University of Southern California

ECE 105: Introduction to Electrical Engineering

Lecture 19-20

Neural Networks

Yasser Khan

Rehan Kapadia

Timeframe

USC University of Southern California

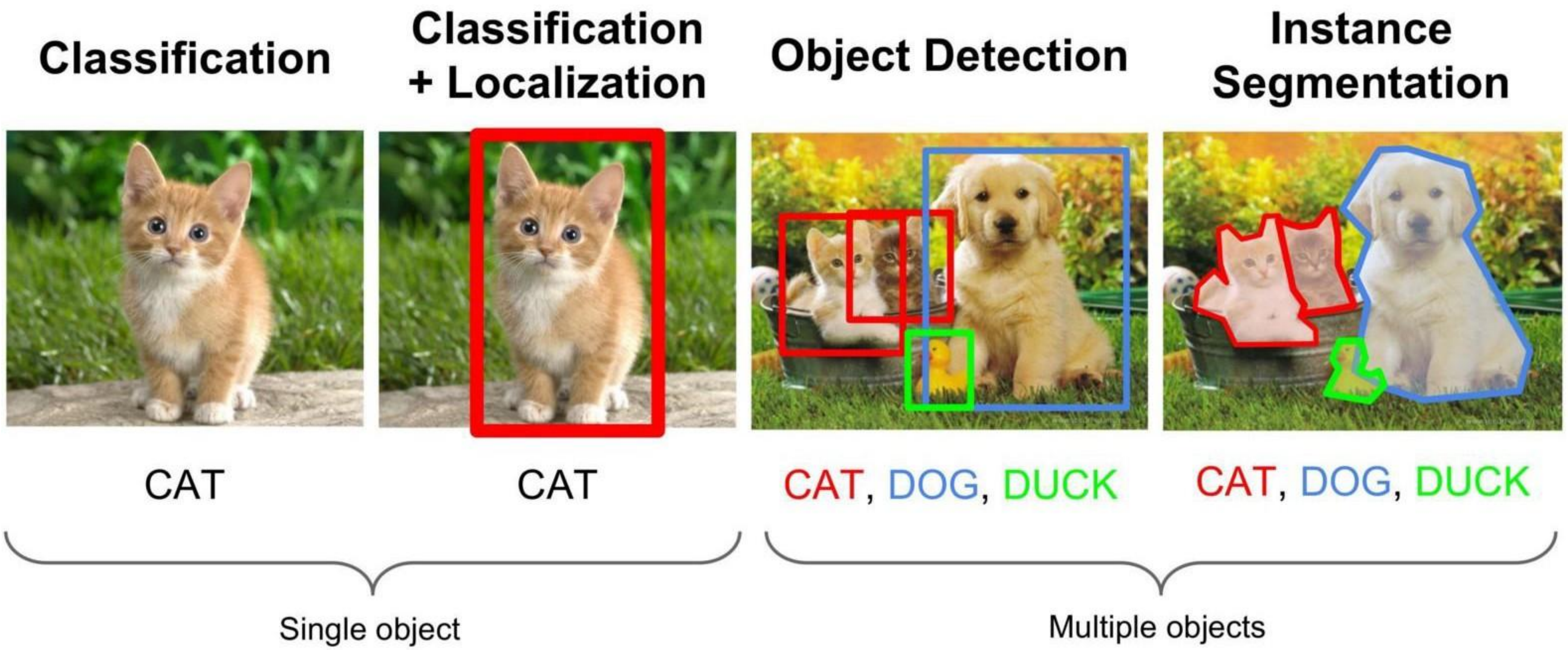
- 1943: McCulloch–Pitts “neuron”
  - Started the field
- 1962: Rosenblatt’s perceptron
  - Learned its own weight values; convergence proof
- 1969: Minsky & Papert book on perceptrons
  - Proved limitations of single-layer perceptron networks
- 1982: Hopfield and convergence in symmetric networks
  - Introduced energy-function concept
- 1986: Backpropagation of errors
  - Method for training multilayer networks
- Present: Probabilistic interpretations, Bayesian and spiking networks

Timeframe

- 1943: McCulloch – Pitts “neuron”
  - Started the field
- 1962: Rosenblatt’s perceptron
  - Learned its own weight values; convergence proof
- 1969: Minsky & Papert book on perceptrons
  - Proved limitations of single-layer perceptron networks
- 1982: Hopfield and convergence in symmetric networks
  - Introduced energy-function concept
- 1986: Backpropagation of errors
  - Method for training multilayer networks
- Present: Probabilistic interpretations, Bayesian and spiking networks

Why Deep Neural Networks?

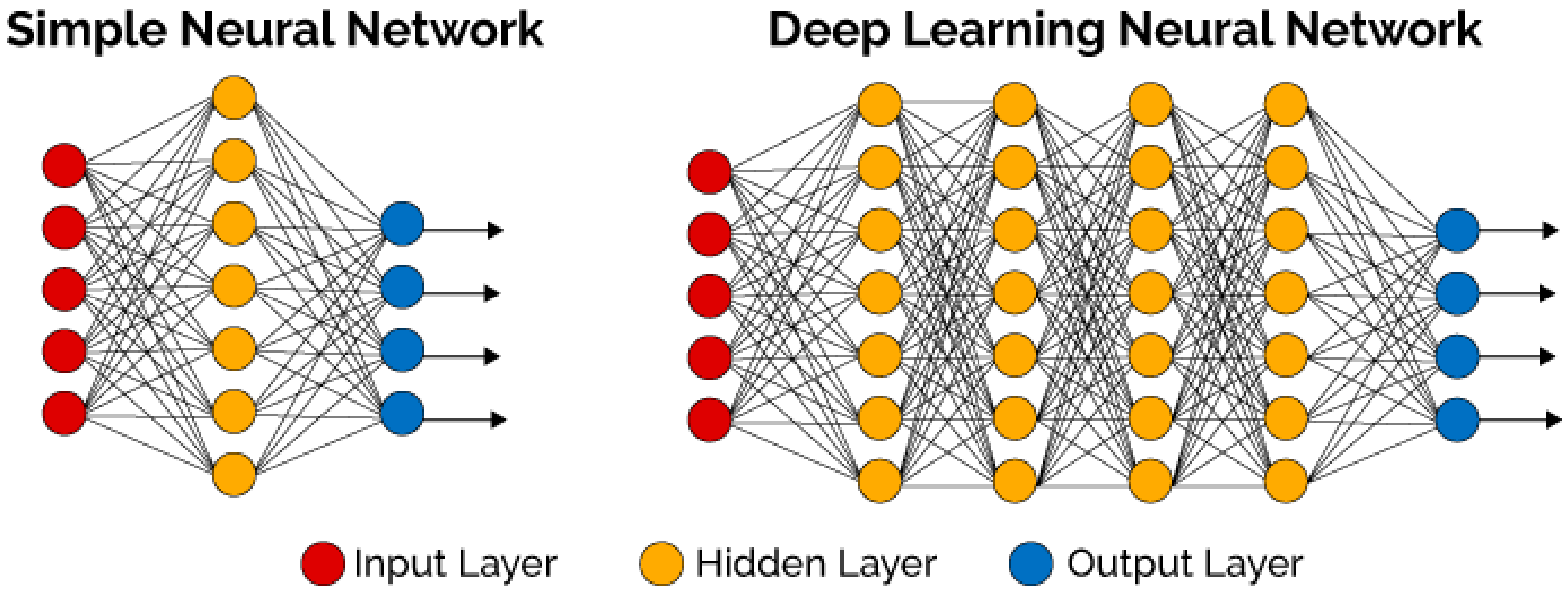
USC University of Southern California



- Deep neural networks are excellent for certain classes of problems, such as image recognition, speech recognition etc.
- Importantly, as the amount of data scales, the performance of deep learning continues to improve

Image from Medium: “Review of Deep Learning Algorithms for Object Detection”

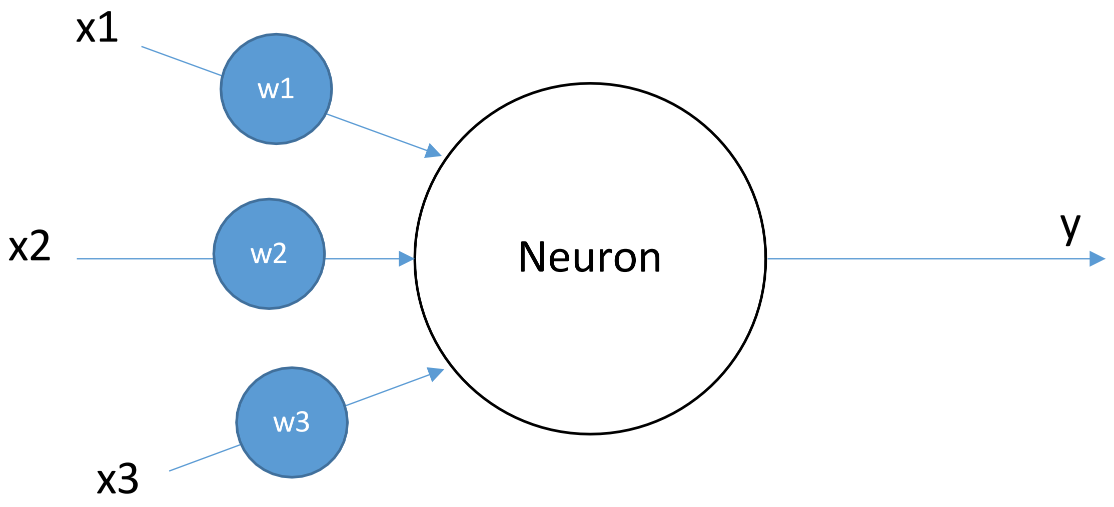
What's so deep about Deep Neural Networks?



- The "deep" refers to the number of layers of 'units' between the input layer and output layer.
- Since this is a neural network, the units are called neurons

https://becominghuman.ai/deep-learning-made-easy-with-deep-cognition-403fbe445351

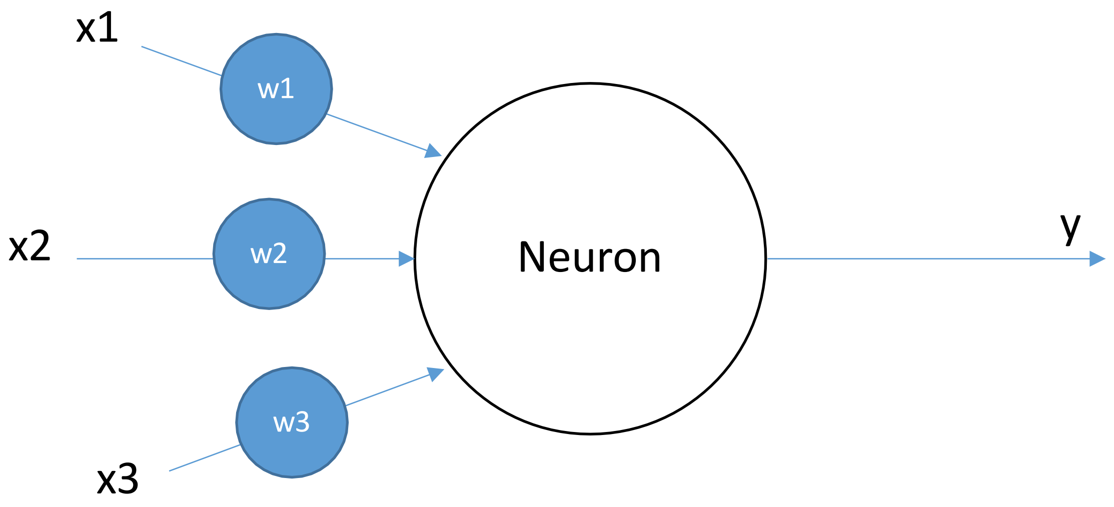
Fundamental Unit of a Neural Network



Mathematical Definition Y = f(w1*x1 + w2*x2 + w3*x3 + b)

- The above is the fundamental unit of a neural network
  - W_i refers to the weights, sometimes called synaptic weights
  - The neuron is the mathematical function that connects inputs to outputs
  - X_i refers to the inputs
  - Y refers to the output

Fundamental Unit of a Neural Network

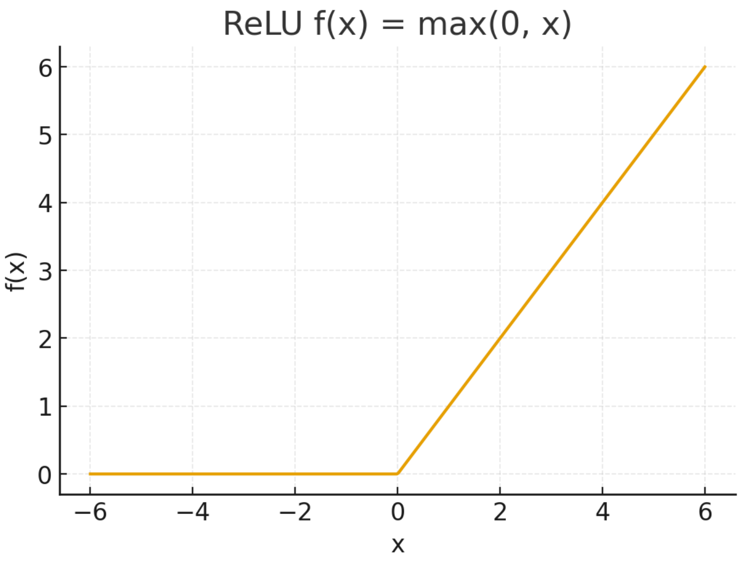


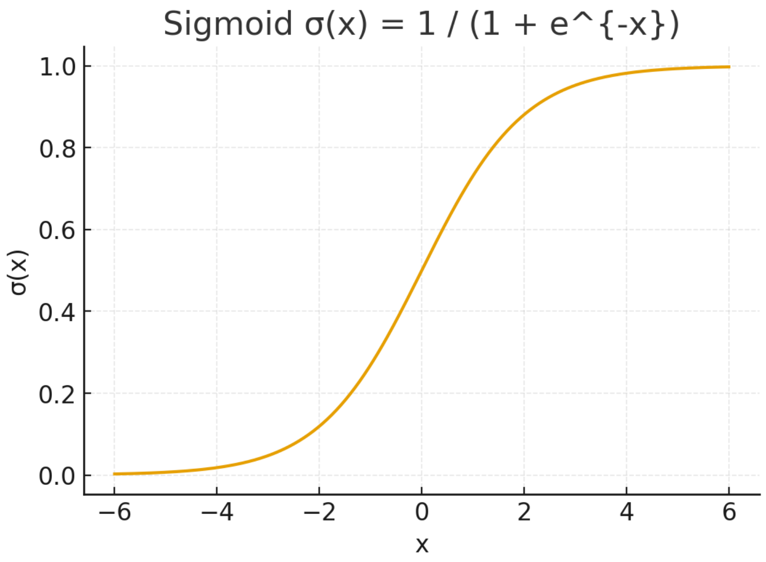
Mathematical Definition Y = f(w1*x1 + w2*x2 + w3*x3 + b)

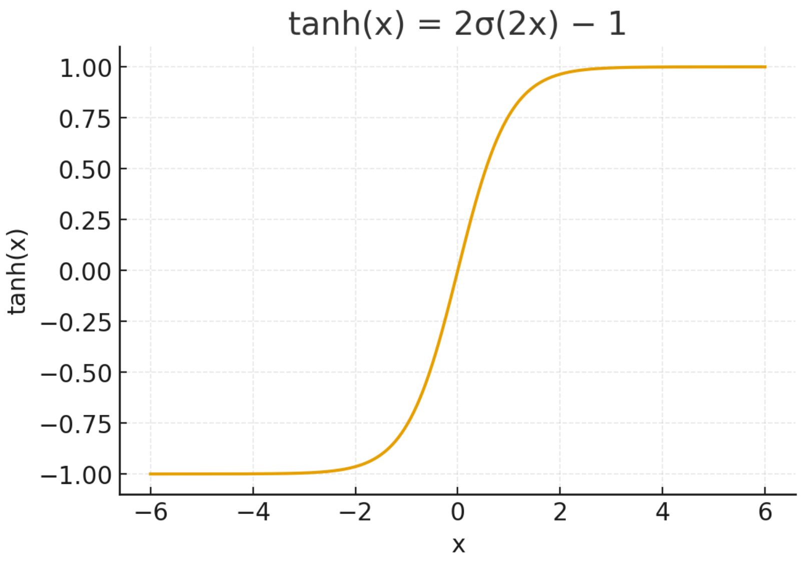
- Many different functions are used for the neuron, but the three most popular are
  - Sigmoid: $\sigma(x) = 1/(1+\exp(-x))$
  - Rectified Linear Unit (ReLU): $f(x) = \max(0,x)$
  - tanh: $\tanh(x) = 2\sigma(2x) - 1$

Activation functions of a Neural Network

University of Southern California







Rectified Linear Unit (ReLU)

$$\mathrm{ReLU}(x) = \max(0, x)$$

- Range: $[0, \infty)$
- Shape: 0 for negatives, linear for positives
- Pros: Simple and cheap; mitigates vanishing gradients on $x > 0$
- Cons: "Dead ReLU" when neurons get stuck at 0 (weights push inputs negative)

Sigmoid

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

- Range: $(0, 1)$
- Shape: "S"-curve; compresses large $|x|$ toward 0 or 1
- Pros: Probabilistic interpretation for binary classification logits
- Cons: Saturates for large $|x|$ → small gradients ("vanishing gradients"); outputs not zero-centered

Hyperbolic tangent (tanh)

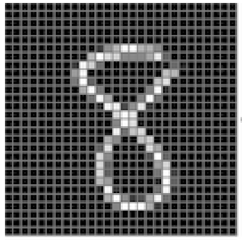
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \;=\; 2\,\sigma(2x) - 1$$

- Range: $(-1, 1)$
- Shape: S-curve but zero-centered (often trains better than sigmoid for some RNN/MLP setups)
- Cons: Still saturates at large $|x|$ → potential vanishing gradients

Operation of a neural network

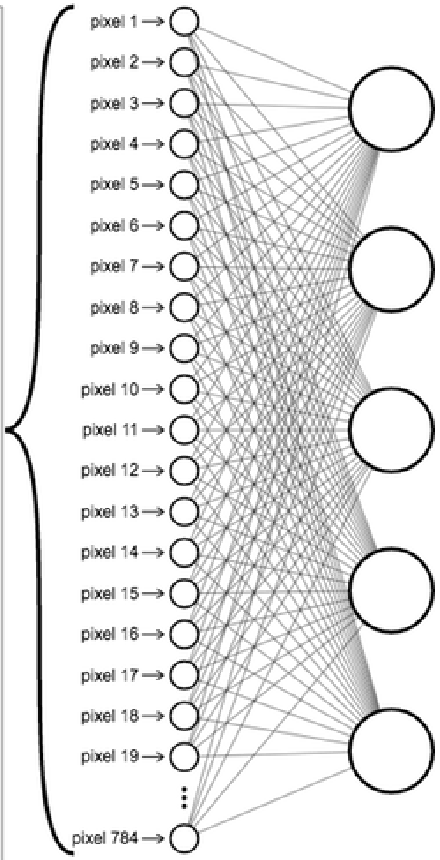
USC
University of
Southern California

pixel



28 x 28
784 pixels

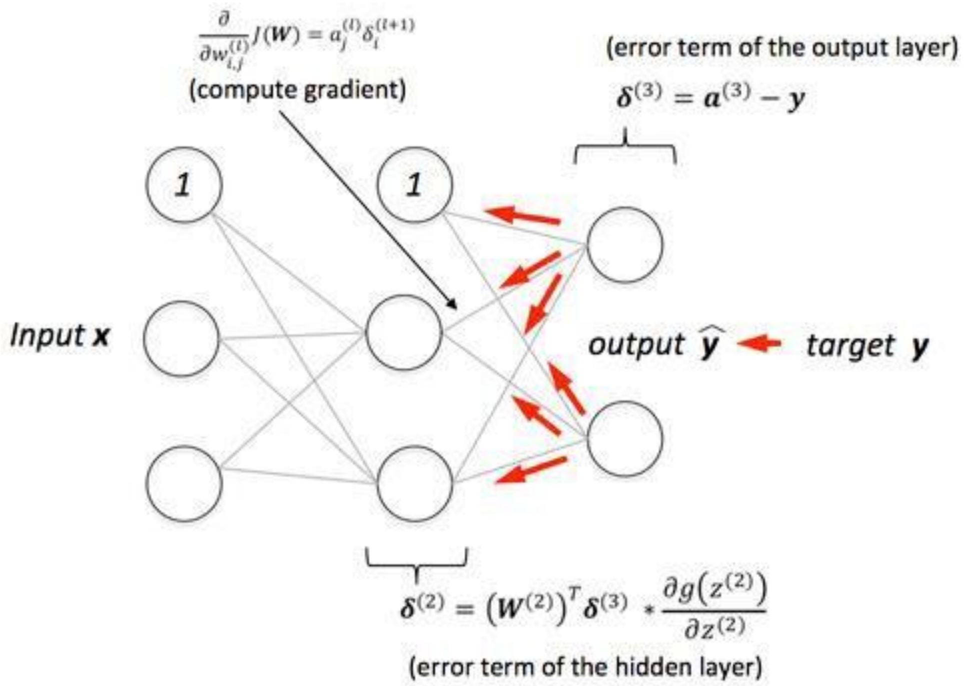
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



- Image is made up of a set number of pixels
  - Each pixel becomes an input
  - Example here: each pixel is from 0 to 256 in greyscale
  - Each input is connected to all the neurons of the hidden layer
  - Adjusting the weights will then eventually allow recognition of the digits

https://becominghuman.ai/deep-learning-made-easy-with-deep-cognition-403fbe445351

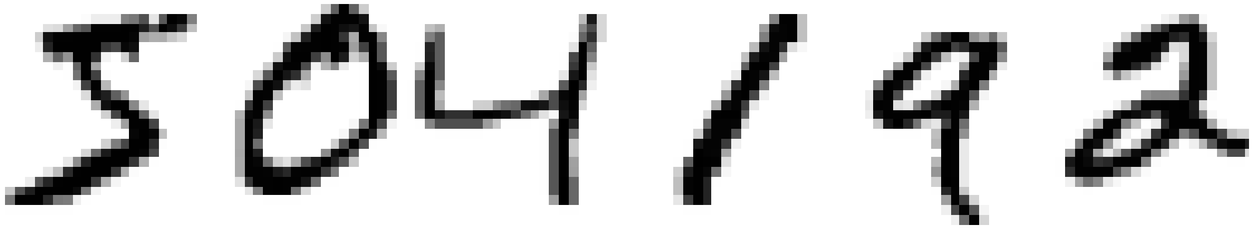
Training of an Artificial Neural Network



- Backpropagation algorithm a form of supervised learning
  - We initialize all the weights of the ANN, and then feed the data into it
  - Each training example generates an output.
  - We know the correct output for those training samples, so for each output, we can define an error
  - Then, we can essentially go backwards, assigning some part of that error to the weights between output neuron and previous neurons
  - Adjusting the weights allows us to move towards the correct answer, as defined by us

https://github.com/rasbt/python-machine-learning-book/blob/master/faq/visual-backpropagation.md

Handwriting Recognition

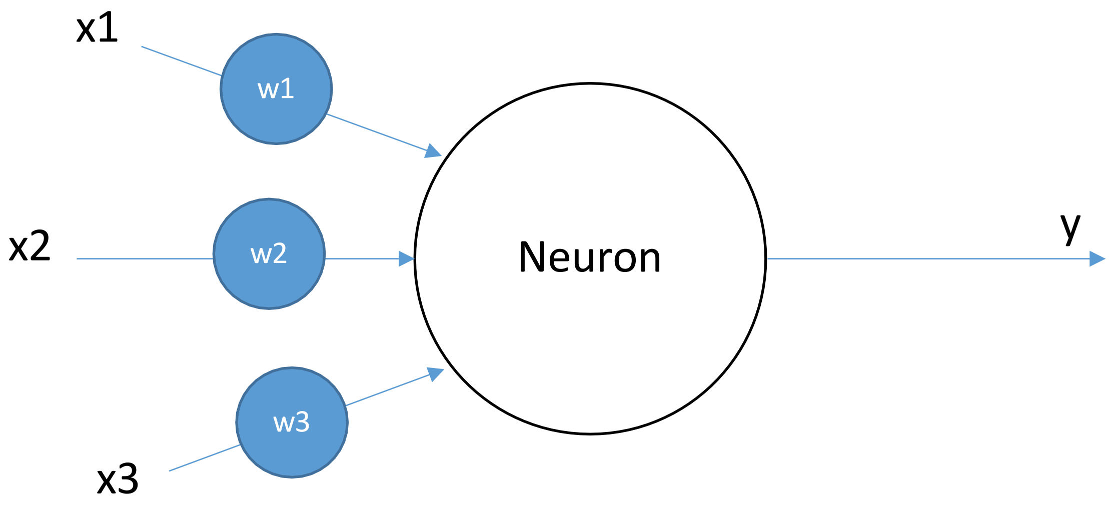


- We can easily recognize these numbers
  - But this uses a huge number of neurons (~billion or more) and an even larger number of connections/synapses (100’s of billions)

Approach

- Need a large set of training data
  - Also need a set on which to test your trained model

Perceptron

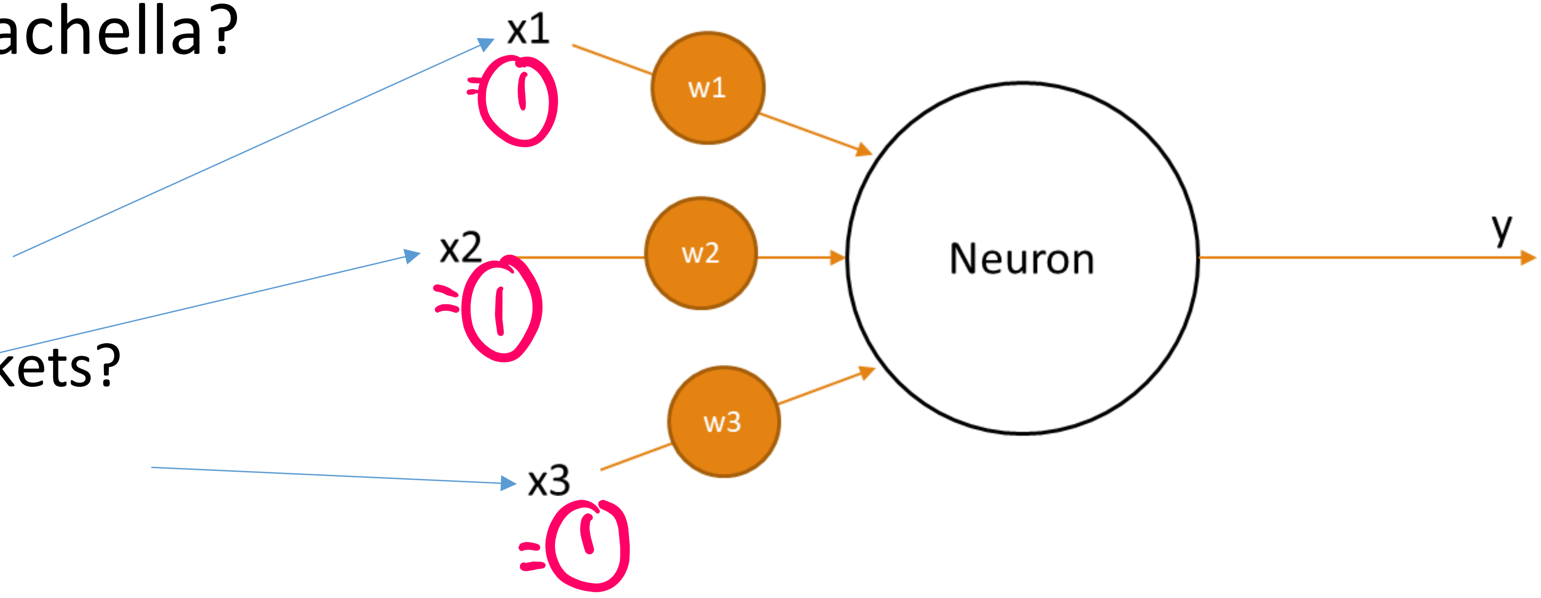


Mathematical Definition Y = f(w1*x1 + w2*x2 + w3*x3 + b)

- For the perceptron:
  - Output = 0 if $\Sigma w*x \leq$threshold
  - Output = 1 if $\Sigma w*x >$threshold

What can we answer with a perceptron?

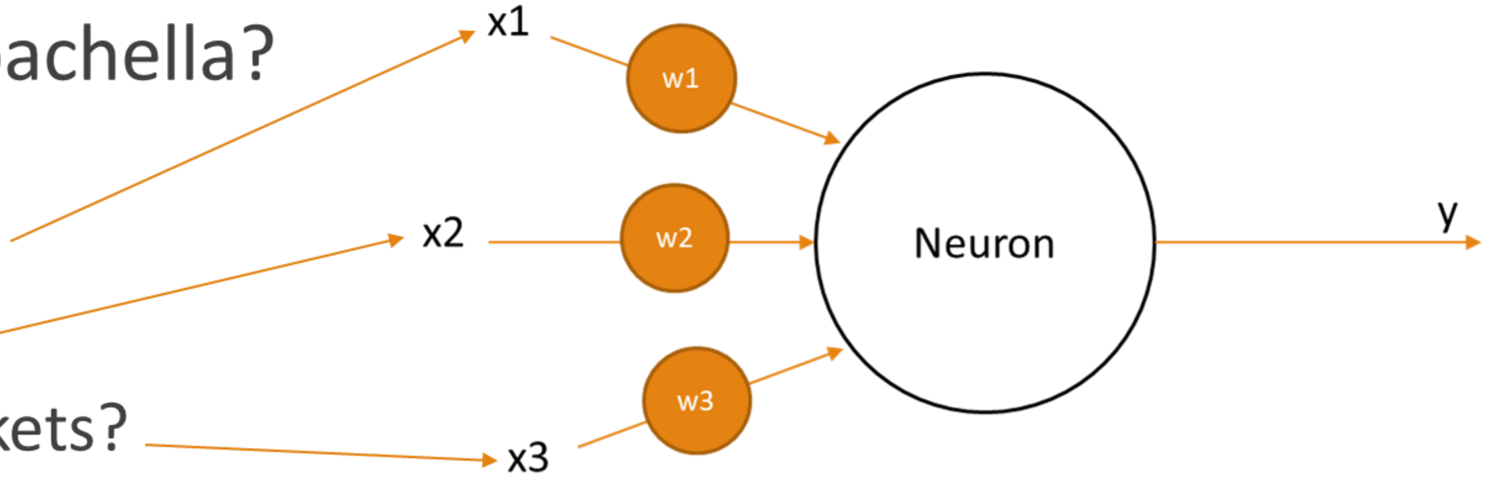
- Do we want to go to Coachella?
  - Factor 1: Good weather?
  - Factor 2: Friends going?
  - Factor 3: Do we have tickets?



How would we decide this?

Do we want to go to Coachella?

- Factor 1: Good weather?
- Factor 2: Friends going?
- Factor 3: Do we have tickets?

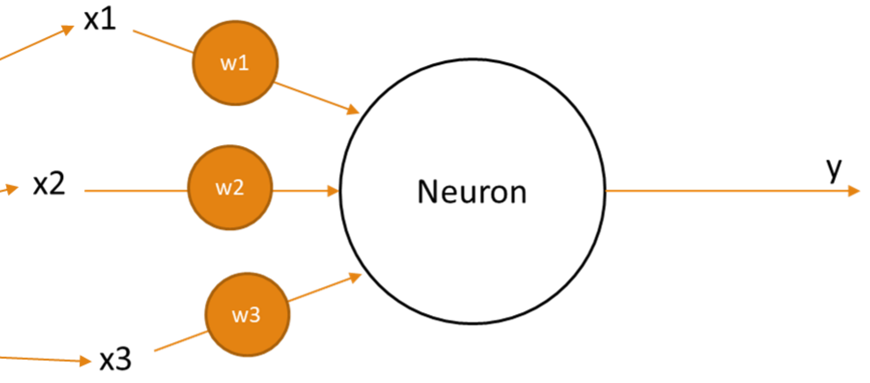


- Each variable is either a 1 or 0
  - The decision making happens in the weights
  - Here we have to assign weights and a threshold value

How would we decide this?

Do we want to go to Coachella?

- Factor 1: Good weather?
- Factor 2: Friends going?
- Factor 3: Do we have tickets?



Threshold = 10

w1 = 5

w2 = 5

w3 = 11

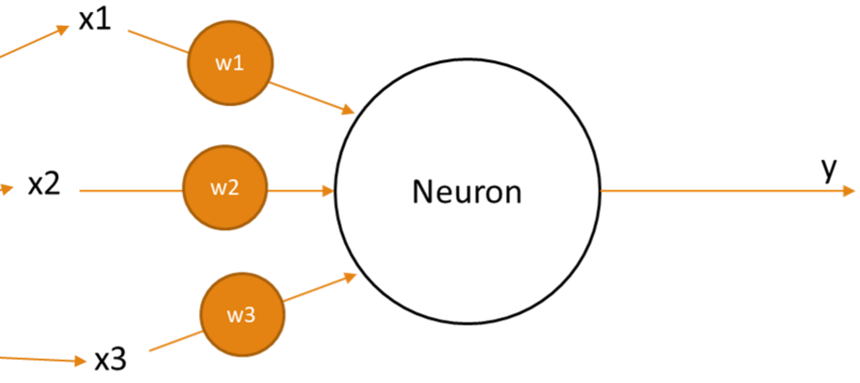
Determines the output

This is a silly distribution. Either you have tickets and you're going, or you don't and you aren't.

How would we decide this?

Do we want to go to Coachella?

- Factor 1: Good weather?
- Factor 2: Friends going?
- Factor 3: Do we have tickets?



Threshold = 12

w1 = 5

w2 = 5

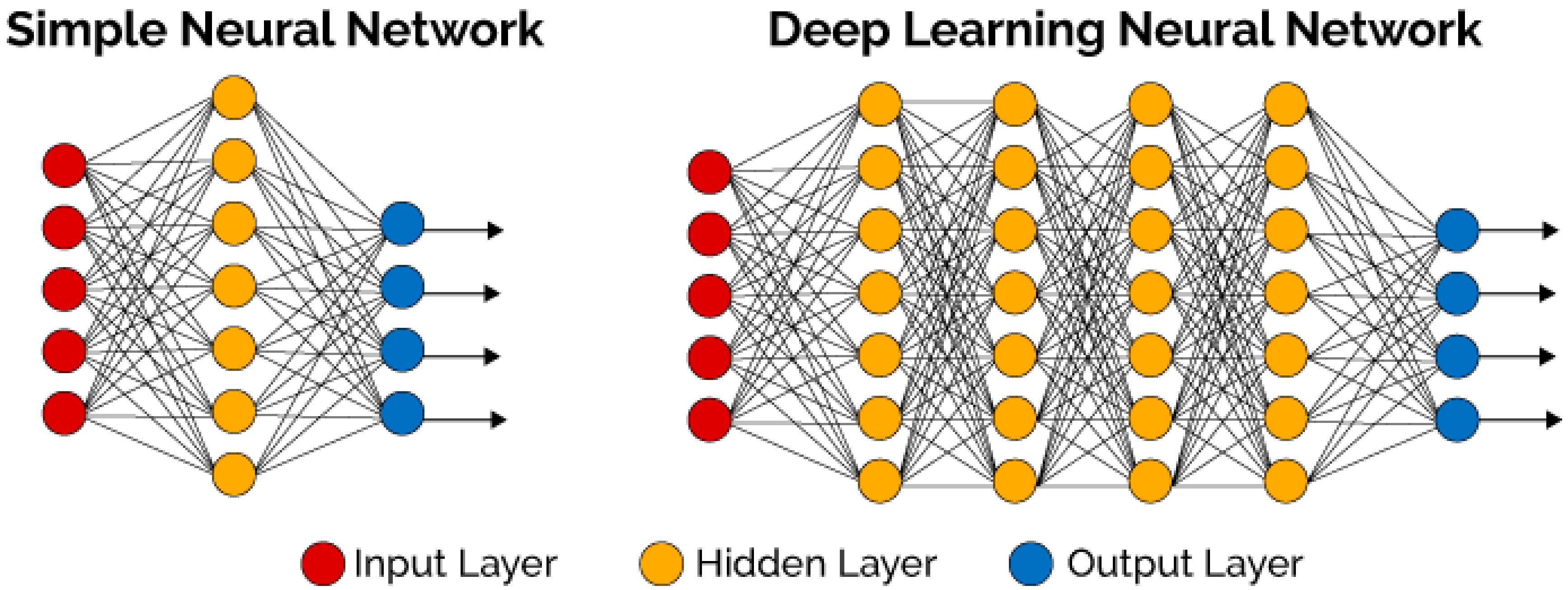
w3 = 11

NO

Now this basically says that we only go if we have tickets AND one of the other conditions are true, but not both of them.

This is a silly example, but a reasonable starting point to think about how we build up complex decision making.

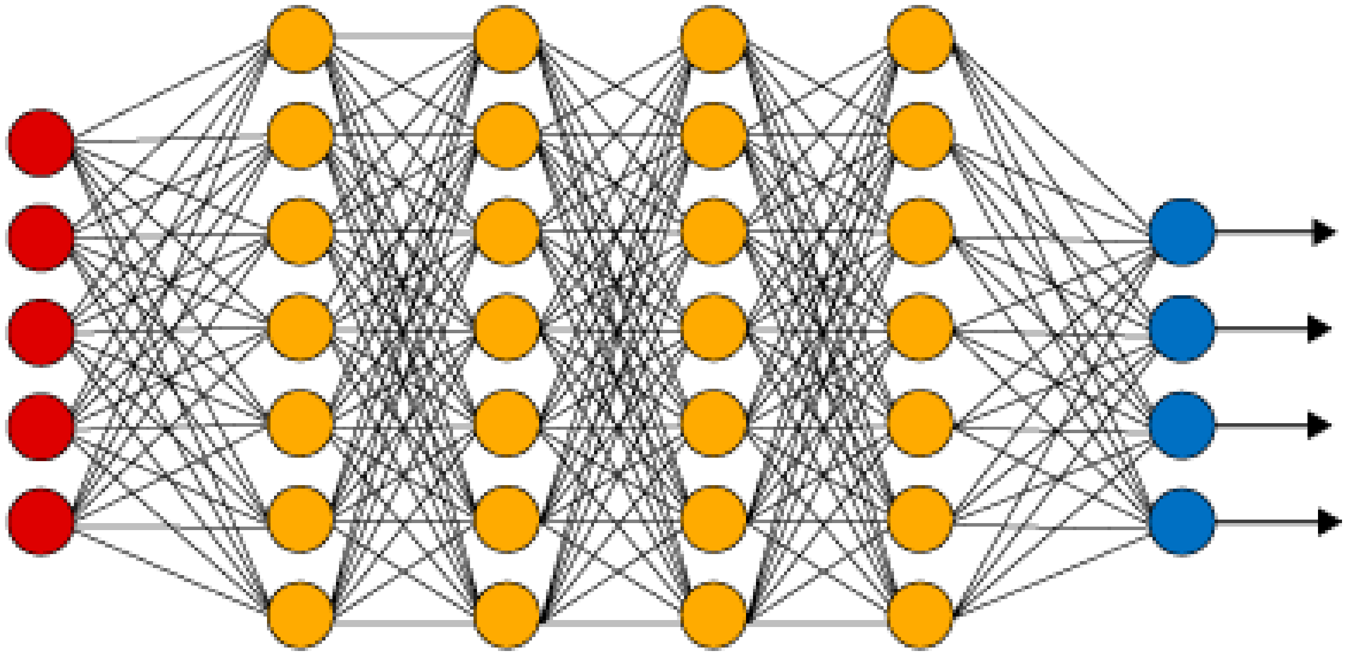
Addition of other nodes increases complexity



- We can now make decisions based on more inputs, and have a more nuanced way of getting to the outputs.

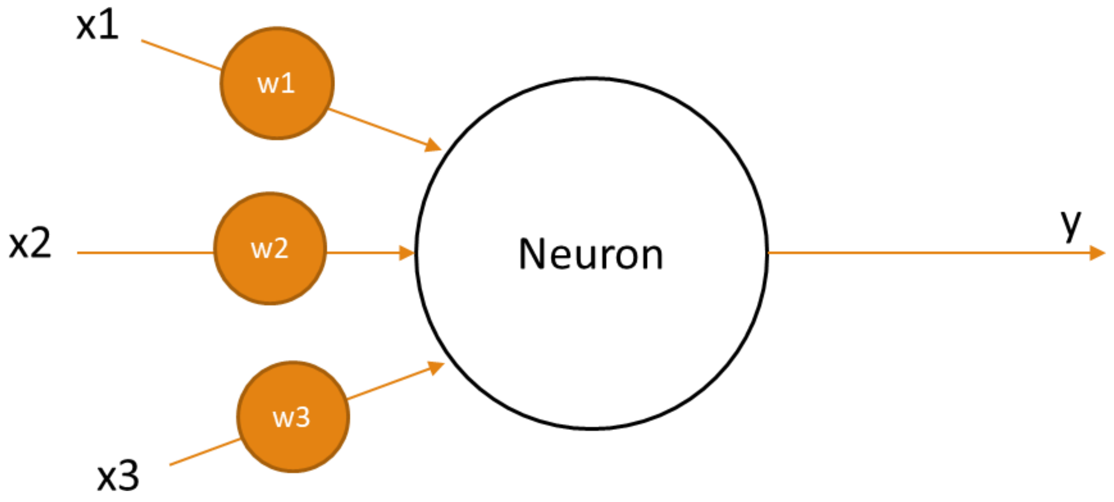
How does information change through layers?

USC
University of Southern California



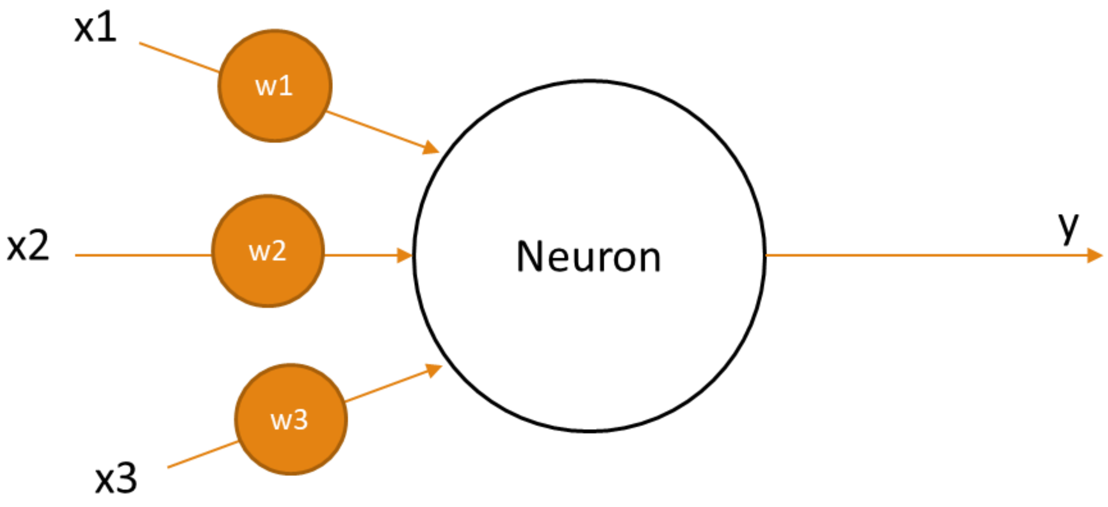
- As we move through the layers of the DNN, we are somewhat making decisions on more and more abstract items

Perceptrons



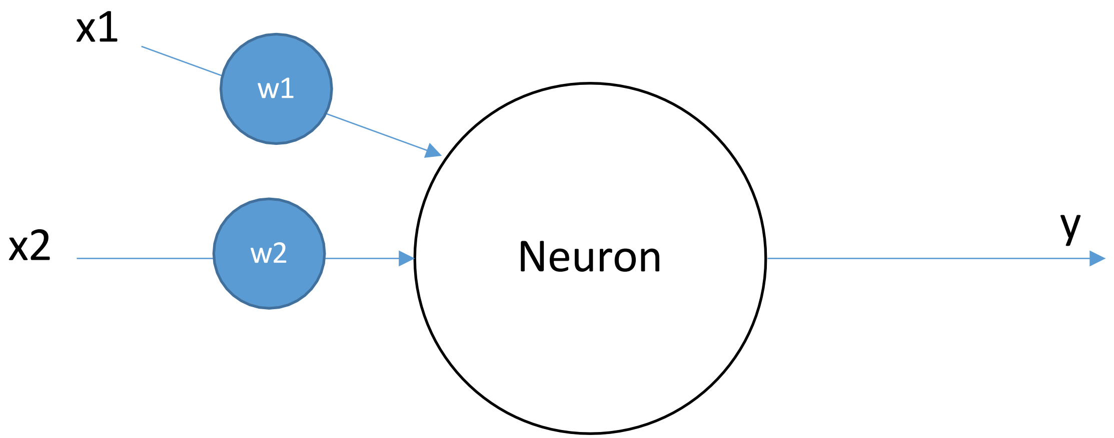
- We can assume
  - Inputs as a vector x = {x1, x2, x3}
  - Weights as vector w = {w1, w2,w3}
  - Then sum is just the dot product, and the become
    - 0 when when $x{\cdot}w+b \leq 0$
    - 1 when when $x{\cdot}w+b > 0$

Perceptrons



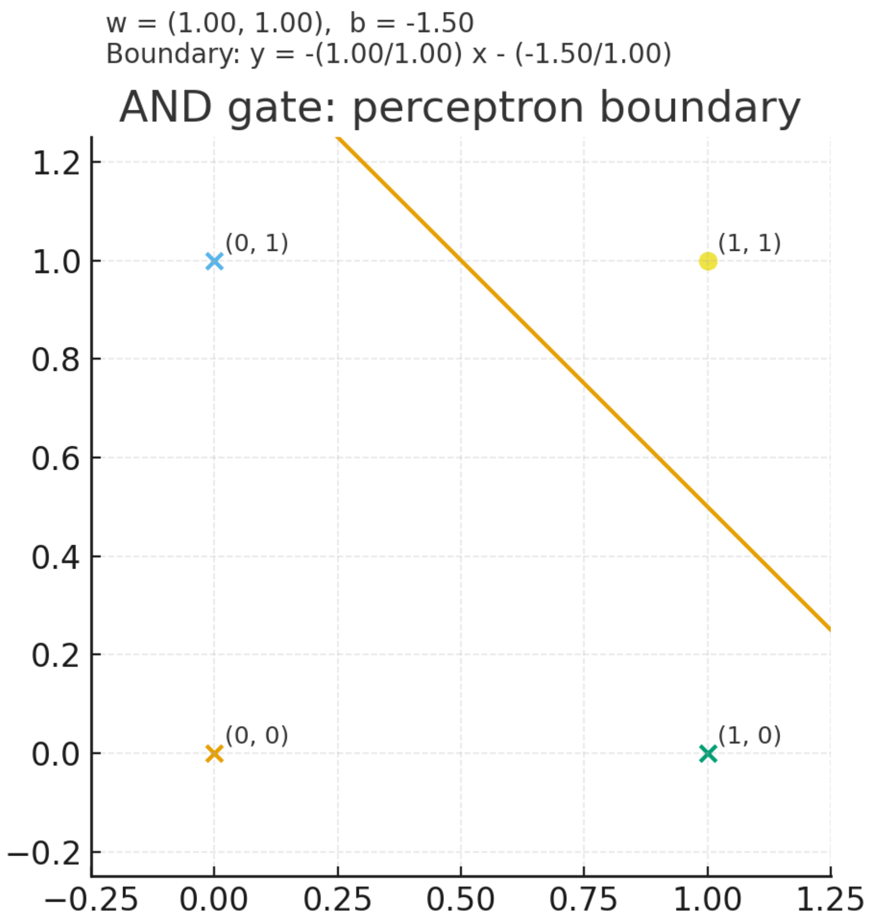
- 0 when when $x \bullet w + b \leq 0$
- 1 when when $x \bullet w + b > 0$
  - b can be thought of the bias
  - Large positive b -> neuron that fires on most inputs
  - Negative b-> neuron that rarely fires

Perceptrons for logical functions?



- How could we implement
  - AND
  - OR
  - NAND

AND gate



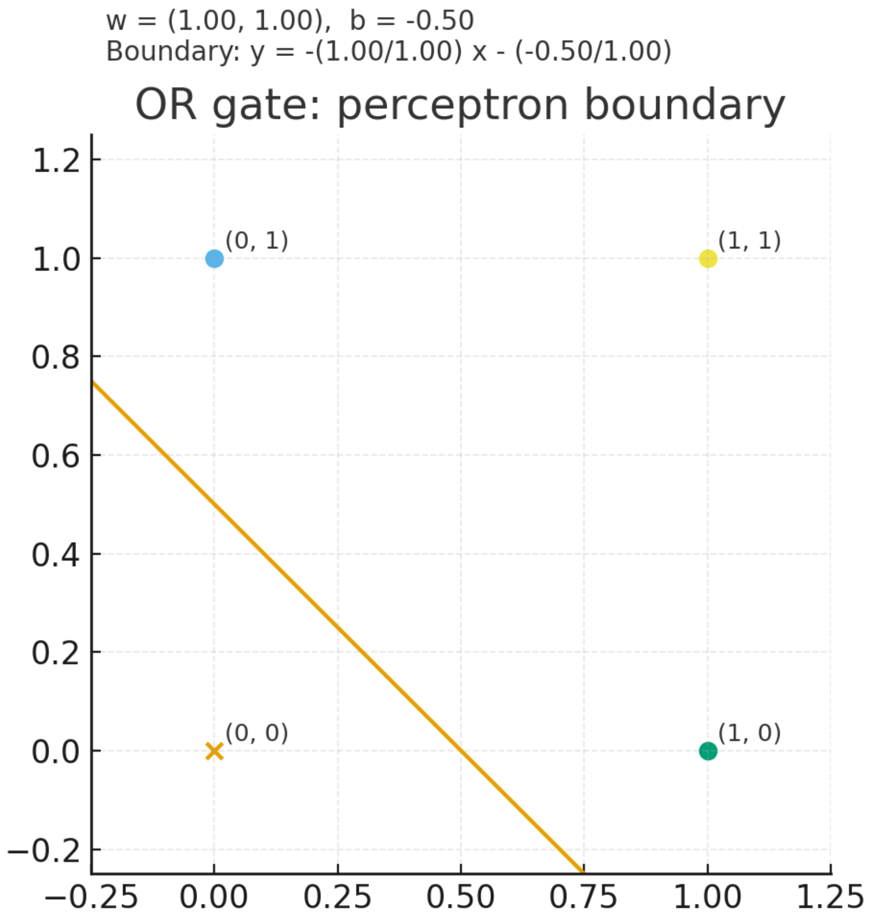
AND gate

Weights/bias: $w_1 = 1,\ w_2 = 1,\ b = -1.5$

x_1	x_2	$w \cdot x + b$	Output
0	0	$0 + 0 - 1.5 = -1.5$	0
0	1	$0 + 1 - 1.5 = -0.5$	0
1	0	$1 + 0 - 1.5 = -0.5$	0
1	1	$1 + 1 - 1.5 = 0.5$	1

Decision boundary: $x_1 + x_2 = 1.5$. Only $(1, 1)$ crosses.

OR gate



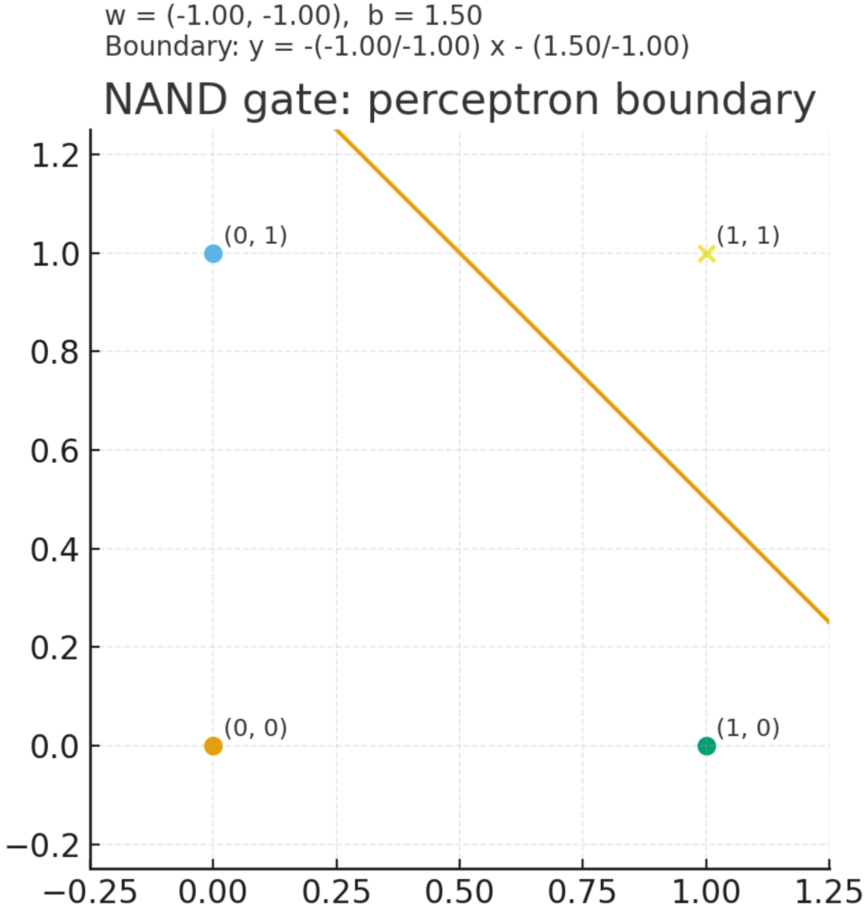
OR gate

Weights/bias: $w_1 = 1,\ w_2 = 1,\ b = -0.5$

x_1	x_2	$w \cdot x + b$	Output
0	0	$0 + 0 - 0.5 = -0.5$	0
0	1	$0 + 1 - 0.5 = 0.5$	1
1	0	$1 + 0 - 0.5 = 0.5$	1
1	1	$1 + 1 - 0.5 = 1.5$	1

Boundary: $x_1 + x_2 = 0.5$. Any input with at least one 1 is positive.

NAND gate



NAND gate

Weights/bias: $w_1 = -1,\ w_2 = -1,\ b = 1.5$

x_1	x_2	$w \cdot x + b$	Output
0	0	$0 + 0 + 1.5 = 1.5$	1
0	1	$0 - 1 + 1.5 = 0.5$	1
1	0	$-1 + 0 + 1.5 = 0.5$	1
1	1	$-1 - 1 + 1.5 = -0.5$	0

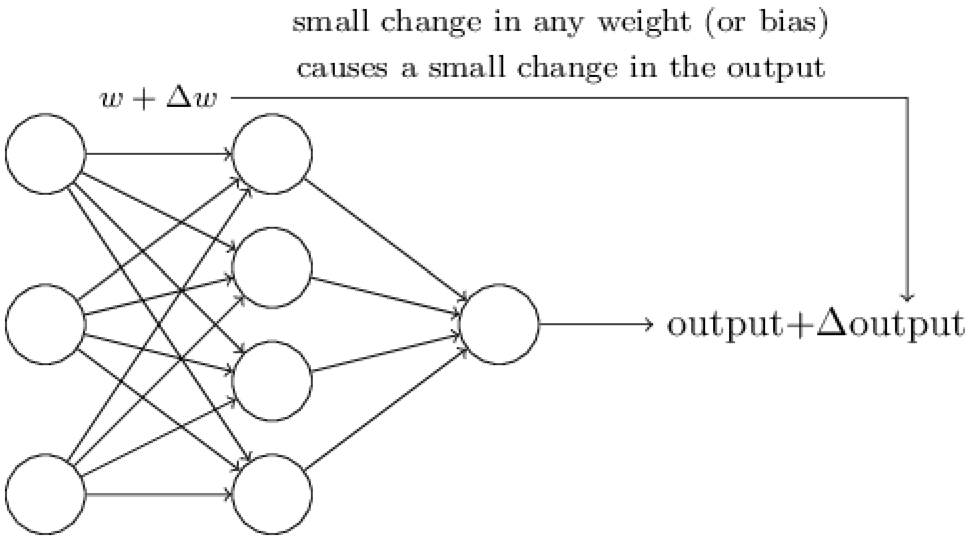
This is the logical NOT of AND (i.e., 1 except at $(1, 1)$).

So are perceptrons just more complex NAND gates?

University of Southern California

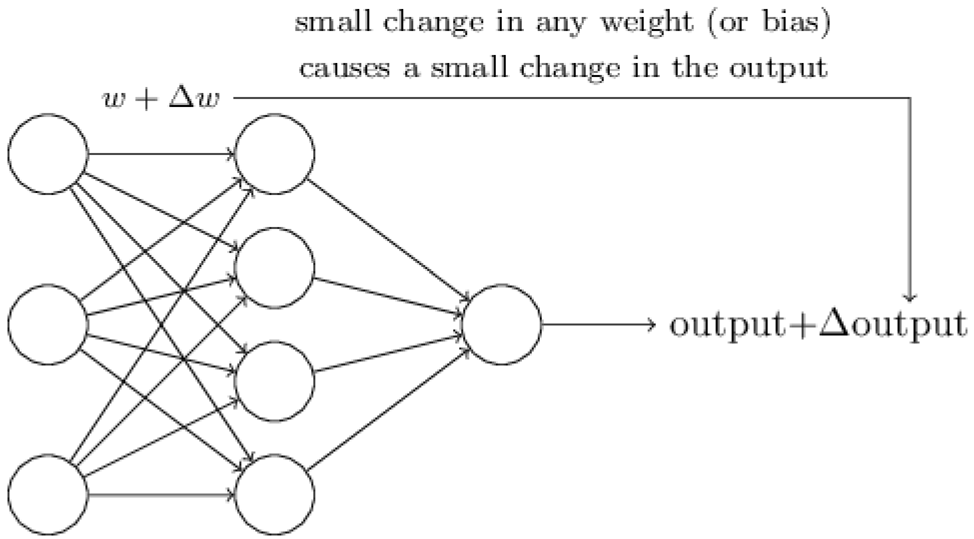
- No
  - The key strength lies in the algorithms which allow us to tune the weights
  - It also needs to be something that's done with just the input, and not something that relies on a human
- What's so important about this?
  - When we create a circuit, we basically presolve some set of problems
  - Then we lay out the circuit to solve those
  - However, with our 'learning' approach, we don't have to do that. We just create some basic architecture, and let it automatically tune itself to get the right answer

Sigmoid Neurons



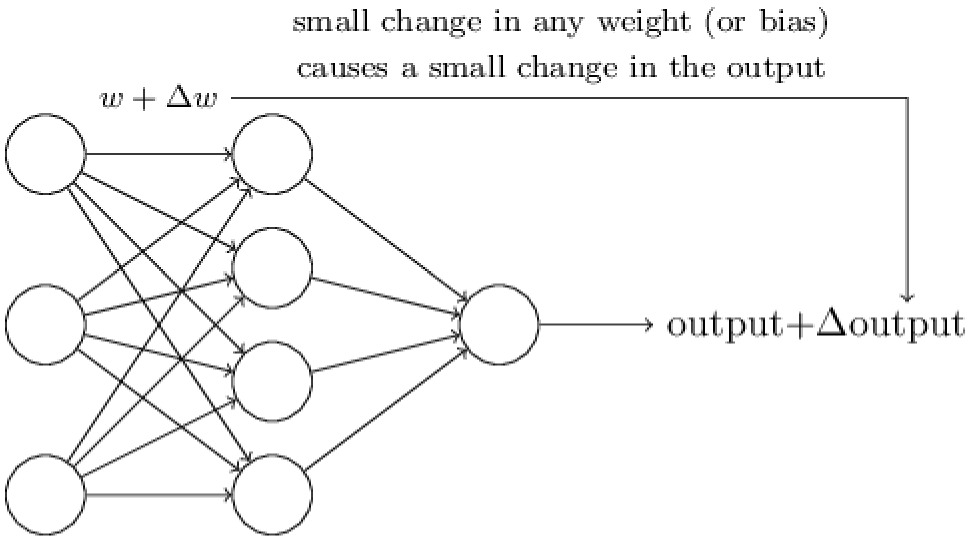
- If we want to learn, then we should have a small change in weight in some synapse to have a corresponding small change in the output

Sigmoid Neurons



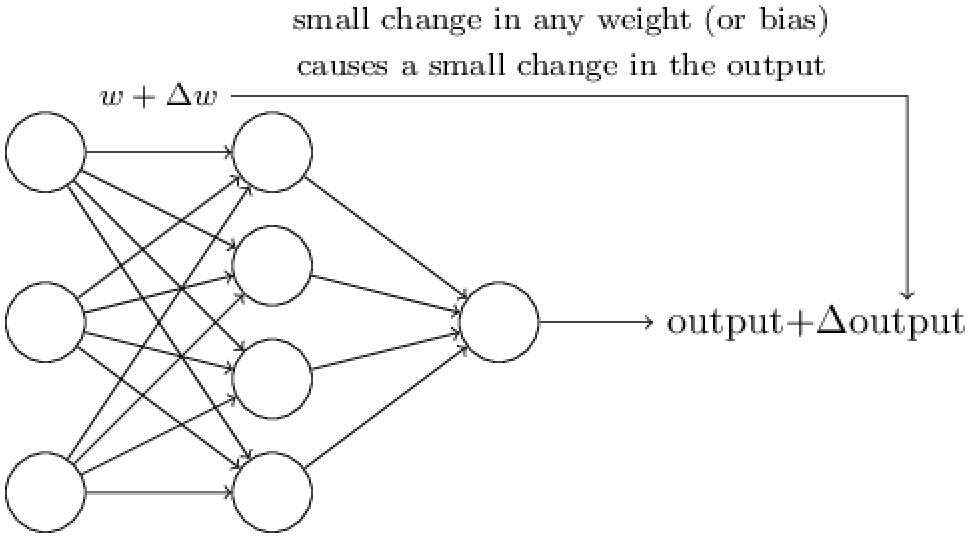
- Why is this important?
  - Let’s say we feed an input which is a handwritten digit
    - It’s a 3
  - Our NN classifies it as a 4…
    - It would be great if we could just change it slightly and have a small change in our output

Sigmoid Neurons



- Why is this important?
  - But if we have a perceptron, then any small weight change can invert the neuron output
    - This then propagates down the NN and can radically change everything in a fashion that is complex to precalculate
  - You fixed your incorrect classification of the 3, but messed up others

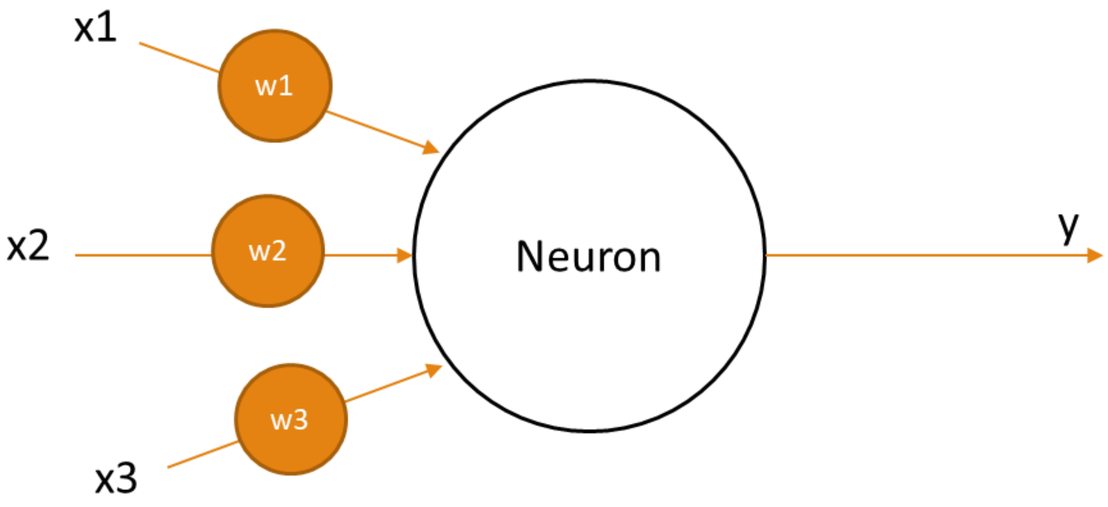
Sigmoid Neurons



- Why is this important?
  - So with perceptrons, it is challenging to have an algorithm that will allow you to easily learn complex data
  - To circumvent this, the sigmoid neuron was introduced

Sigmoid Neurons

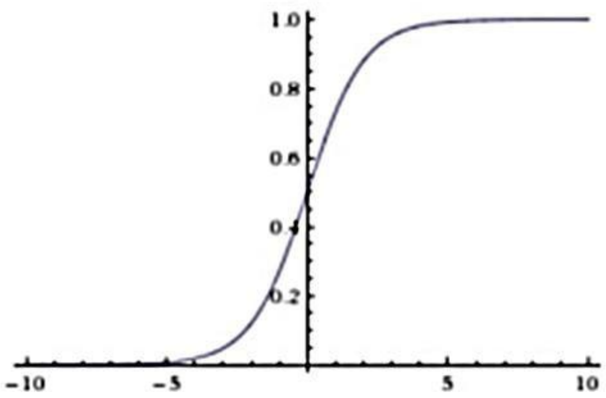
USC University of Southern California



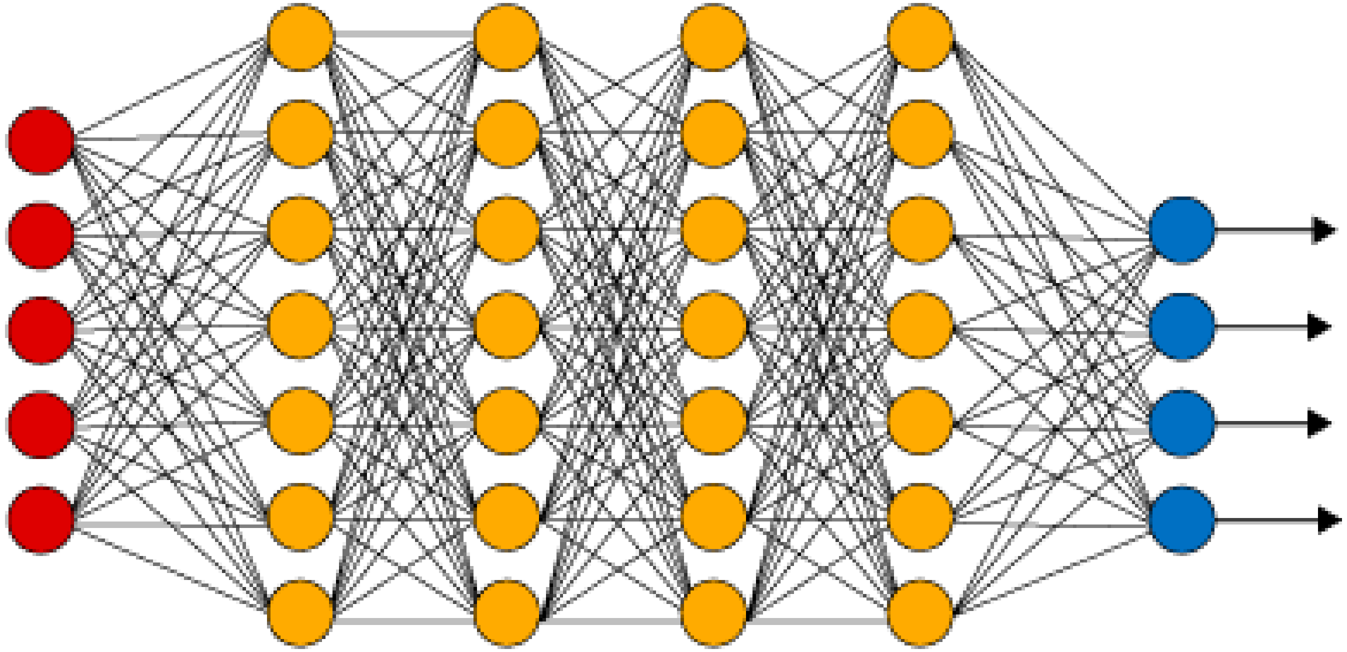
- Looks the same as before…
  - However, inputs can now be anything between 0 and 1, not just 0 or 1.
  - Equation is now more complex:
    - $\sigma(x \cdot w+b)=\sigma(z)= \frac{1}{1+e^{-z}}$

Similarities between Perceptrons and Sigmoids

- $\sigma(x\cdot w+b)=\sigma(z)=\frac{1}{1+e^{-z}}$
  - When z is large positive number, then $\sigma(z)$ ~1
  - When z is large negative number, then $\sigma(z)$ ~0
  - Key difference is that we have smoothed out any sudden changes…



Sigmoid Neurons

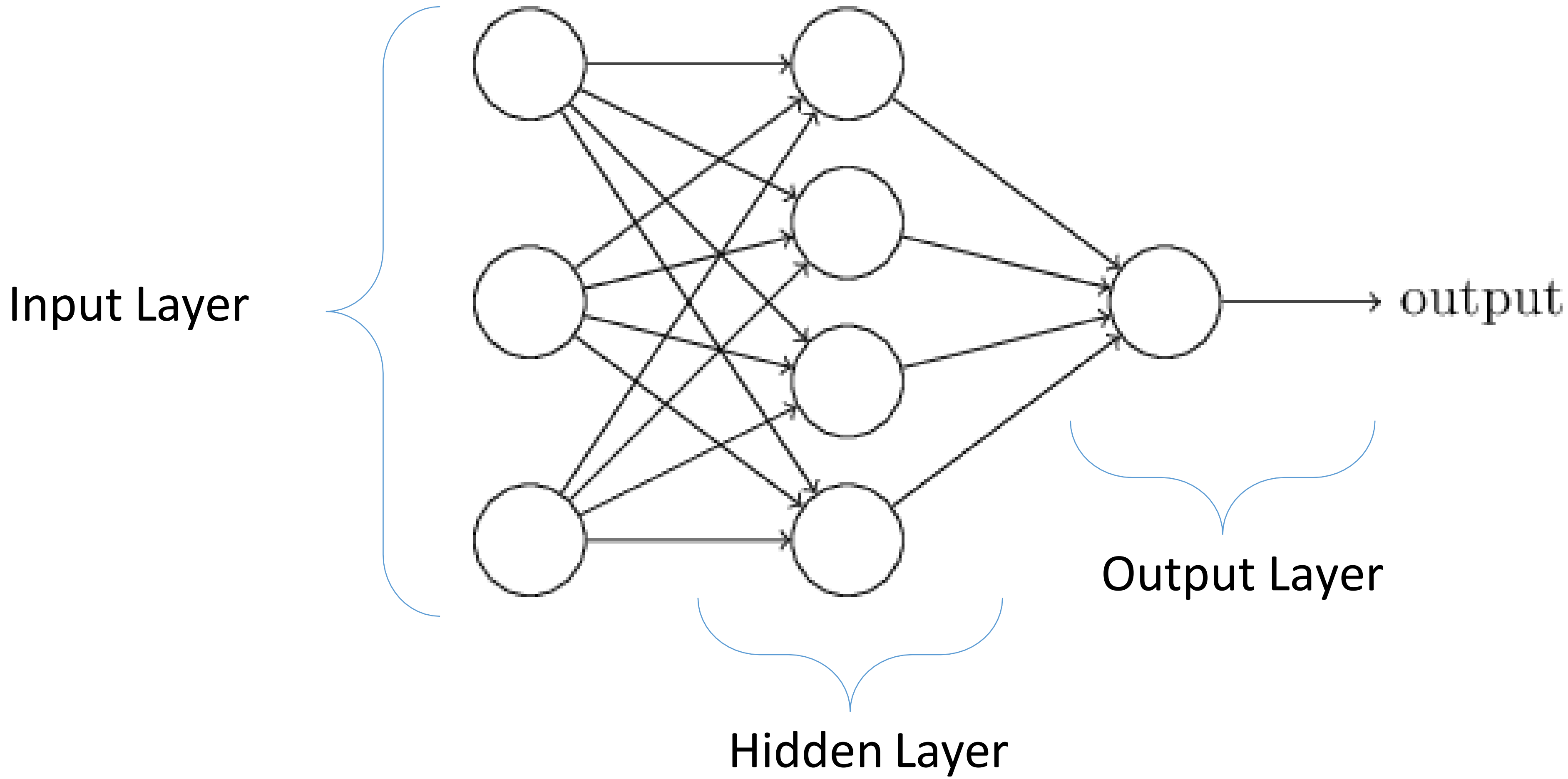


- Now, if we have our NN
  - Perceptron neurons are easy to evaluate
    - 0 or 1; yes, or no;
  - Since sigmoids are continuous, we have to setup some criteria to decide what that output neuron is saying
    - Maybe if a value is >0.5, then that input is a 9.

Neural Network Architecture

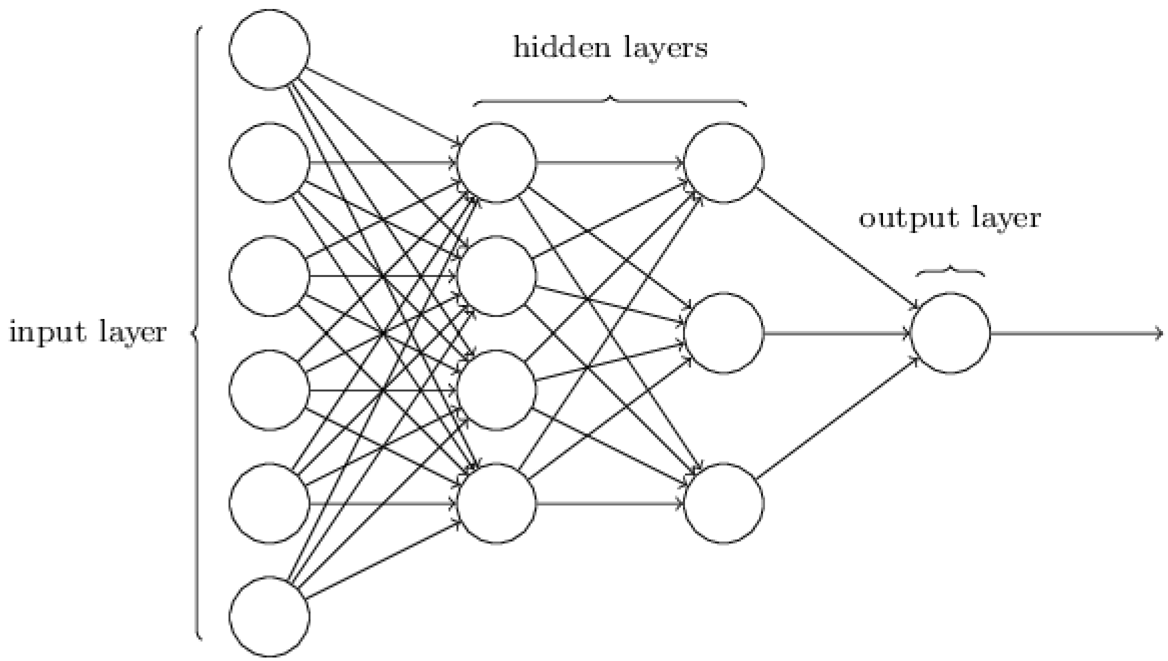
USC University of Southern California

Three Layer Neural Network



Four Layer Neural Network

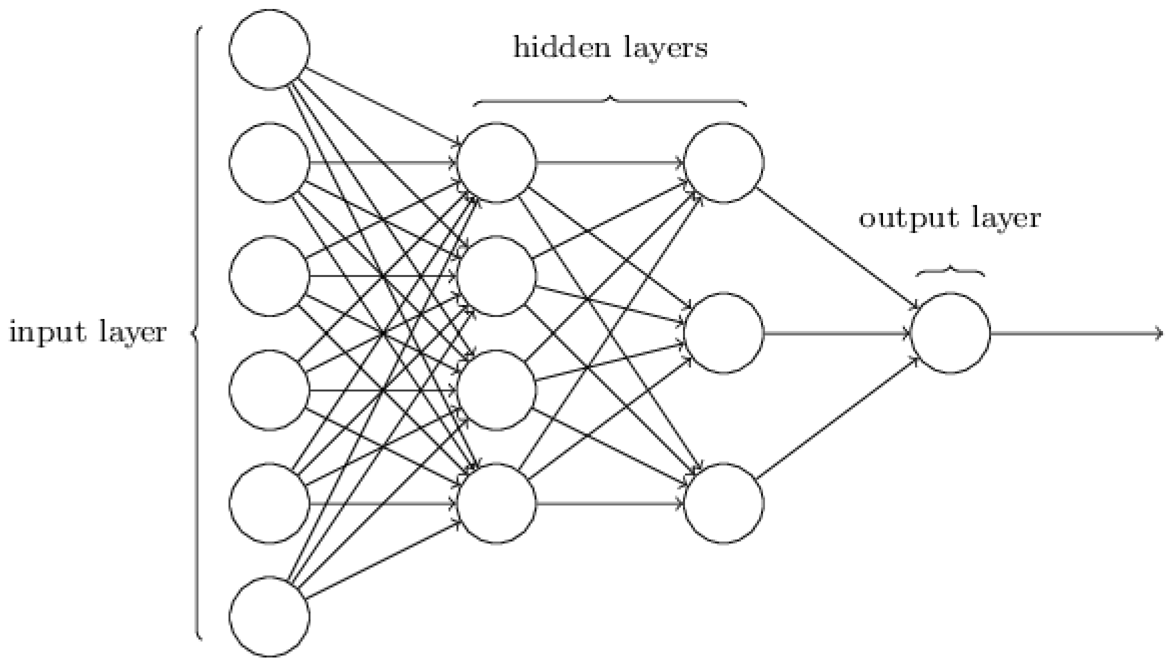
USC
University of Southern California



- This is a multi-layer perceptron (MLP)

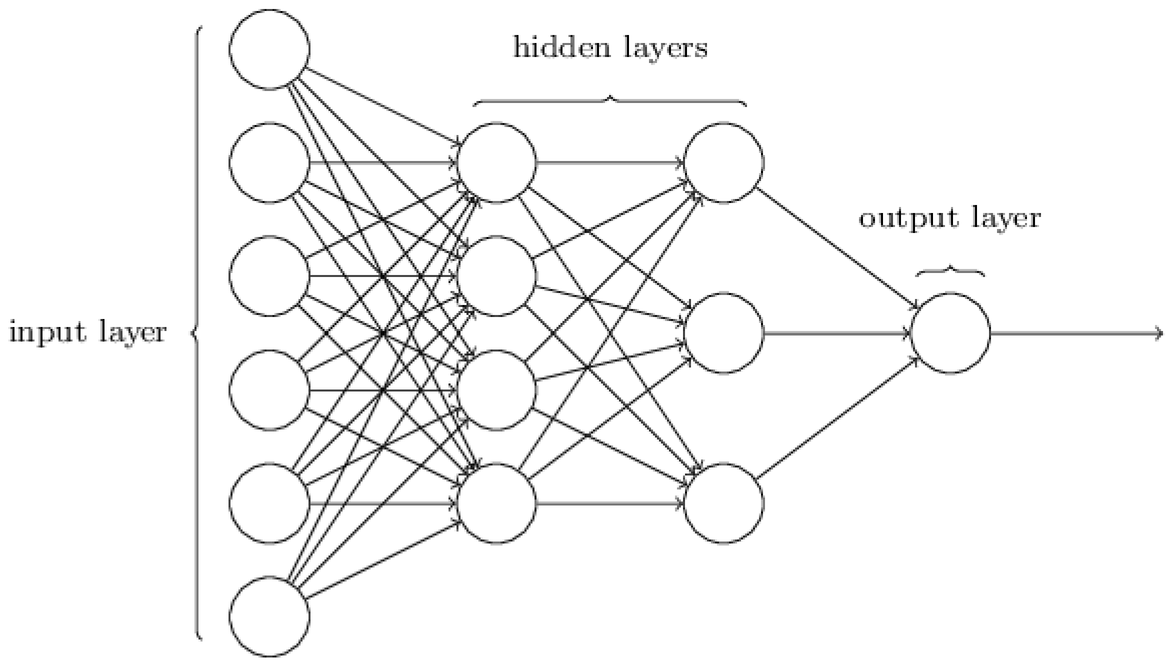
How do we design our Neural Network Topology?

USC University of Southern California



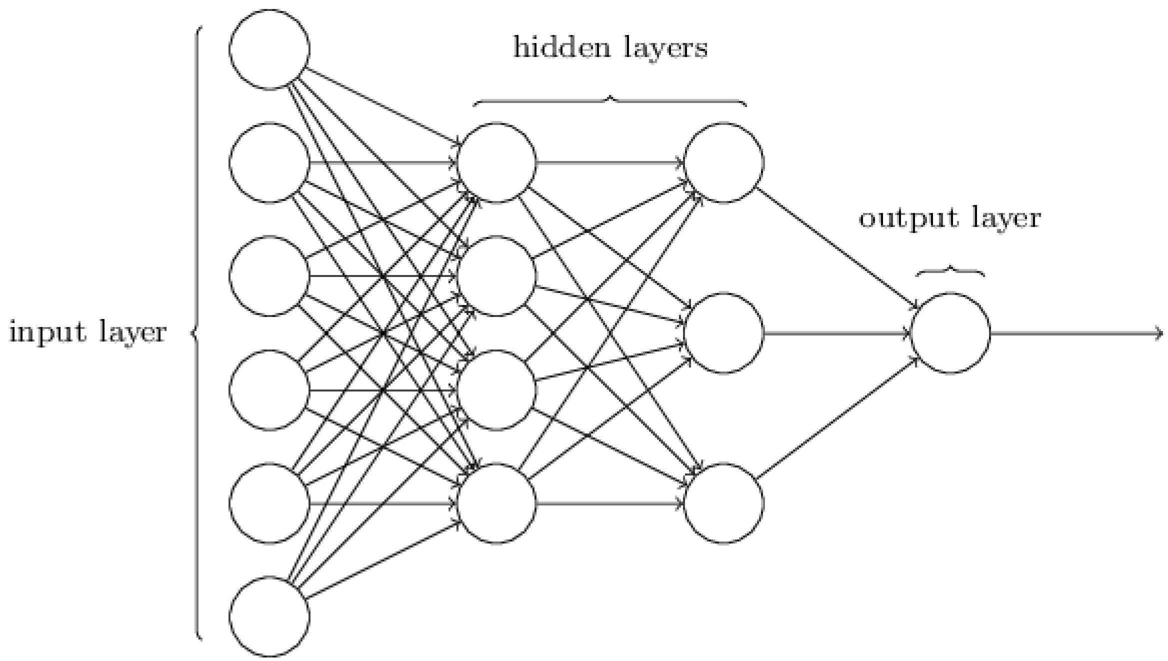
- Inputs
  - Depends on the number and type of inputs
- Outputs
  - If we are trying to do handwritten digit recognition, then we could use 10 outputs (0, 1, … , 9)

How do we design our Neural Network Topology?



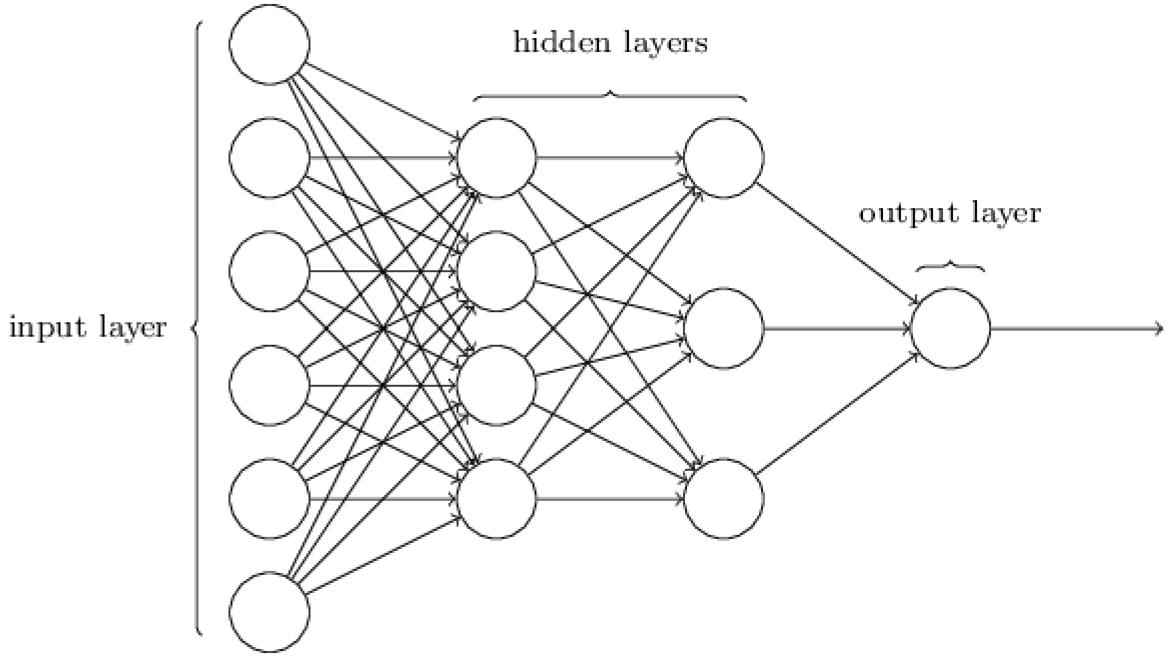
- Input: 64 x 64 pixel greyscale image
  - We could have 4096 input neurons, each with values between 0 and 1, which represent the greyscale intensity of the pixel
  - Here each input represents a pixel

How do we design our Neural Network Topology?



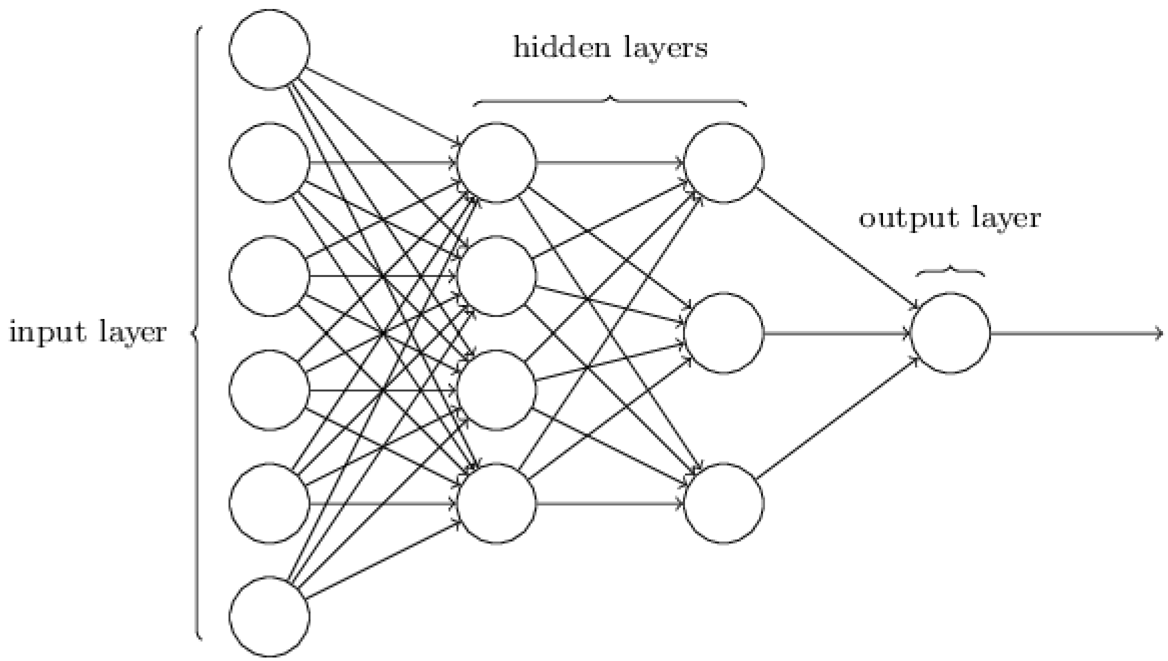
- Output: 0-9
  - With 10 output neurons, we could assign each one a digit, and then depending on the outputs, only 1 of those neurons should output a 'yes'
    - Recall 'yes' is defined as output being greater than some value (e.g. output>0.5)

How do we design our Neural Network Topology?



- Hidden layers?
  - This part is art, not science
  - Driven by heuristics depending on problem

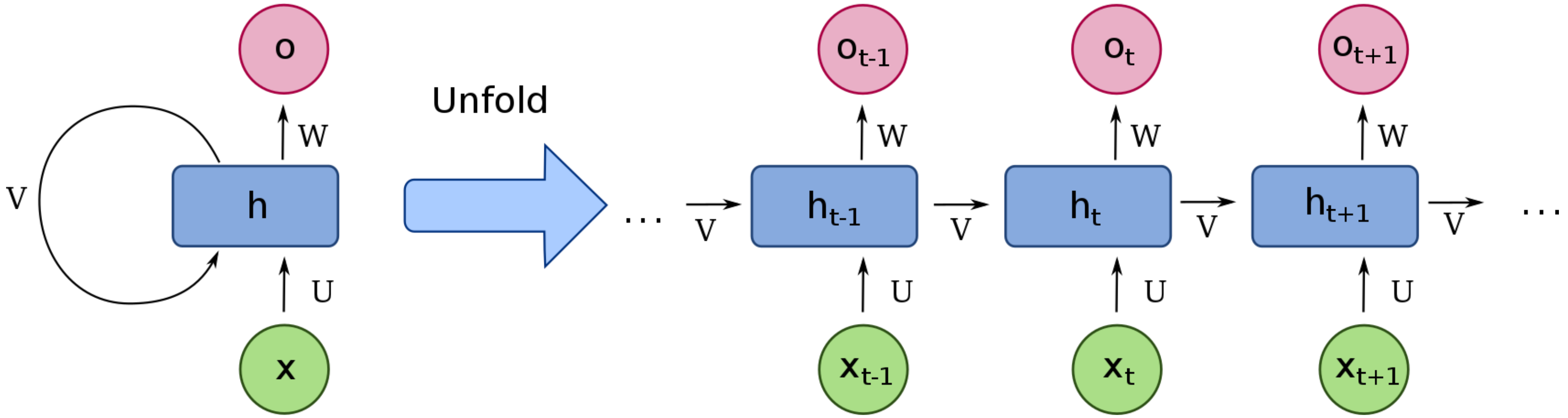
How do we design our Neural Network Topology?



- Note:
  - These are feed-forward neural networks
  - All inputs move forward, never backward, during operation

Aside: Recurrent Neural Networks

USC University of Southern California

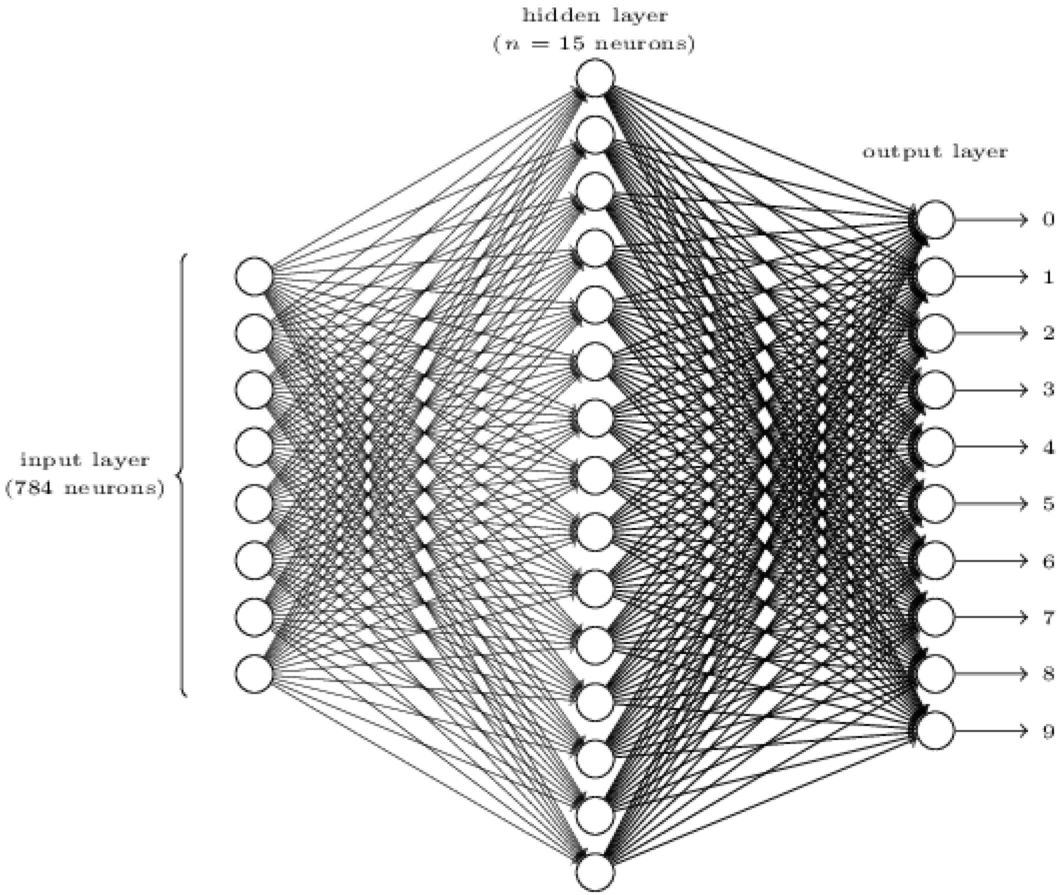


- Note:
  - The output of a neuron actually is an input
  - RNN has memory
  - There is some kind of time dependence, where the behavior can evolve over time
  - This is necessary as the output of a neuron cant be fed back in if it were to be evaluated intantaneously

Getting back to handwriting

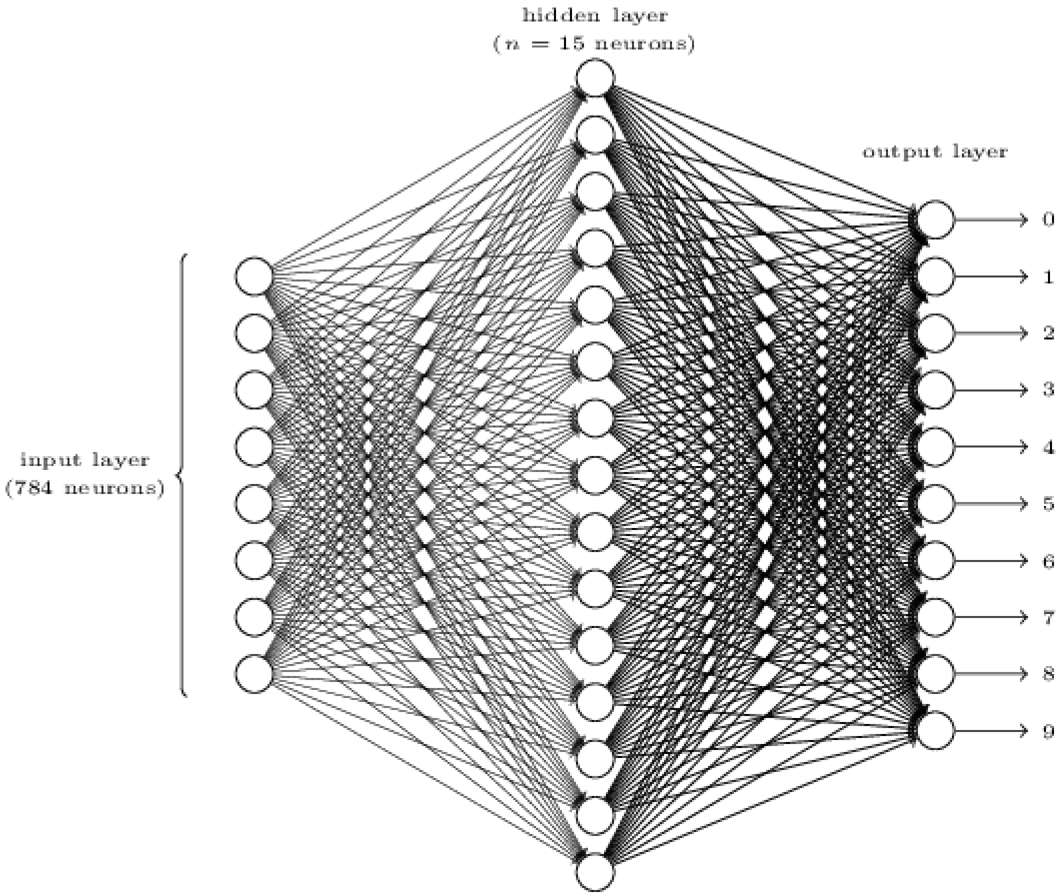
- This number is 6 different images
  - Building an algorithm to automatically recognize and segment is a bit outside the scope of what we cover here and relatively do-able.
  - So we will focus on a single digit
  - Once we have that, segmenting can be done by taking a single image, and segmenting it into many different portions, and then using the handwritten digit classifier to assign scores to the segmentation and use the most highly scored version

An ANN for Handwriting



- What are the inputs?
  - 28 x 28 pixels images of scanned handwritten digits
  - Input layer has 784 neurons
  - Input pixels are greyscale with 0 representing white and 1 representing black
- One hidden layer
  - This only has 15 neurons, need to try different numbers of neurons to figure out how many hidden layers

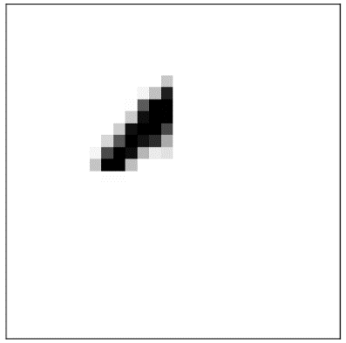
An ANN for Handwriting



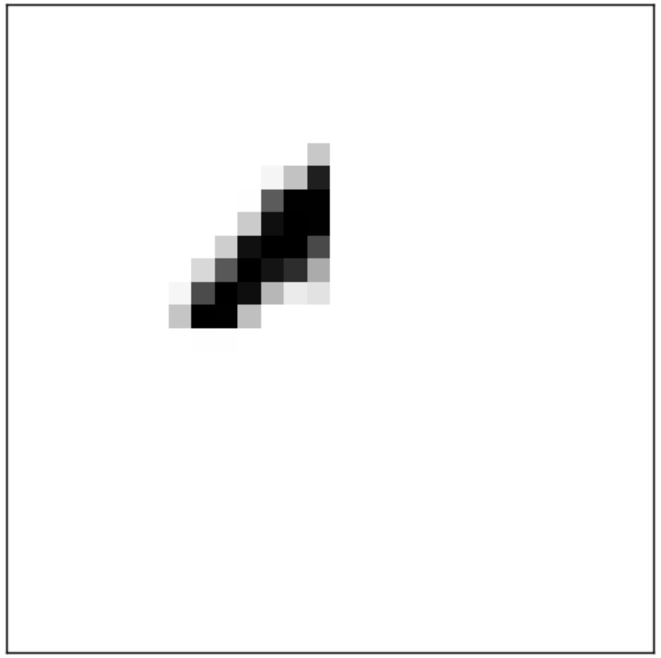
- Output layer
  - 10 neurons, each representing a digit
  - For a given digit, one neuron should be 1, while the others should be 0.
  - In reality, we find the neuron with the highest value, and say that is the one which activated

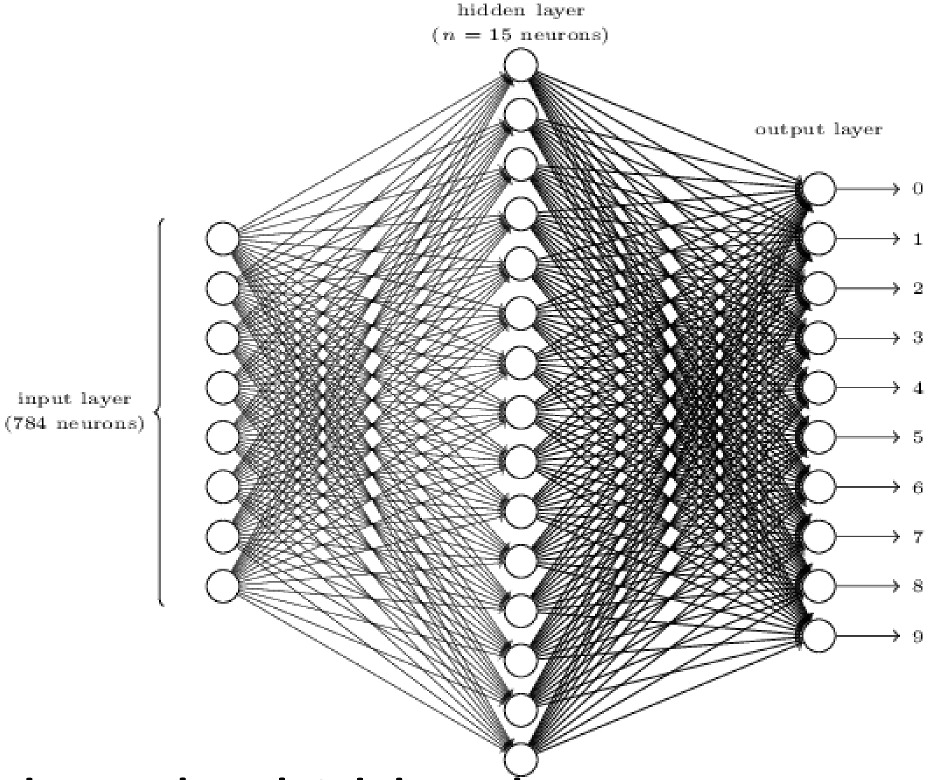
An ANN for Handwriting

- Why do 10 neurons work better than 4?
  - Need to consider how this network is working
    - The ‘0’ output neuron is basically summing up all the inputs from the hidden layer and trying to decide whether the digit is a 0.
    - So then, what are the hidden layer neurons doing?
  - As a heuristic, let’s assume that the first hidden neuron detects whether a certain pattern is present in the image:



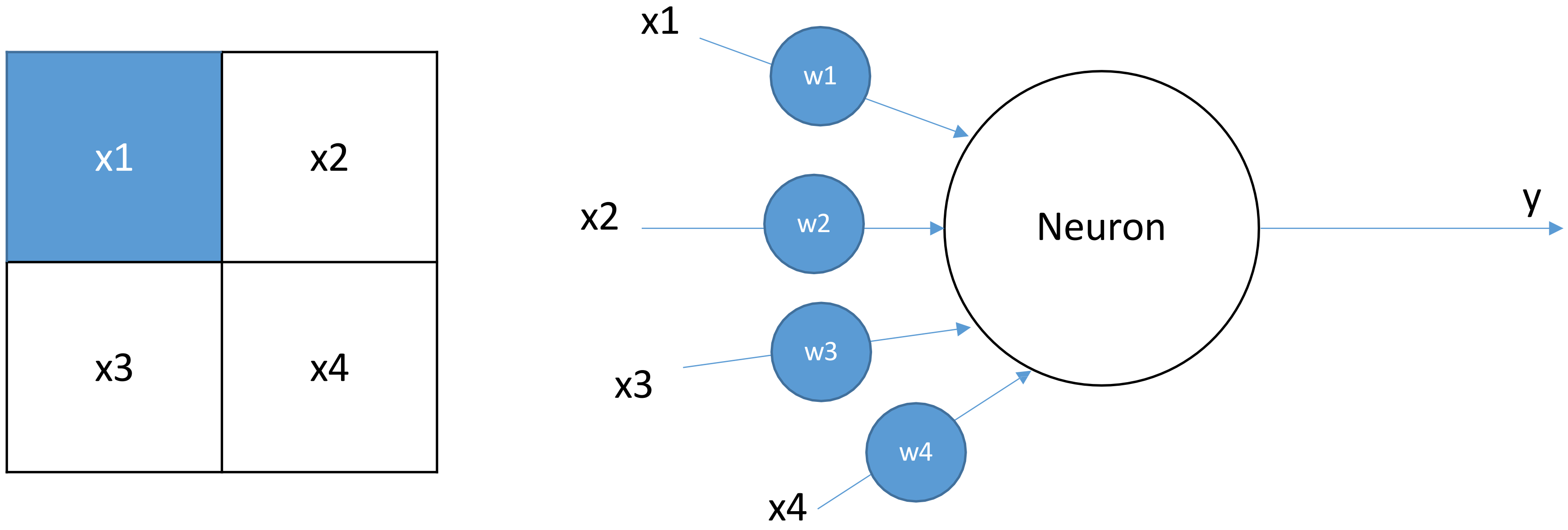
How does the hidden layer neuron find a specific image?





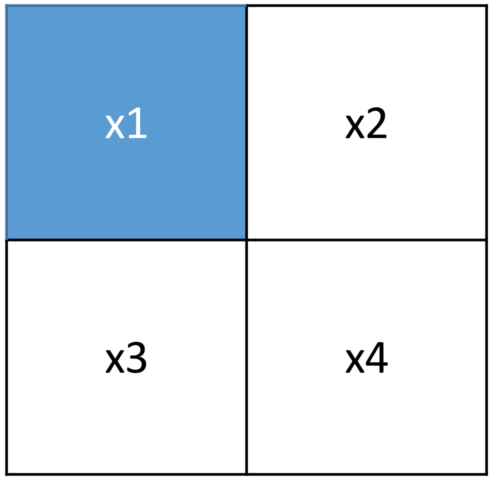
- Recall, every single input is connected to the hidden layer
  - So if I want the first hidden layer neuron to find that pattern, how would I bias it?

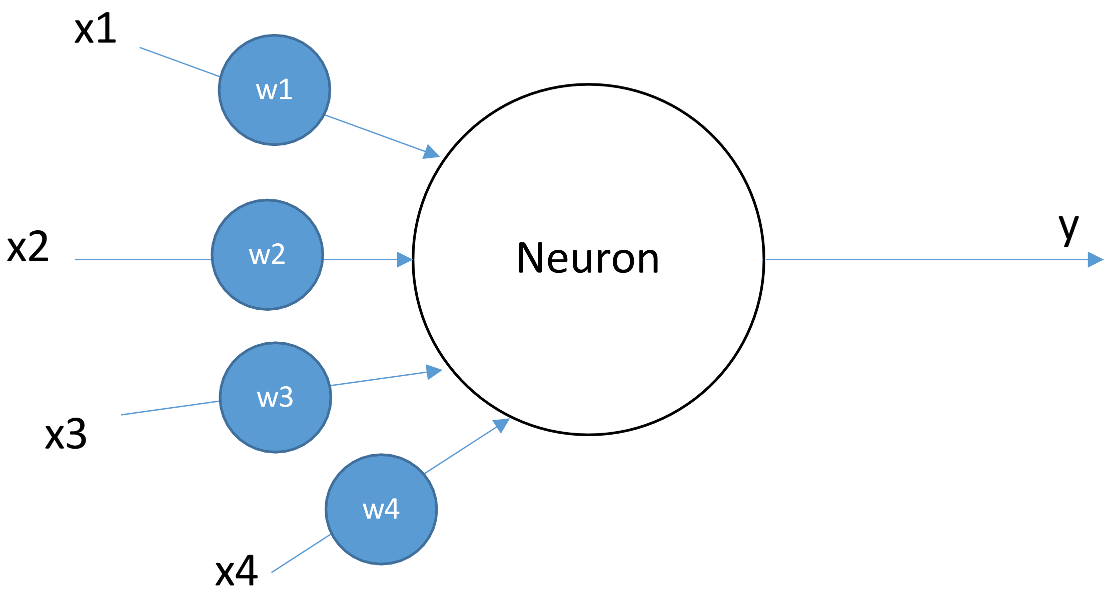
Let’s do a simple example



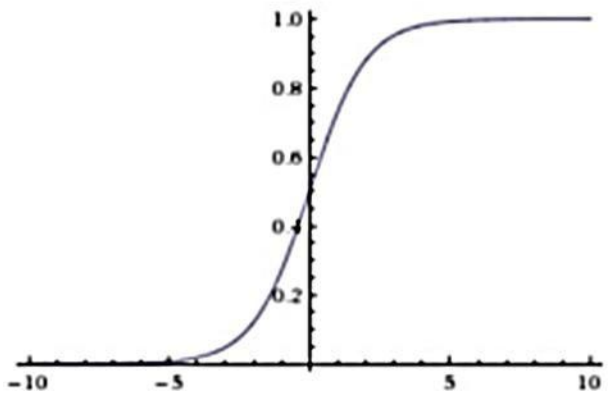
- I want to design a hidden layer neuron which tells me if the top left box is filled
  - So, we have four inputs
  - We want the output y to be high if x1 is dark and x2-4 are light

Let’s do a simple example

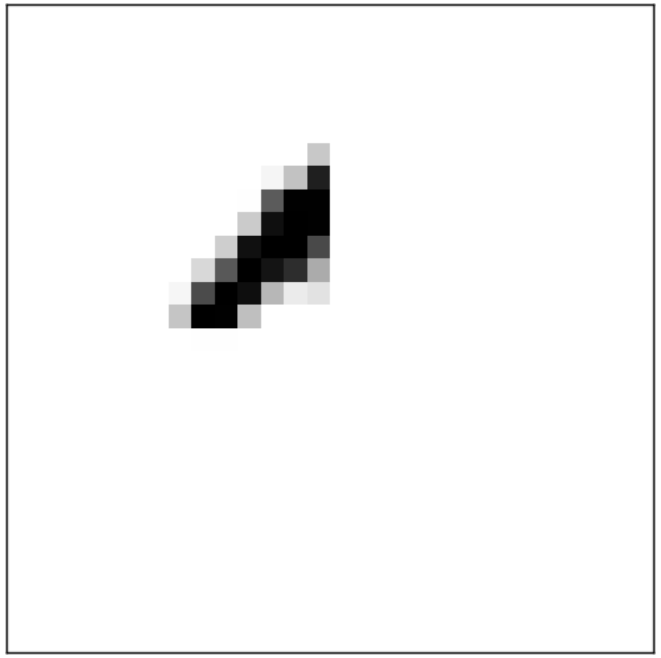


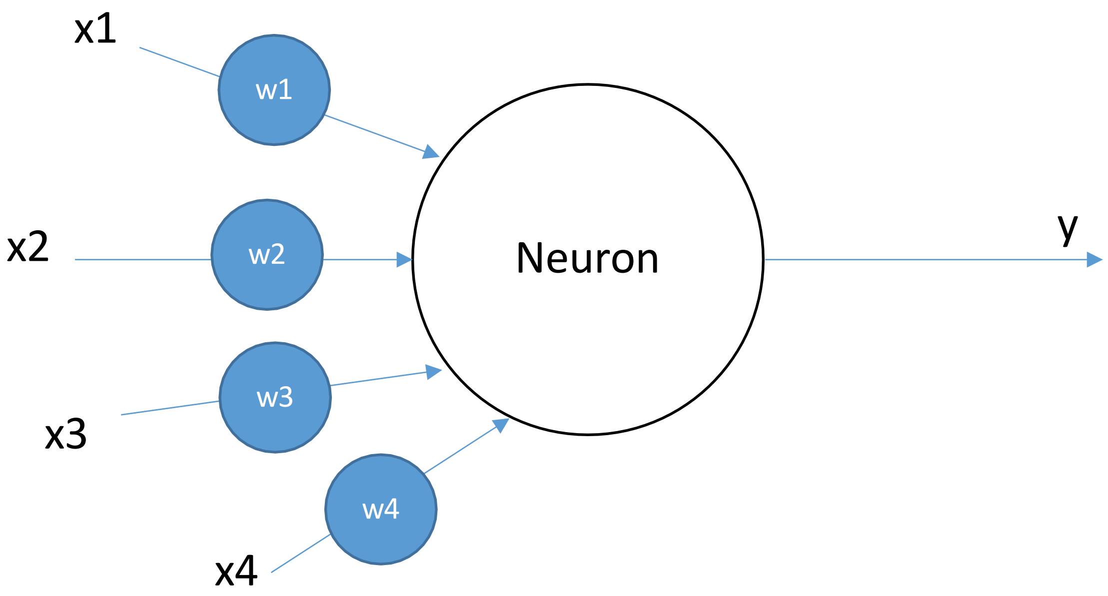


- Recall: $\sigma(x \cdot w+b)=\sigma(z)= \frac{1}{1+e^{-z}}$
  - So, want to weight w1 heavily, while dramatically lowering the weights of x2-x4
  - We want the output y to be high if x1 is dark and x2-4 are light

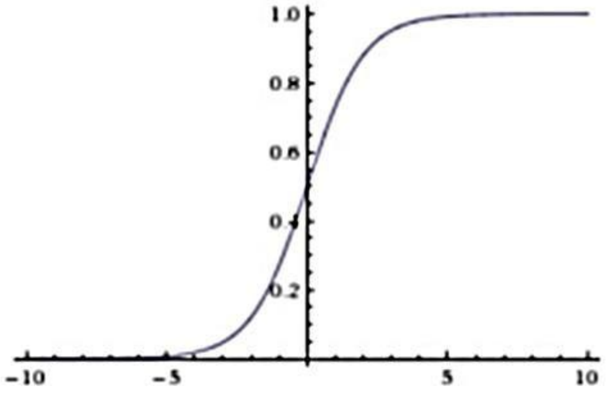


Back to the task at hand

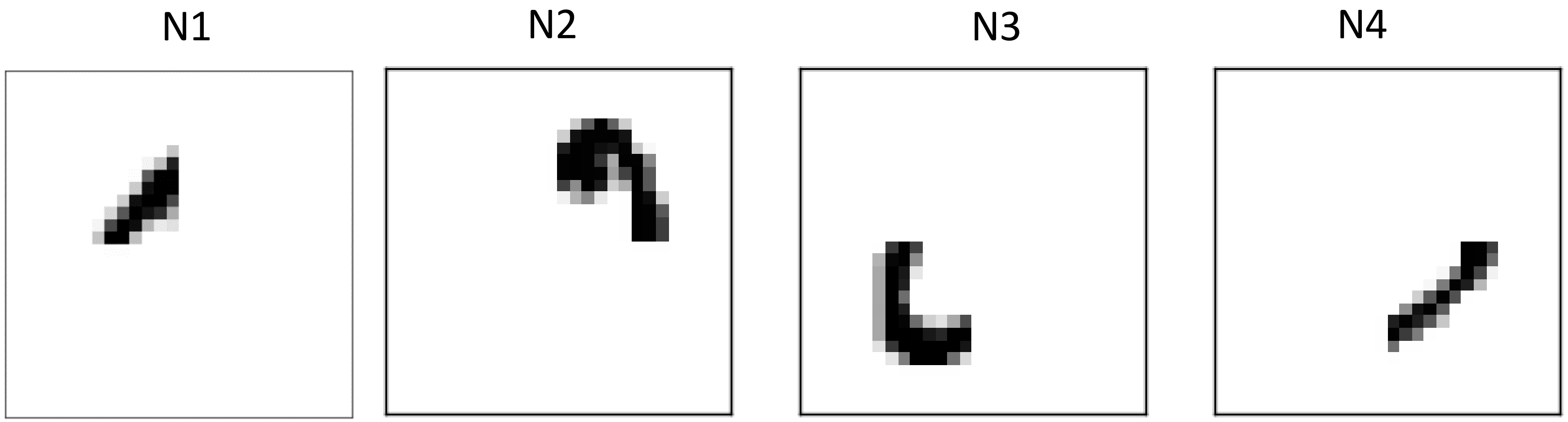


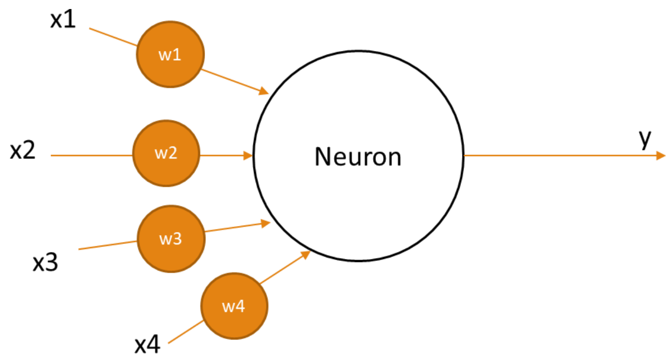


- So, want to weight the pixels which make up this pattern heavily, while reduce the weights of the other pixels

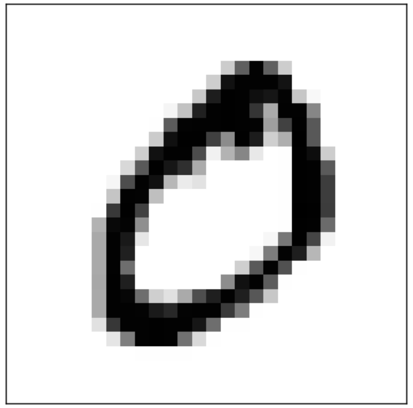


Other hidden layer neurons could then be detecting other images

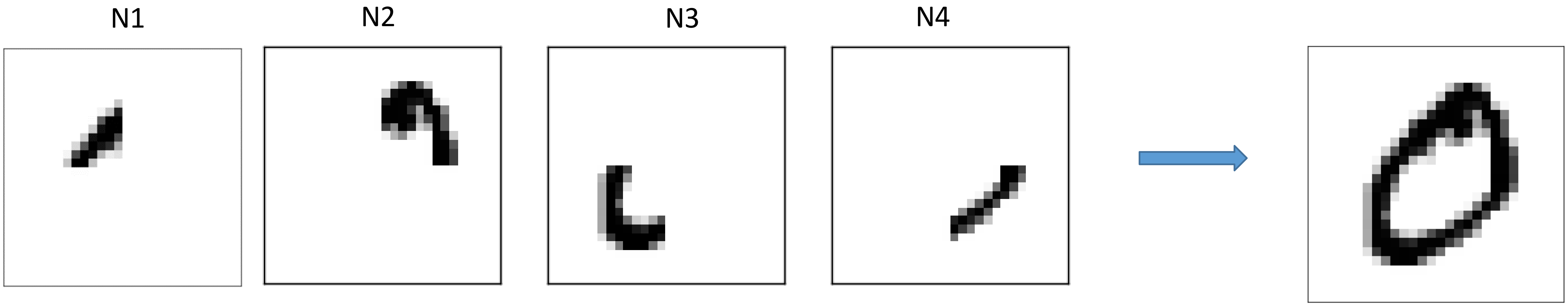




- Then neuron 2 and 3 and 4 could be detecting these other patterns
- Together these 4 images make up a zero



Other hidden layer neurons could then be detecting other images



- If we see that all 4 of these neurons are firing, then we can make the assumption that we are seeing a zero.
  - Obviously this is a toy example, don't get carried away trying to poke holes into it
- Now, we can possibly see why it's better to have 10 outputs over 4
- With the above example, the 0 output neuron could just put the most weight on the outputs from the hidden neurons N1 through N4, and weaken the others
- But, if we had only 4 output neurons, our neural network would have to essentially do 2 calculations
  - First, we would need to figure out what the number was
  - Then we would need to transform that into binary

Learning how Artificial Neural Networks Learn with your Biological Neural network

USC University of Southern California

- We will use the MNIST Data set
  - http://yann.lecun.com/exdb/mnist/
  - Two parts
    - 60,000 Training images (28 x 28 pixels, greyscale)
    - They were scanned samples from >200 people, ½ government employees, ½ high school students
    - 10,000 Test images
    - These test data were collected from different people

How to train your artificial neural network

University of Southern California

- Use x as the annotation for the input data vector, a 784 long string of digits between 0 and 1 representing the greyscale values.
- The output is then a vector y=y(x) with 10 digits.
- As an example:
  - If input image x represents a 1
  - Output vector y(x) = (0,1,0,0,0,0,0,0,0,0)
  - What we need to do is develop an algorithm that let’s us find the weights and biases so output is as close to y(x) given above as possible.

How to train your artificial neural network

- We need to figure out what we want to optimize
  - Obviously, the number of images correctly recognized, duh….
- But what if we used that as the objective function?
  - C(w,b)=Number of correctly classified images
  - Could we easily optimize with this?
  - Not really, this is not a smooth function of w (all the weights in the NN) or b (all the biases in the NN)

How to train your artificial neural network

- So what if we defined another ‘cost function’?
  - C(w,b)=Number of correctly classified images
  - Could we easily optimize with this?
  - Not really, this is not a smooth function of w (all the weights in the NN) or b (all the biases in the NN)