



Neural Networks

Machine Learning Decal

Hosted by Machine Learning at Berkeley

Agenda

Linear and Logistic Regression Recap

Motivation

The Perceptron

The Neural Network

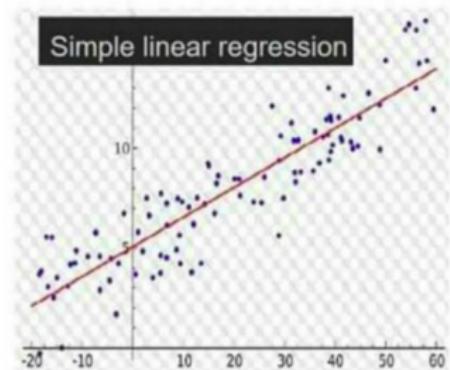
Training

Coding Demo

Questions

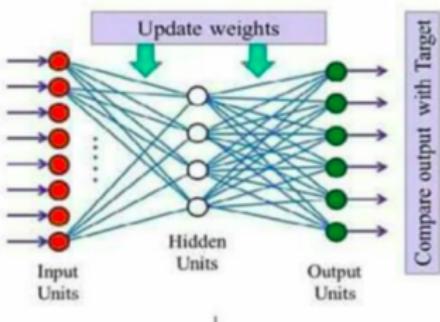
Recap from last lecture

if you
don't love
me at my



then you
don't deserve
me at my

Artificial Neural Network



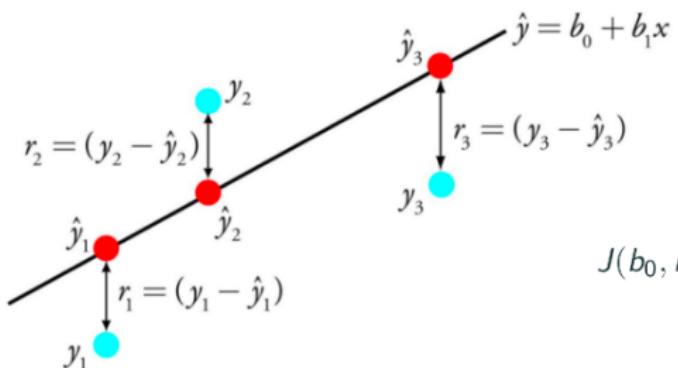
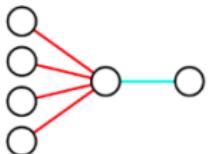
Linear and Logistic Regression Recap

Linear Regression

$$Y = f(X, w_1, b_1)$$

$$f(x, w, b) = x \cdot w + b$$

$$\begin{matrix} X \\ \dots & \dots & \dots & \dots \end{matrix} \cdot \begin{matrix} w \\ \textcolor{red}{\boxed{}} \\ \textcolor{red}{\boxed{}} \\ \textcolor{red}{\boxed{}} \end{matrix} + \begin{matrix} b \\ \textcolor{cyan}{\boxed{}} \end{matrix} = \begin{matrix} Y \\ \dots \end{matrix}$$

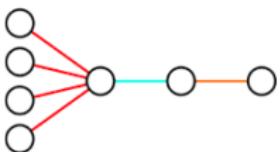


$$J(b_0, b_1) = \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

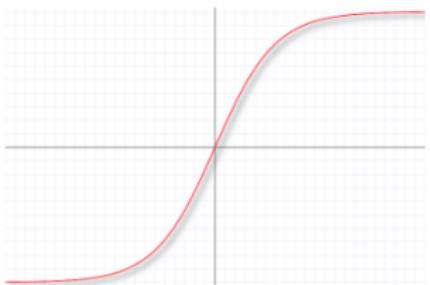
Logistic Regression

$$Z = f(X, \mathbf{w}_1, b_1)$$
$$Y = g(Z)$$

$$\begin{matrix} x \\ \dots & \dots & \dots & \dots \end{matrix} \cdot \begin{matrix} w \\ \dots \\ \dots \\ \dots \end{matrix} + \begin{matrix} b \\ \dots \end{matrix} = \begin{matrix} z \\ \dots \end{matrix}$$
$$z \xrightarrow{g} Y$$

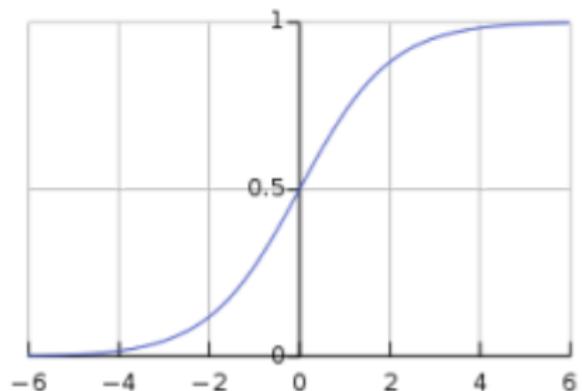


$$g(x) = \frac{1}{1 + e^{-x}}$$

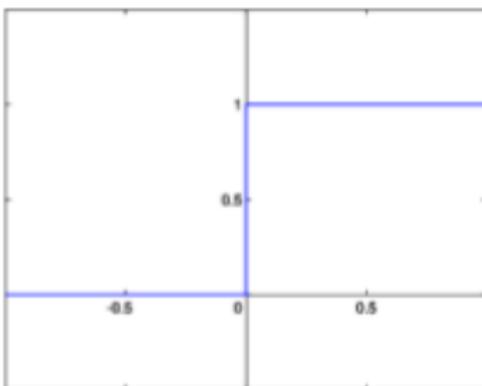


$$J(b) = - \sum_{i=1}^m \left(y^{(i)} \cdot \ln z^{(i)} + (1 - y^{(i)}) \cdot \ln (1 - z^{(i)}) \right)$$

Activation vs Prediction Functions



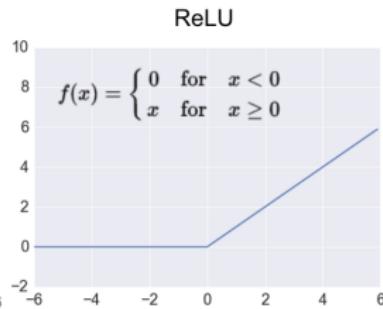
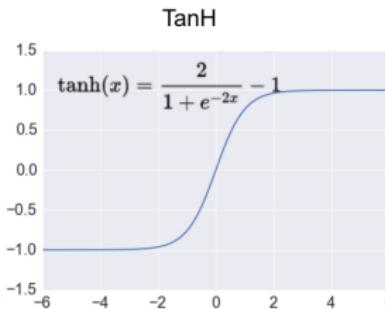
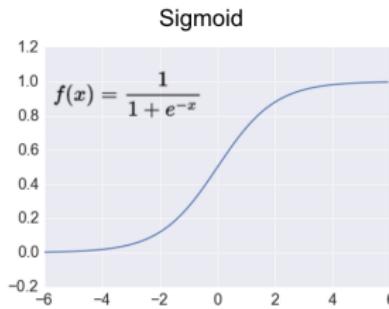
VS



General Activation Functions

For gradient descent to work, we need activation functions that are

- Continuous
- Monotonically increasing
- Generally Differentiable

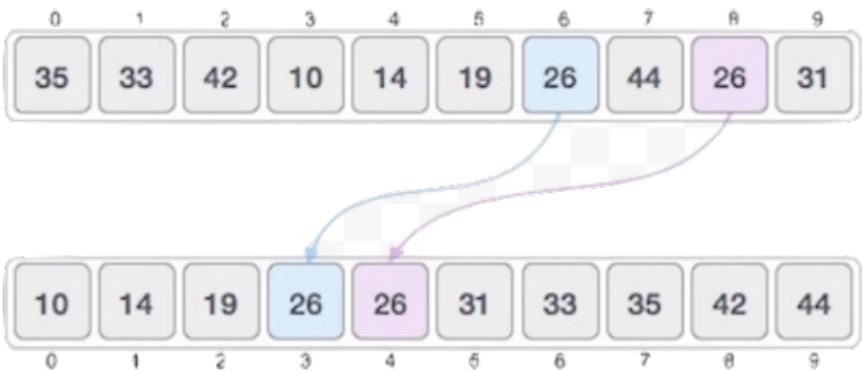


Motivation

Computers have always been great for...

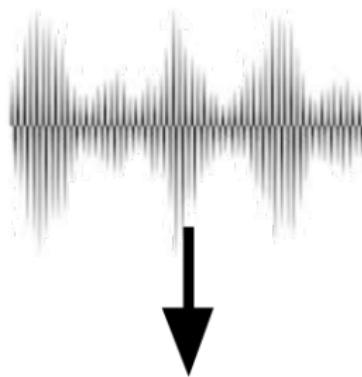


A	B	C	A'	B'	C'	$A'BC'$	$A'BC$	$AB'C$	ABC	$M=A'BC'+A'BC+AB'C+ABC$
0	0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0
0	1	0	1	0	1	1	0	0	0	1
0	1	1	1	0	0	0	1	0	0	1
1	0	0	0	1	1	0	0	0	0	0
1	0	1	0	1	0	0	0	1	0	1
1	1	0	0	0	1	0	0	0	0	0
1	1	1	0	0	0	0	0	0	1	1

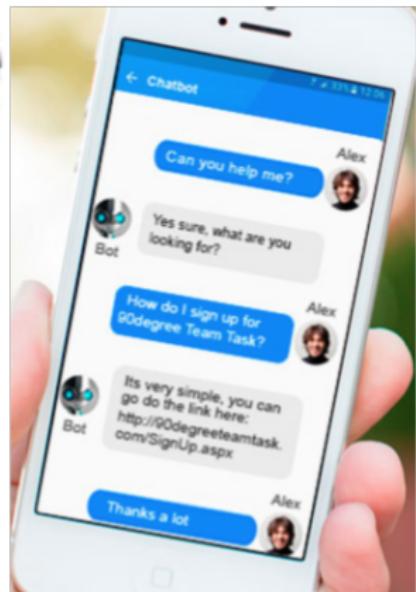


But have historically struggled with “simple intuition”

What animal is this?



How are you?



Why?



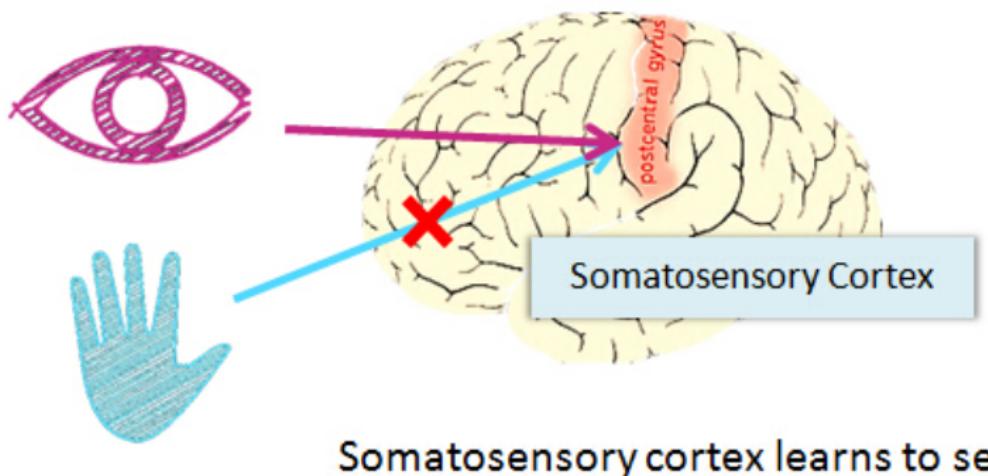
What I see



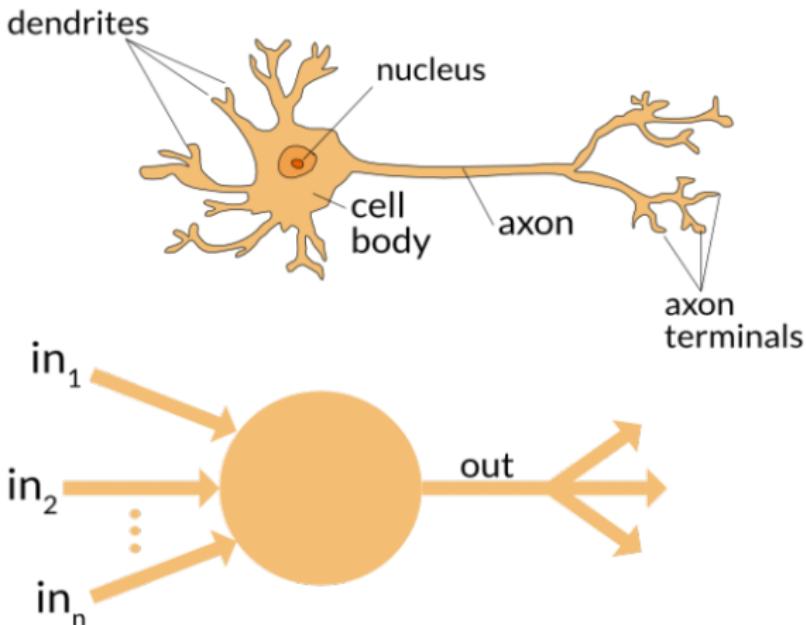
What a computer sees

```
08 02 22 97 38 15 00 40 00 75 06 05 07 78 52 12 50 77 91 08  
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00  
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65  
52 70 95 23 04 60 11 42 69 24 66 56 01 32 56 71 37 02 34 91  
22 31 16 71 51 67 63 89 41 92 36 54 22 40 28 66 33 13 80  
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50  
32 98 81 28 64 23 47 10 26 38 40 67 59 54 70 66 18 38 64 70  
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21  
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72  
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95  
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92  
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57  
86 56 00 48 35 71 59 07 05 44 44 37 44 60 21 58 51 54 17 58  
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40  
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66  
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69  
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36  
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16  
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54  
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

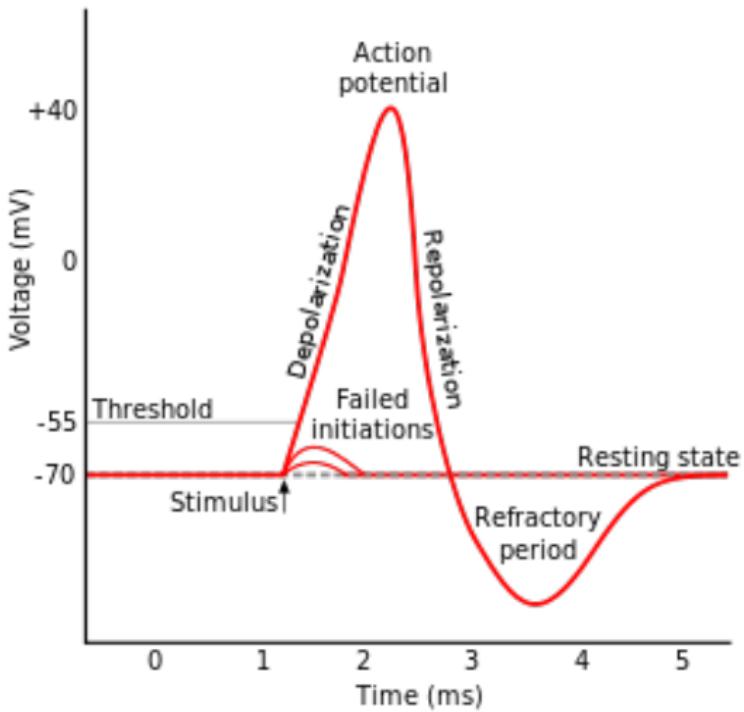
The "One Learning Algorithm" Hypothesis



Biological inspiration

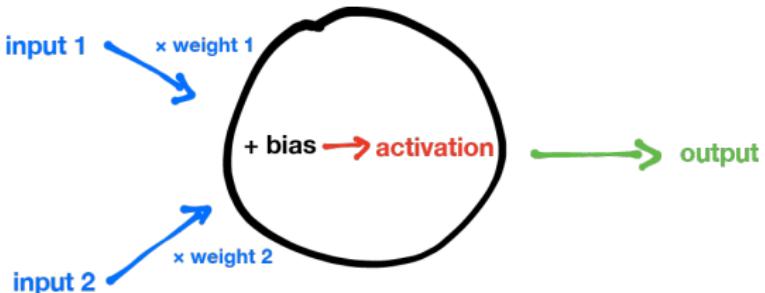


Biological Inspiration



The Perceptron

The Perceptron

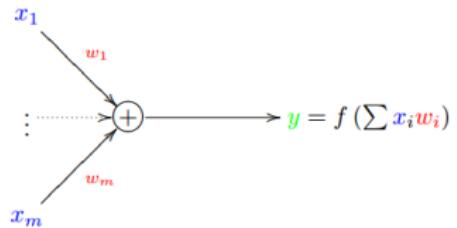


- Inputs to the neuron are multiplied by weights (the parameters) and then summed
- A bias term (another parameter) is added to the sum
- An activation is then applied (for instance, $\tanh(x)$ or $\text{ReLU}(x)$)

The Perceptron



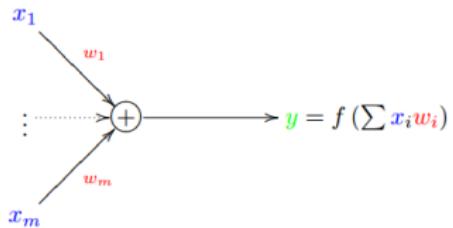
McCulloch and Pitts (1943) proposed the ‘integrate and fire’ model:



The Perceptron

Let $f(x) = x^2$, $x = (1, 4, -2)$, $w = (-2, 3, 2)$. What is the output?

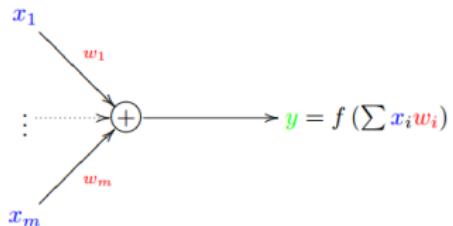
McCulloch and Pitts (1943) proposed the ‘integrate and fire’ model:



The Perceptron



McCulloch and Pitts (1943) proposed the ‘integrate and fire’ model:

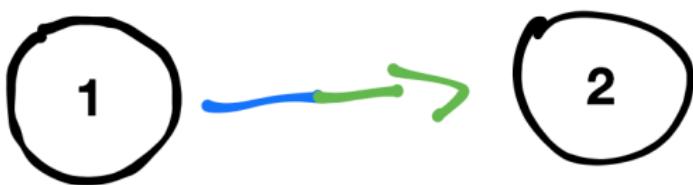


$$\sum_i x_i w_i = (1)(-2) + (4)(3) + (-2)(2) = 6$$

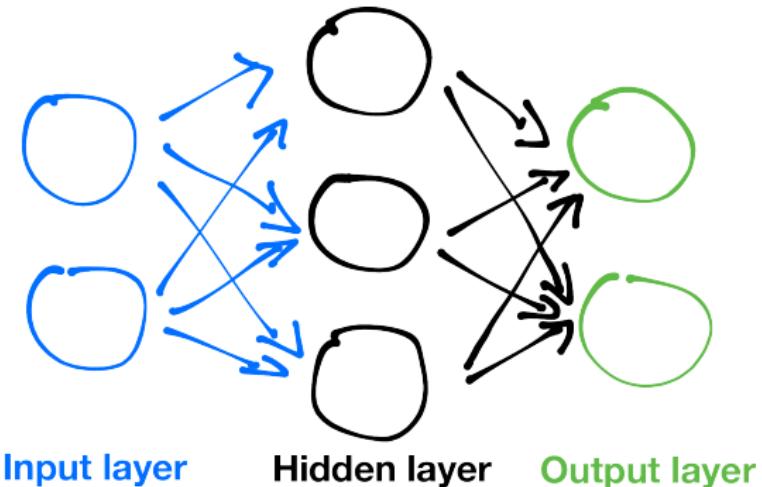
$$f\left(\sum_i x_i w_i\right) = f(6) = 36$$

The Neural Network

Linking it Together

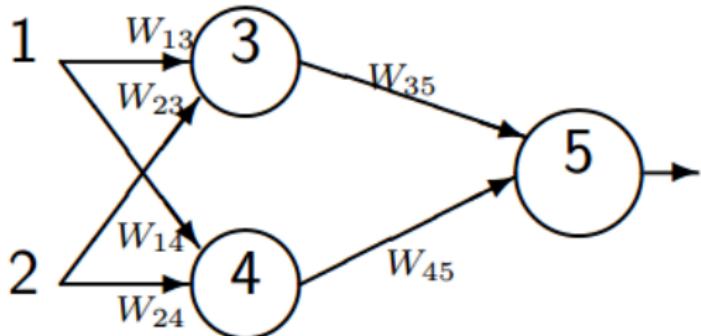


- The output of a neuron becomes the input for another neuron



- Layered Architecture
- Three types of layers:
 - **Input Layer** - Data is passed into these neurons
 - **Hidden Layer** - These neurons are "hidden from view"
 - **Output Layer** - These neurons output the result of the network

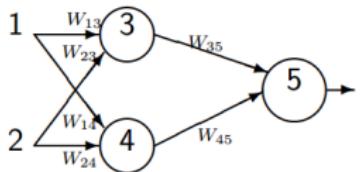
Feedforward Example



$w_{13} = 2$	$w_{35} = 2$
$w_{23} = -3$	
$w_{14} = 1$	$w_{45} = -1$
$w_{24} = 4$	

$$f(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Feedforward



$w_{13} = 2$	$w_{35} = 2$
$w_{23} = -3$	
$w_{14} = 1$	$w_{45} = -1$
$w_{24} = 4$	

$$f(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$w_{13}(1) + w_{23}(2) = (2)(1) + (-3)(2) = -4$$

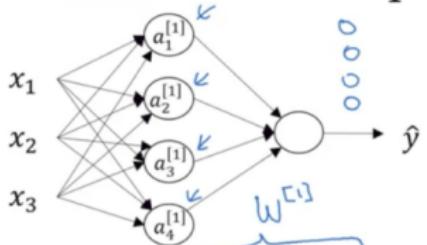
$$w_{14}(1) + w_{24}(2) = (1)(1) + (4)(2) = 9$$

$$z_3 = 0$$

$$z_4 = 1$$

$$z_5 = f(w_{35}(0) + w_{45}(1)) = f((-1)(1)) = 0$$

Neural Network Representation



$$z^{[1]} = \begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + b^{[1]}$$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

Diagram illustrating the computation of the hidden layer activations $a^{[1]}$ from the input x through the weight matrix $w^{[1]}$ and bias vector $b^{[1]}$.

The diagram shows the forward pass:

$$\begin{aligned} z_1^{[1]} &= w_1^{[1]T} x + b_1^{[1]} \\ z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]} \\ z_3^{[1]} &= w_3^{[1]T} x + b_3^{[1]} \\ z_4^{[1]} &= w_4^{[1]T} x + b_4^{[1]} \end{aligned}$$

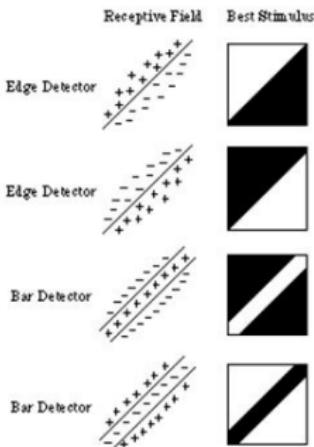
These intermediate values are then passed through the activation function σ :

$$\begin{aligned} a_1^{[1]} &= \sigma(z_1^{[1]}) \\ a_2^{[1]} &= \sigma(z_2^{[1]}) \\ a_3^{[1]} &= \sigma(z_3^{[1]}) \\ a_4^{[1]} &= \sigma(z_4^{[1]}) \end{aligned}$$

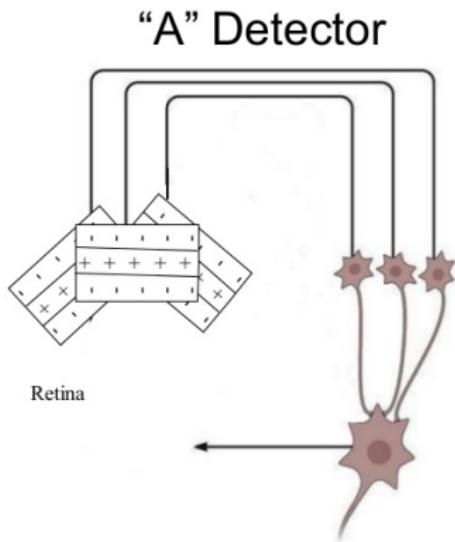
Final output \hat{y} is shown as a vertical vector of zeros.

Why having more perceptrons works?

- Hubel and Weisel
 - <https://www.youtube.com/watch?v=IOHayh06LJ4>
- Biological neurons in the visual cortex are edge detectors



- By combining the output of edge detecting neurons, we can make more complex detectors





Somewhere out there, there's a perfect function that tells you exactly what to do for some sensor input (**Platonist view**)

What does a neural network do?



- Neural networks are function approximators
- What is a function?
 - Anything that maps an input to a single output

$$f : X \rightarrow Y$$

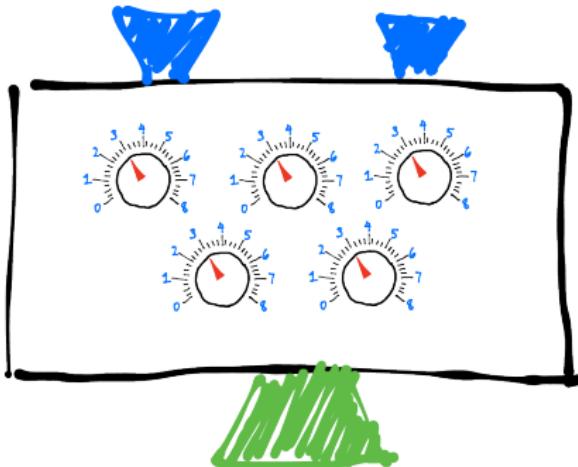
- Neural Networks are capable of learning a decision making process that fits given data well

- Take for example a self driving car

$$f : X \rightarrow Y$$

- f is a function that maps sensor readings to a driver's action
- X is the set of all possible combinations of sensor readings
- Y is the set of all possible outputs to a car

Neural Networks as a Black Box



- Neural networks approximate functions by adjusting **parameters**
 - Modern networks often times have hundreds of millions of parameters
 - We train neural networks to find parameters that approximate our function as closely as possible

Training

How do Neural Networks learn?



ML@B

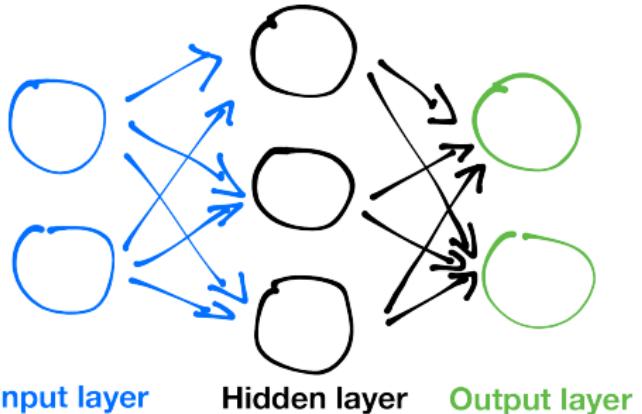
- So this architecture can (theoretically) approximate any function.
- But how do we actually find the correct parameters?
- Gradient Descent!

- We can define a cost function

$$C(x, \text{parameters}) = \frac{1}{2}(y - \hat{f}(x))^2$$

- x is our input training example
- y is our training example label
- $\hat{f}(x)$ is the output of our network (a function of the parameters and x)
- Gradient descent allows us to find a local minimum of C given the derivatives of C with respect to the parameters

Backpropagation



Neural network:

$$h_{\Theta}(x) \in \mathbb{R}^K \quad (h_{\Theta}(x))_i = i^{th} \text{ output}$$

$$\begin{aligned} J(\Theta) = & -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_{\Theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_{\Theta}(x^{(i)}))_k) \right] \\ & + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2 \end{aligned}$$

Coding Demo

MNIST Dataset



label = 5



label = 0



label = 4



label = 1



label = 9



label = 2



label = 1



label = 3



label = 1



label = 4



label = 3



label = 5



label = 3



label = 6



label = 1



label = 7



label = 2



label = 8



label = 6



label = 9



Can we measure loss using only predicted digit?

Suppose we applied an activation function to our final output neuron mapping its value to one of 0 through 9.
Then we use MSE between output and label for loss.

label = 5



Prediction:

0	1	2	3	4	5	6	7	8	9
25	16	9	4	1	0	1	4	9	16

Loss:

Another idea

Activation Functions have to be continuous!

Maybe we could map to a range of real numbers.

Again we use MSE between output and label for loss.

Output:	0.79	1.83	2.54	2.99	3.45	4.89	5.32	6.22	7.13	8.72
Loss:	17.72	10.04	6.051	4.040	2.402	0.012	0.102	1.488	4.536	13.83

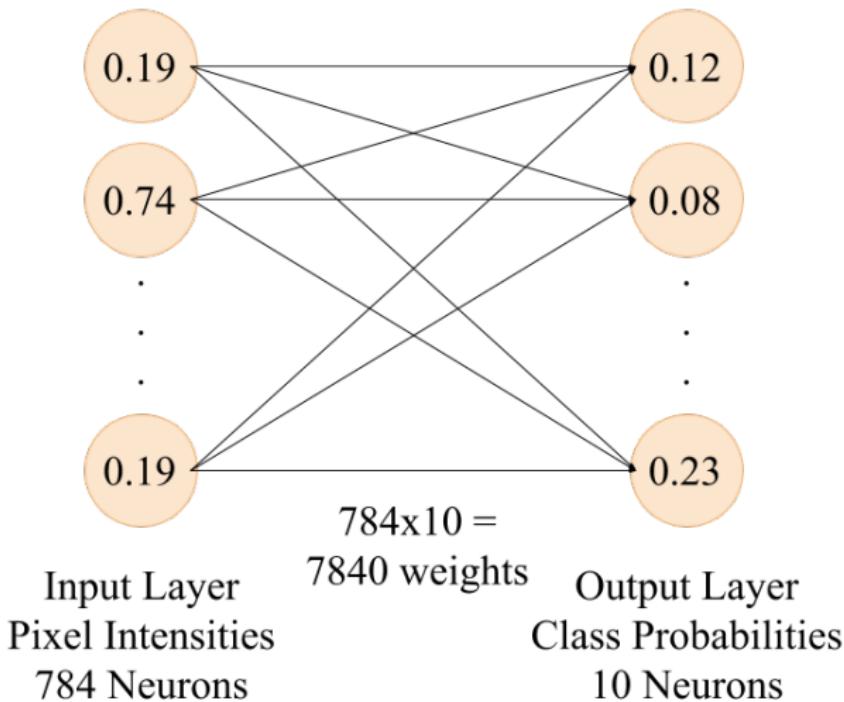
(Note our actual prediction would be whichever digit is closest to our output)

- Although the outputs we want to predict correspond to numbers, they are ultimately **unordered classes**.
- Having the model directly predict numbers imposes a certain notion of ordering and similarity which is inherently not true.
- Predicting what type digit should be similar to predicting what type of animal.

- Instead of outputting one value, we output 10 values one for each class.
- Each number represents the probability the model is assigning for each digit.
- Labels are converted to one hot vectors
- Actual prediction is class assigned the highest probability.

Class:	0	1	2	3	4	5	6	7	8	9
Output:	0.024	0.064	0.174	0.003	0.064	0.473	0.174	0.001	0	0.024
Label:	0	0	0	0	0	1	0	0	0	0

Most Basic Neural Network



Softmax Activation and Cross Entropy Loss

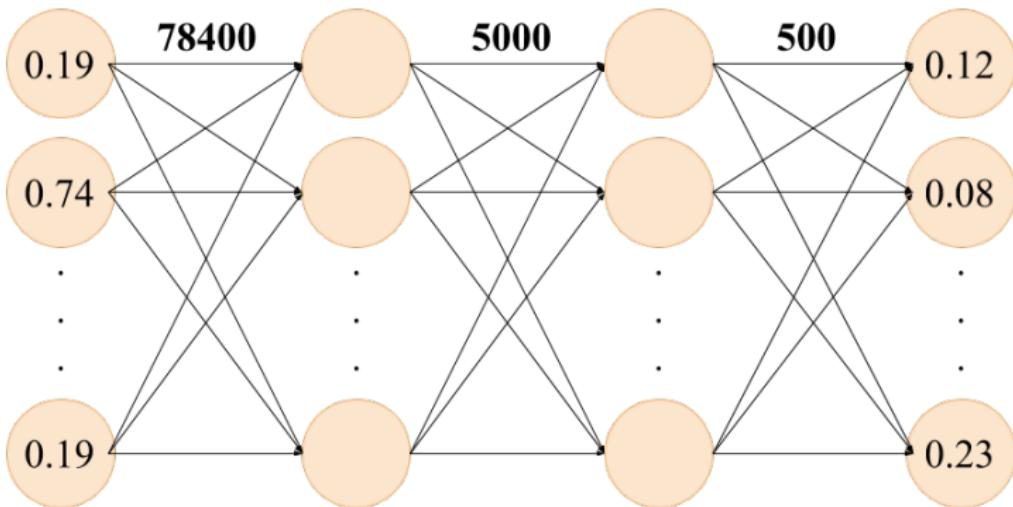


	0	1	2	3	4	5	6	7	8	9
SM Input:	4.1	2.2	2.9	-1.8	2.5	4.2	3.3	-2.5	-1.5	1.1
Output:	0.326	0.044	0.12	0.002	0.044	0.326	0.12	0.001	0.001	0.016
Label:	0	0	0	0	0	1	0	0	0	0
-ln(out):	1.121	3.121	2.121	6.121	3.121	1.121	2.121	7.121	7.121	4.121
-ln(1-out):	0.394	0.045	0.128	0.002	0.045	0.394	0.128	0.001	0.001	0.016
LOSS:	0.394	0.045	0.128	0.002	0.045	1.121	0.128	0.001	0.001	0.016

Total Loss: 1.822

$$Loss = -(y * \ln(\hat{y}) + (1 - y) * \ln(1 - \hat{y}))$$

Adding Hidden Layers to a FeedForward NN



Input Layer	Hidden Layer	Hidden Layer	Output Layer
Pixel Intensities	ReLU Activation	ReLU Activation	Class Probabilities
784 Neurons	100 Neurons	50 Neurons	10 Neurons

Questions
