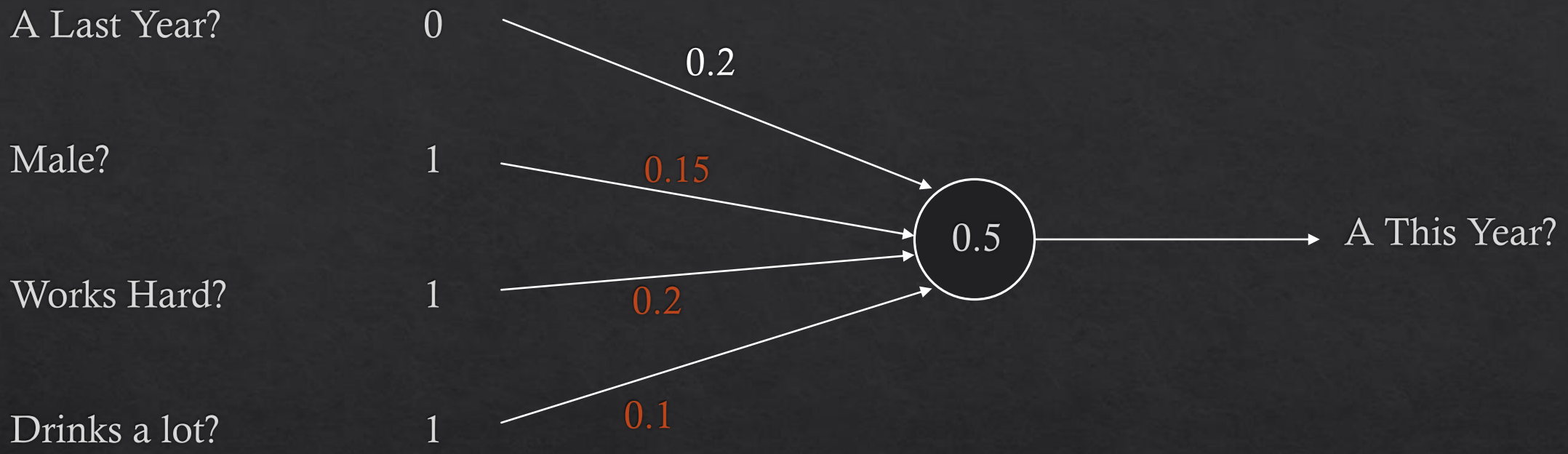
- Let d = learning rate (float)
- Until every example produces the correct output
- For each example in training set:
  - If output is 0 but should be 1: Raise weights on active connections by d
  - If output is 1 but should be 0: Lower weights on active connections by d

Student	A last year?	Male?	Works Hard?	Drinks?	A this year?
Richard	Yes	Yes	No	Yes	No
Allen	Yes	Yes	Yes	No	Yes
Alison	**No**	**No**	**Yes**	**No**	**No**
Jeff	No	Yes	No	Yes	No
Gail	Yes	No	Yes	Yes	Yes
Simon	No	Yes	Yes	Yes	No



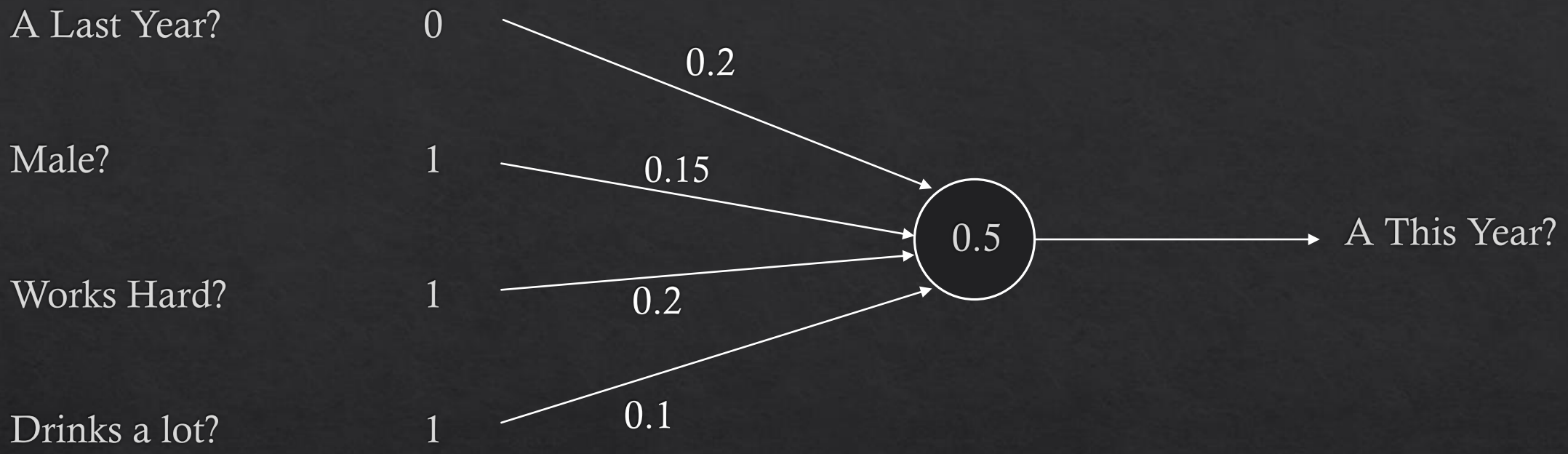
- Let d = learning rate (float)
- Until every example produces the correct output
- For each example in training set:
  - If output is 0 but should be 1: Raise weights on active connections by d
  - **If output is 1 but should be 0: Lower weights on active connections by d**

Student	A last year?	Male?	Works Hard?	Drinks?	A this year?
Richard	Yes	Yes	No	Yes	No
Allen	Yes	Yes	Yes	No	Yes
Alison	No	No	Yes	No	No
Jeff	No	Yes	No	Yes	No
Gail	Yes	No	Yes	Yes	Yes
Simon	**No**	**Yes**	**Yes**	**Yes**	**No**



- Let d = learning rate (float)
- Until every example produces the correct output
- For each example in training set:
  - If output is 0 but should be 1: Raise weights on active connections by d
  - **If output is 1 but should be 0: Lower weights on active connections by d**

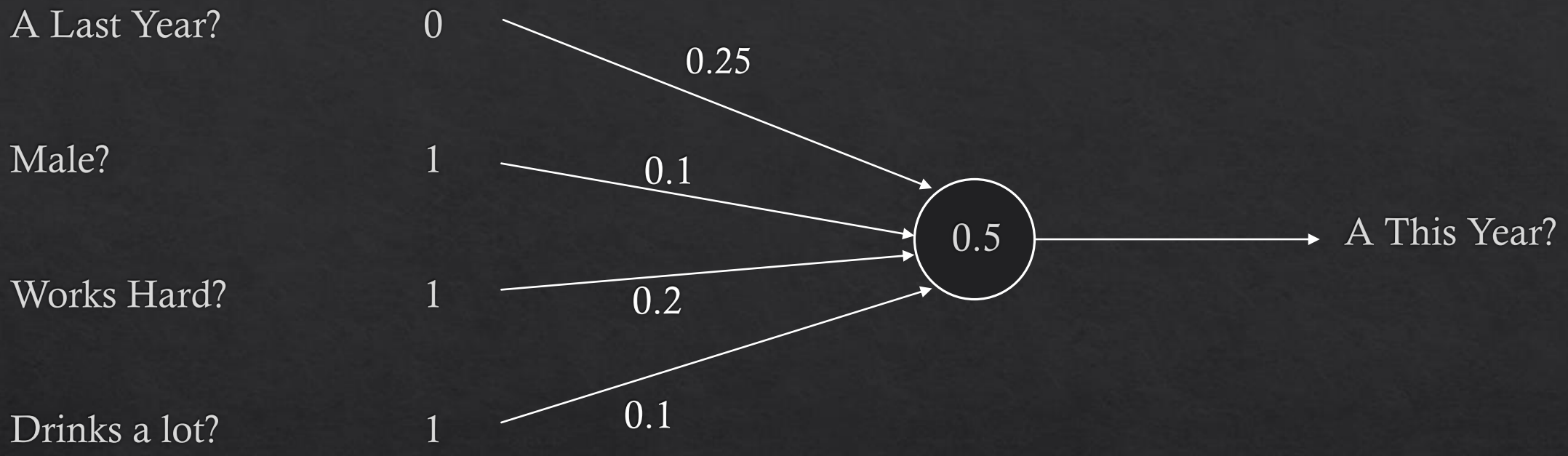
Student	A last year?	Male?	Works Hard?	Drinks?	A this year?
Richard	Yes	Yes	No	Yes	No
Allen	Yes	Yes	Yes	No	Yes
Alison	No	No	Yes	No	No
Jeff	No	Yes	No	Yes	No
Gail	Yes	No	Yes	Yes	Yes
Simon	**No**	**Yes**	**Yes**	**Yes**	**No**



- Not done yet
- Need to go through training examples again until the network converges and produces good output for all examples

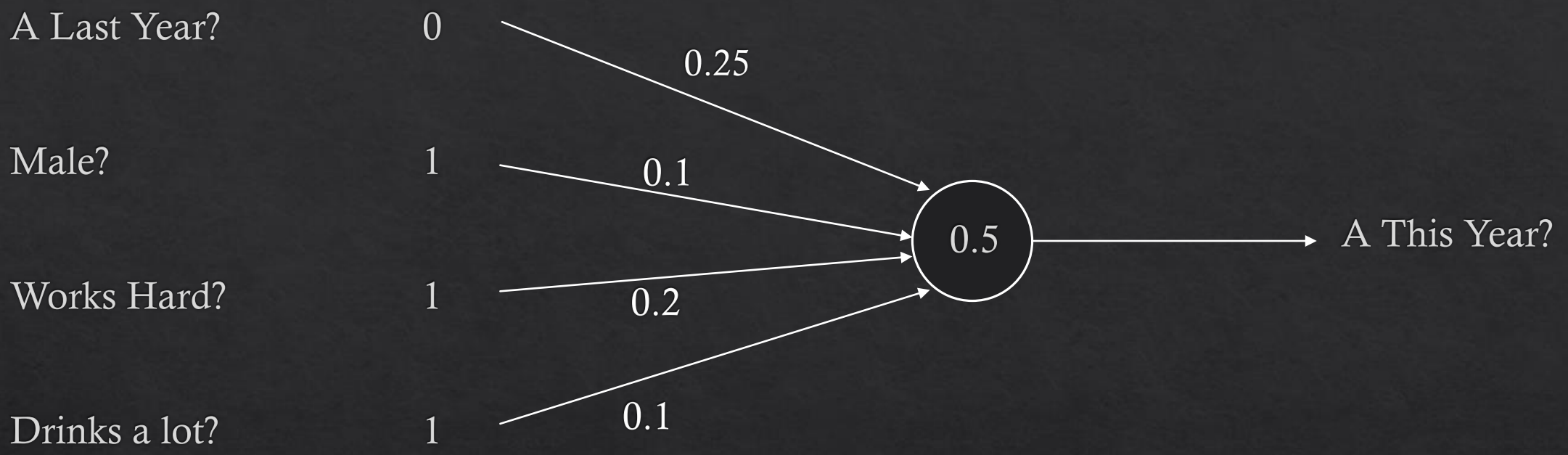
- What if this is not possible?

Student	A last year?	Male?	Works Hard?	Drinks?	A this year?
Richard	Yes	Yes	No	Yes	No
Allen	Yes	Yes	Yes	No	Yes
Alison	No	No	Yes	No	No
Jeff	No	Yes	No	Yes	No
Gail	Yes	No	Yes	Yes	Yes
Simon	No	Yes	Yes	Yes	No



Student	A last year?	Male?	Works Hard?	Drinks?	A this year?
Richard	Yes	Yes	No	Yes	No
Allen	Yes	Yes	Yes	No	Yes
Alison	No	No	Yes	No	No
Jeff	No	Yes	No	Yes	No
Gail	Yes	No	Yes	Yes	Yes
Simon	No	Yes	Yes	Yes	No

- After another pass, the weights are as above
- These work for all training examples!



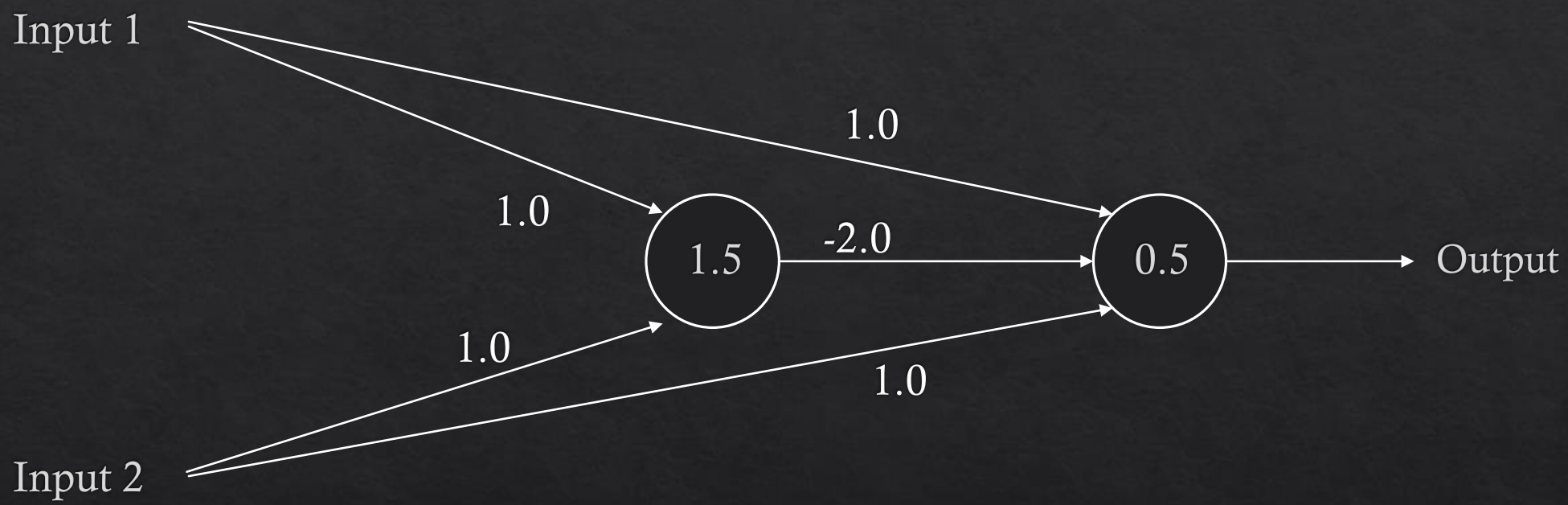
- This single-layer network works for this example…however simple networks like this cannot solve any problem.

- Famously: It is not possible to use one neuron to recognize the XOR function
  - Go on…try to
  - Shown by Minsky and Papert in 1969 (Book is called *Perceptrons*)

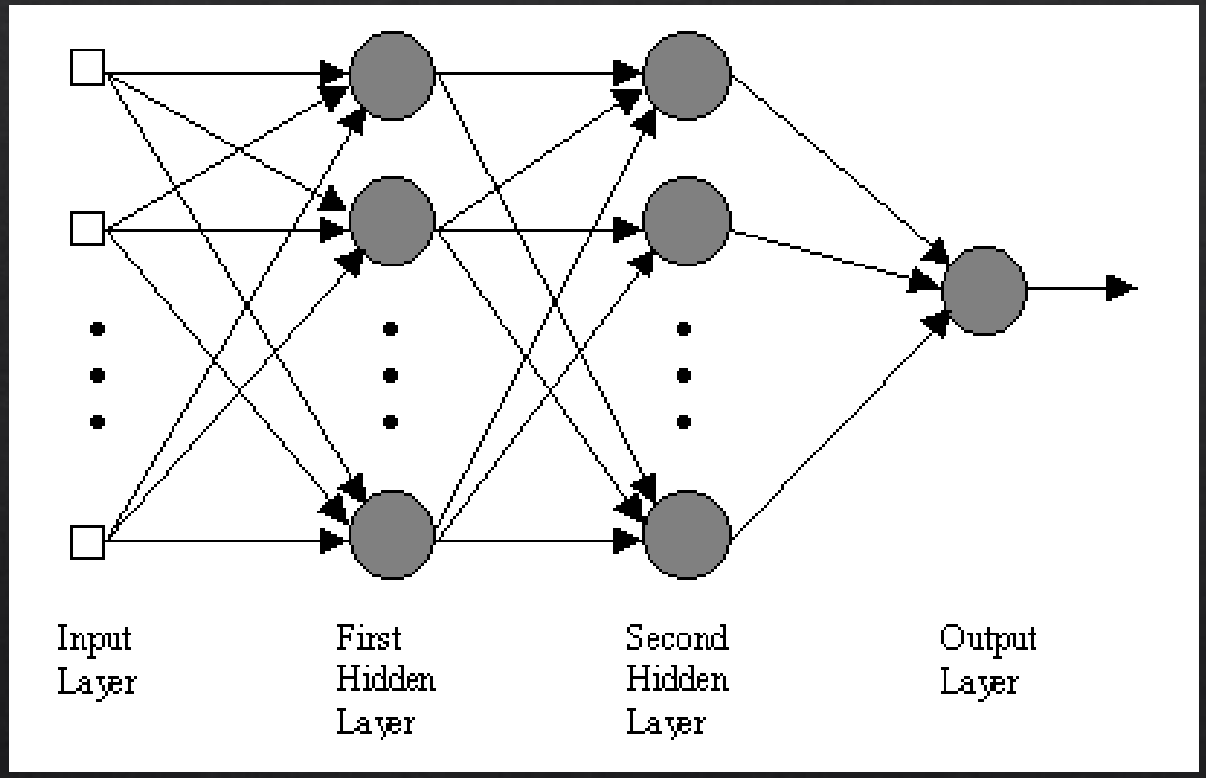
Artificial Neural Networks

More Advanced Networks

XOR Network: Does This Work?

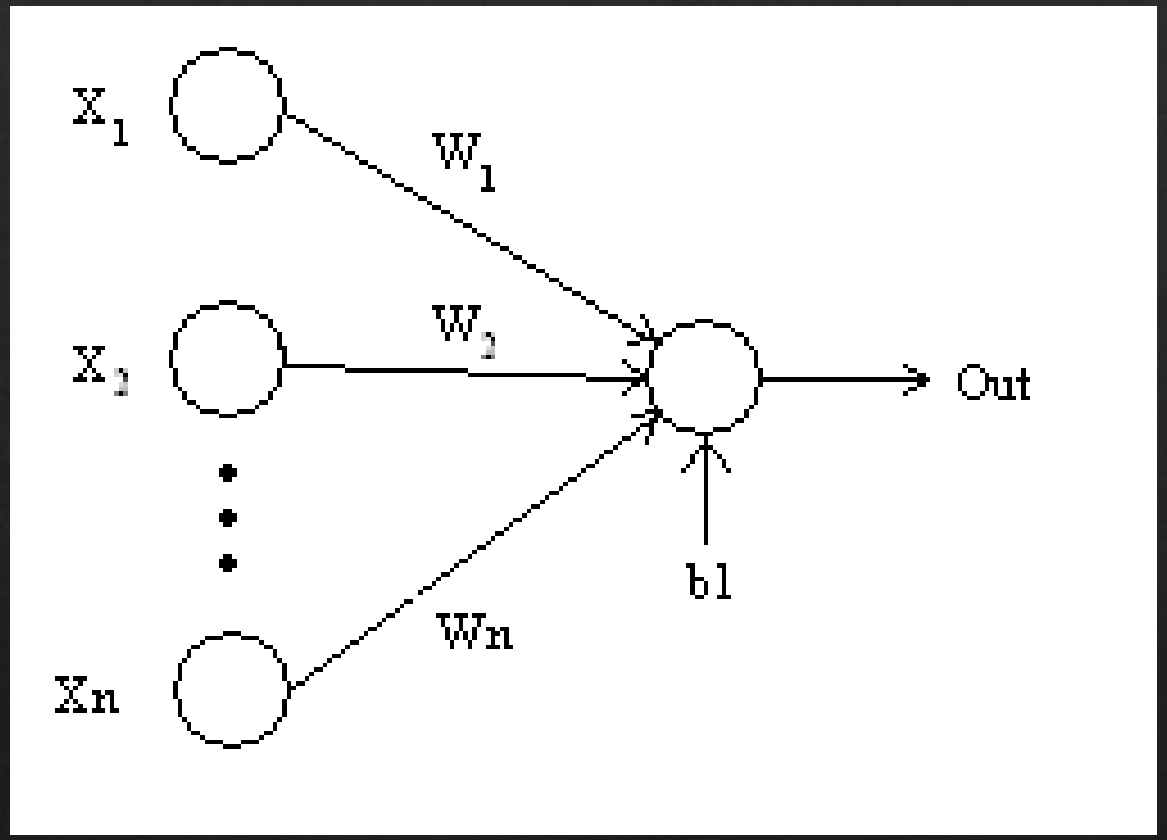


Multi-Layer Networks



- Input Layer:
  - Left-most layer that accepts inputs directly
- Hidden Layer(s):
  - Layers that receive input from and output to other neurons
- Output Layer:
  - Layer of neurons whose output is the output of the system

Perceptron: Basic Functionality



We have seen a basic perceptron like this:

If X1*W1 + X2*W2 … > b1 (Threshold){

    Output = 1

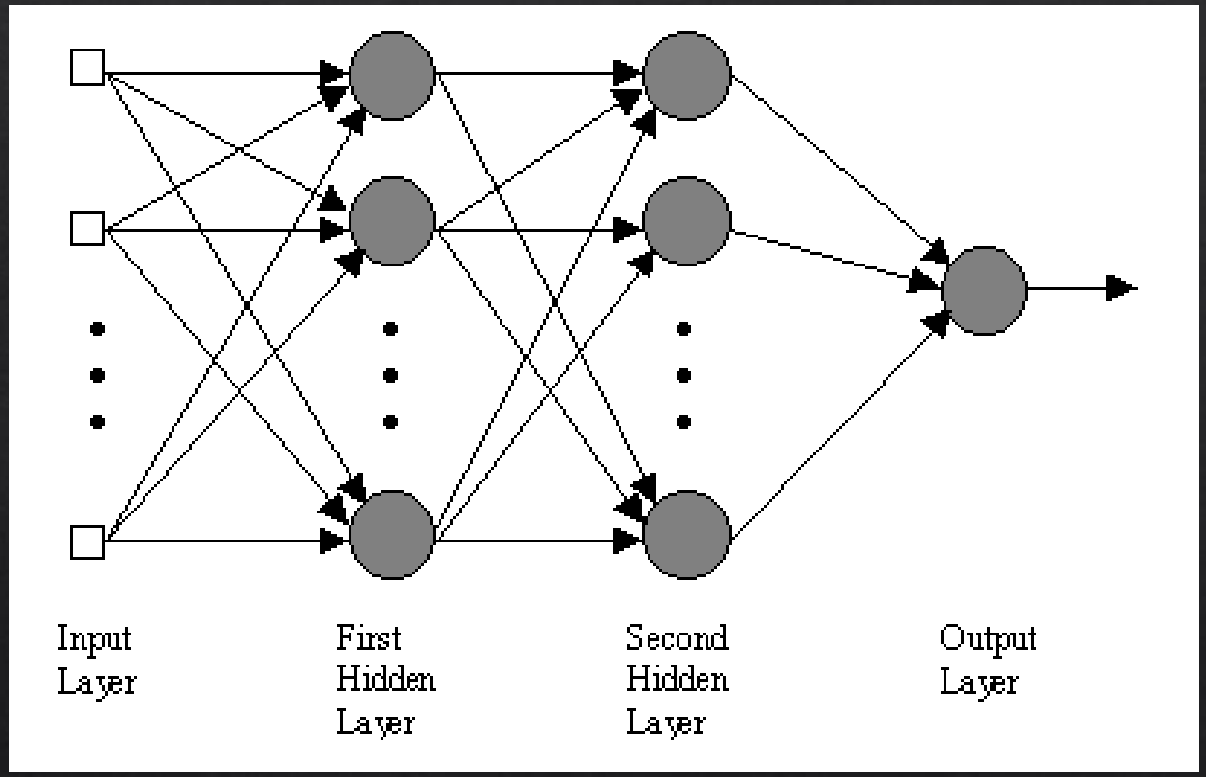
}

Else{

    Output = 0

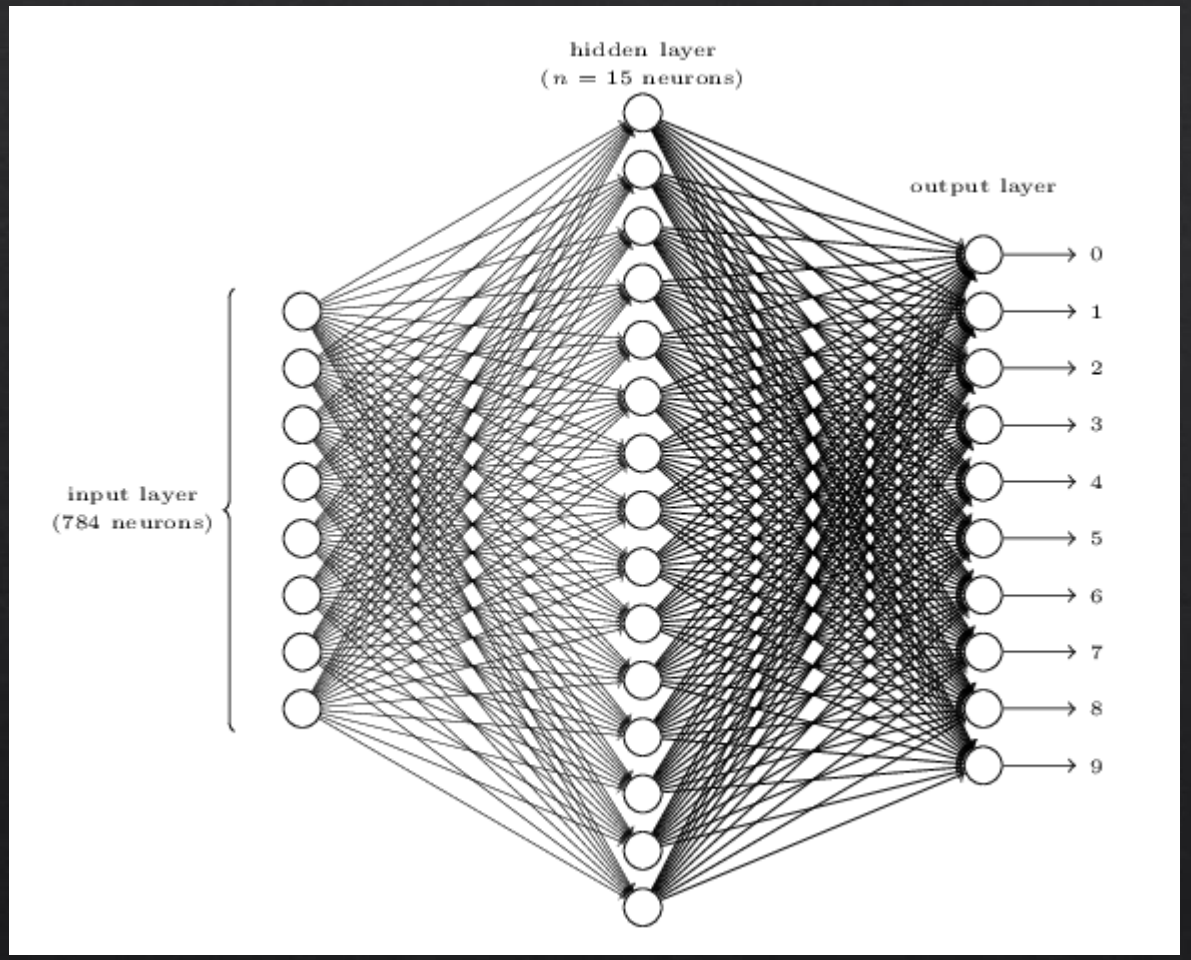
}

Multi-Layer Networks



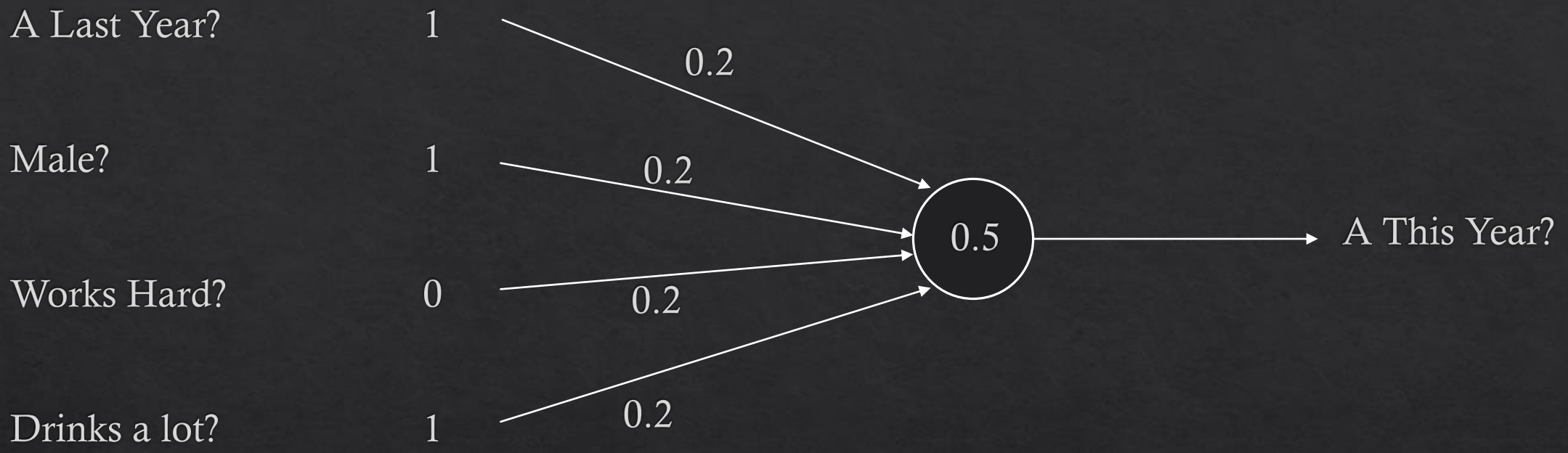
- Works the same way…however:
- How do we update the weights when training the network? It's not clear how this should work.
- Solution: Backpropagation!
  - We'll get there in a sec. First, some more basic stuff.

Multi-Layer Networks



- This network will recognize the single hand-written digits 0-9
- Some of the network is not drawn for simplicity
- You would have to train the network on many examples of hand-written images of these ten digits
- Notice that there are multiple outputs. How would we interpret these?

Artificial Neural Networks
Activation Functions

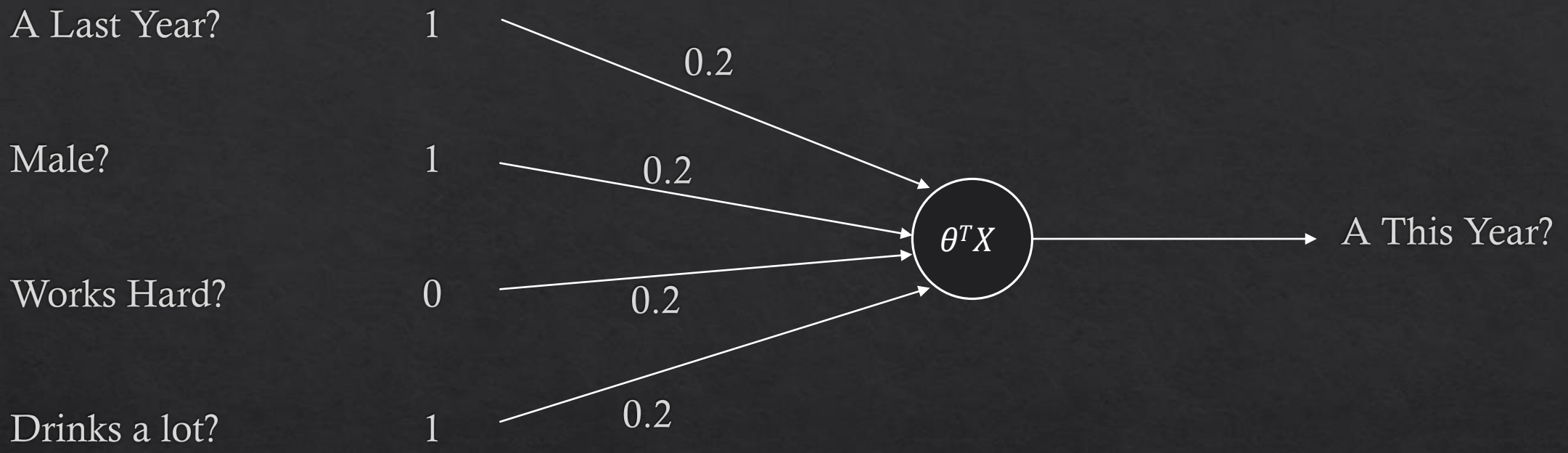


The *activation function* of a neuron is the function it uses to determine whether to fire or not

These *perceptrons* we have seen so far have a simple activation function:

Threshold Activation Function: Fire a 1 on output if total input (plus some bias) exceeds zero

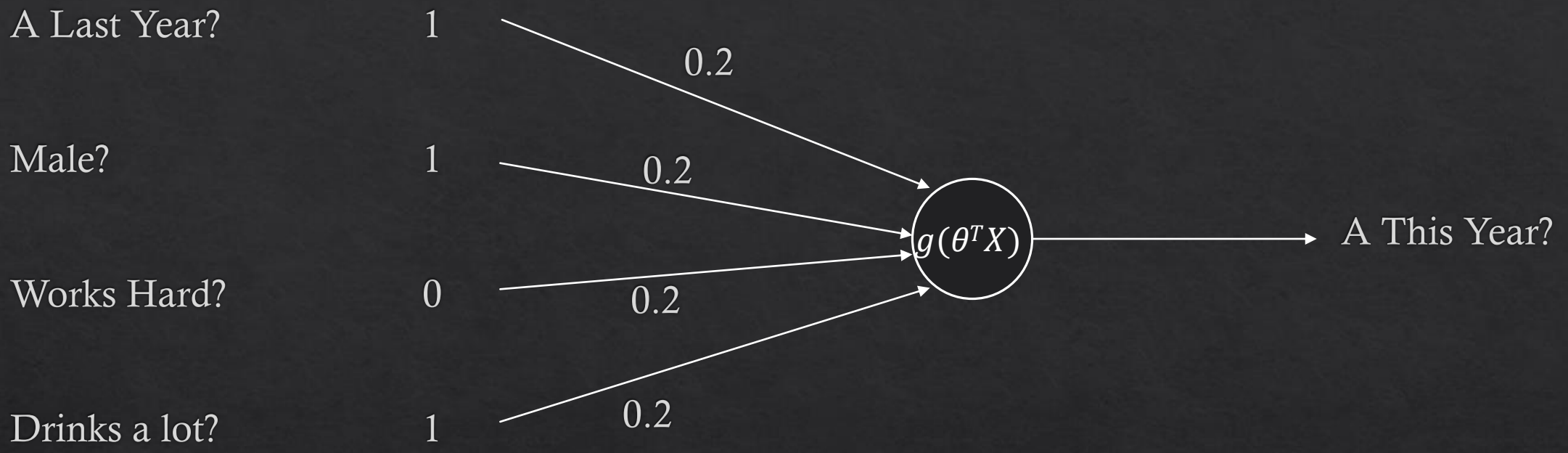
…but we can certainly use others!



Other activation functions:

Linear Activation Function: Output $\theta^T X$ on output. No threshold, this is a continuous function. Notice that final output will be continuous as well.

Could develop a cost function on the outputs of this network and run gradient descent. What do you notice about this?



Other activation functions:

Sigmoid Activation Function: Output g($\theta^T X$) on output. No threshold, this is a continuous function. Notice that final output will be continuous as well. g() is our beloved sigmoid function.

Could develop a cost function on the outputs of this network and run gradient descent. What do you notice about this? Again…it is equivalent to logistic regression.

Artificial Neural Networks
Backpropagation

Cost Function

For the purposes of simplifying our discussion of this algorithm, we are going to assume:

- That our network has *hidden layers*
- That activation function is a *sigmoid*
- The cost function is same as *logistic reg.*

Problem:

- We know how to update weights on a single perceptron network (same as logistic regression)
- Need a way to update weights on edges in middle of the network.

Backpropagation: Overview

An algorithm that updates all theta weights in network (to descend on optimal values)

At right most level, works much like gradient descent (use error between predicted output and correct output)

At inner layers, we need a way to propagate the error back into earlier levels of the network.

I will present an overview of this algorithm.

Backpropagation Algorithm: Overview

Given the Training Set

Repeat until Theta values on each edge converge:

- For each example in Training Set:
  - Perform Forward-Propagation to determine output of network on this example
  - Compute the error of the output (network output minus expected output)
  - Compute the error terms for every other node in network (backwards through network)
    - **Notice that first layer has no error term because it represents input (never changes)*
  - Compute Delta values for each edge (essentially a cumulative error term at each edge)
  - #END FOR LOOP
- Compute Gradient values as a function of Delta values for each edge
- This Gradient value tells you how much to change each edge weight (theta)

#END MAIN LOOP

Backpropagation Algorithm: Intuition / Notation

Show image on board

Backpropagation Algorithm: Notation

Neural Networks have a lot of notation, let's review!!

$\theta^{(4)}_{23}$ //The weight on the edge headed out of layer 4 from node 3 (in layer 4) to node 2 (in layer 5)
//Superscript is the layer, subscripts are node indices within a layer (top to bottom)

$a^{(4)}_{3}$ //The output of third neuron on layer 4 of the network (heading into layer 5)
//This value is same for all edges coming out of this node, so second subscript not necessary

$z^{(5)}_{23}$ //Input to the 2nd node on layer 5 from the 3rd node on layer 4
//Equal to the $\theta^{(4)}_{23} * a^{(4)}_{23}$ *Plus any other inputs from graph

$\delta^{(2)}_{3}$ //The error associated with the 3rd node on layer 2 of the network

Given the Training Set

Repeat until Theta values on each edge converge:

- For each example in Training Set:
  - Perform Forward-Propagation to determine output of network on this example
  - Compute the error of the output (network output minus expected output)
  - Compute the error terms for every other node in network (backwards through network)
    - **Notice that first layer has no error term because it represents input (never changes)*
  - Computer Delta values for each edge (essentially a cumulative error term at each edge)
  - #END FOR LOOP
- Compute Gradient values as a function of Delta values for each edge
- This Gradient value tells you how much to change each edge weight (theta)

#END MAIN LOOP

Given the Training Set

Repeat until Theta values on each edge converge:

For each example in Training Set:

Perform Forward-Propagation to determine output of network on this example

Compute the error of the output (network output minus expected output)

Compute the error terms for every other node in network (backwards through network)

Notice that first layer has no error term because it represents input (never changes)

Computer Delta values for each edge (essentially a cumulative error term at each edge)

#END FOR LOOP

Compute Gradient values as a function of Delta values for each edge

This Gradient value tells you how much to change each edge weight (theta)

#END MAIN LOOP

$$\delta^{(l)}{}_j = a^{(l)}{}_j - yj$$

Talking about last layer's output only.

The error is simply the difference between actual and expected output

'l' here represents the last layer of the network only

Given the Training Set

Repeat until Theta values on each edge converge:

For each example in Training Set:

Perform Forward-Propagation to determine output of network on this example

Compute the error of the output (network output minus expected output)

Compute the error terms for every other node in network (backwards through network)

Notice that first layer has no error term because it represents input (never changes)

Computer Delta values for each edge (essentially a cumulative error term at each edge)

#END FOR LOOP

Compute Gradient values as a function of Delta values for each edge

This Gradient value tells you how much to change each edge weight (theta)

#END MAIN LOOP

$$\delta^{(3)}_j = Sum((\theta^{(3)}_{ij}) * \delta^{(4)}_i) * (g'(z^{(3)}_j))$$

Compute this for every layer down to 2 (1 is the input layer so no error associated there)
*3 is just an example here, really the layer should be generalized

$g'(z^{(3)}_j) = a^{(3)}_j * (1-a^{(3)}_j)$ //trust me, this is the derivative of g

Given the Training Set

Repeat until Theta values on each edge converge:

For each example in Training Set:

Perform Forward-Propagation to determine output of network on this example

Compute the error of the output (network output minus expected output)

Compute the error terms for every other node in network (backwards through network)

**Notice that first layer has no error term because it represents input (never changes)*

Compute Delta values for each edge (essentially a cumulative error term at each edge)

#END FOR LOOP

Compute Gradient values as a function of Delta values for each edge

This Gradient value tells you how much to change each edge weight (theta)

#END MAIN LOOP

$$\Delta^{(l)}_{ij} \mathrel{+}= a^{(l)}_{j} * \delta^{(l+1)}_{i}$$

These values start at 0 and are accumulated over all training examples. So don't reset them after each training example

Note there is one of these for each edge in the network

Given the Training Set

Repeat until Theta values on each edge converge:

For each example in Training Set:

Perform Forward-Propagation to determine output of network on this example

Compute the error of the output (network output minus expected output)

Compute the error terms for every other node in network (backwards through network)

Notice that first layer has no error term because it represents input (never changes)

Compute Delta values for each edge (essentially a cumulative error term at each edge)

#END FOR LOOP

Compute Gradient values as a function of Delta values for each edge

This Gradient value tells you how much to change each edge weight (theta)

#END MAIN LOOP

$$D^{(l)}_{ij} = \left(\frac{1}{m}\right) * \Delta^{(l)}_{ij}$$

This is (again, trust me) the value of the partial derivative with respect to theta on each edge. So…just update theta by this value.

Most of the time we include a regularization term, I'm leaving this out for now, but don't be scared if you see that in a reading

Interesting Video

Neural Network to play Super Mario World:

http://digg.com/video/computer-program-learns-how-to-play-mario