Neural Networks

Machine Learning Decal

Hosted by Machine Learning at Berkeley

Agenda

Motivation

Linear and Logistic Regression Recap

The Perceptron

The Neural Network

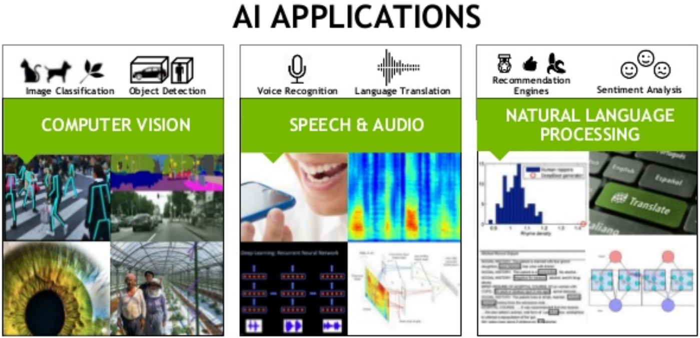
Learning

Coding Demo

Questions

Motivation

'Intuition' is hard to program



Different modes of interpretation

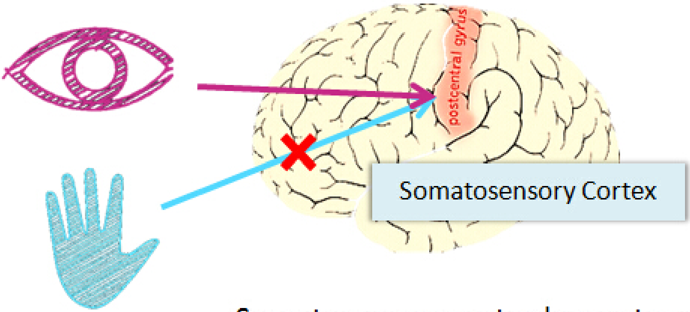
What I see

What a computer sees

08	02	22	97	38	15	00	40	00	75	04	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	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	67	63	89	41	92	36	54	22	40	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	67	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	89	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

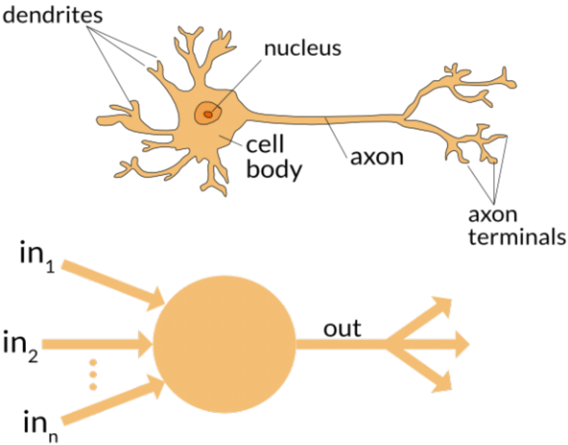
Manual feature extraction is difficult and lacks generalizability.

The "One Learning Algorithm" Hypothesis

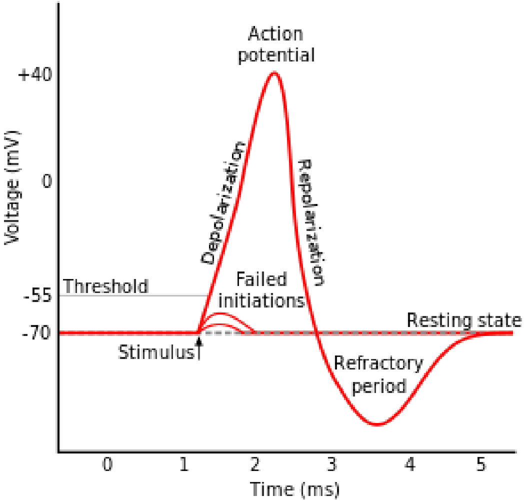


Somatosensory cortex learns to see

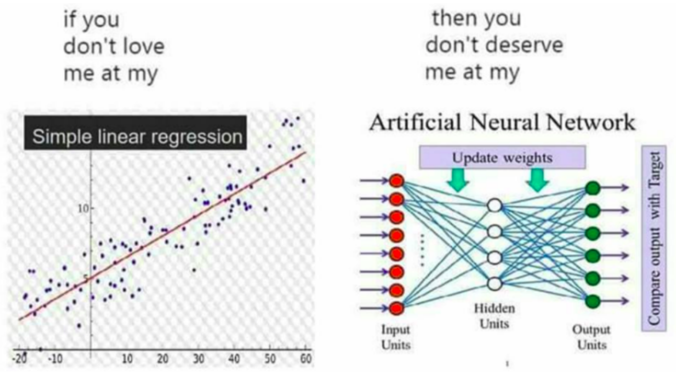
Biological inspiration



Biological Inspiration



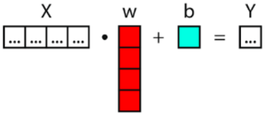
Recap from last lecture

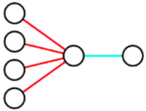


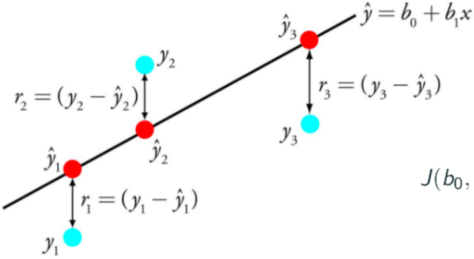
Linear and Logistic Regression Recap

Linear Regression

$$Y = f(X, w_1, b_1)$$
$$f(x, w, b) = x \cdot w + b$$



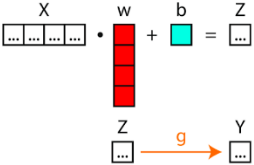


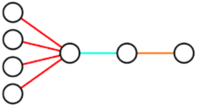


$$J(b_0, b_1) = \frac{1}{2m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2$$

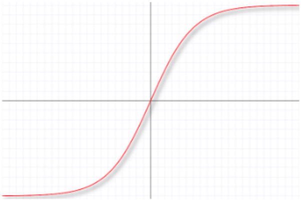
Logistic Regression

$$Z = f(X, {\color{red}w_1}, {\color{purple}b_1})$$
$$Y = g(Z)$$



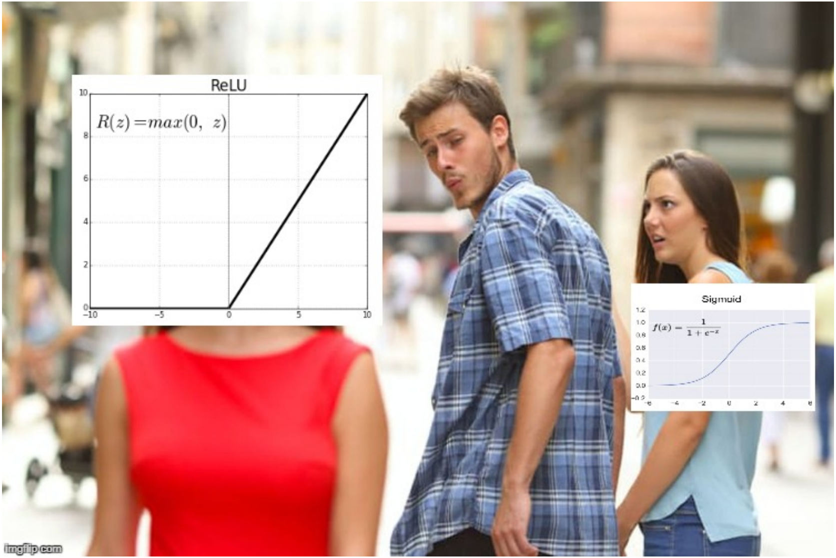


$$g(x) = \frac{1}{1+e^{-x}}$$

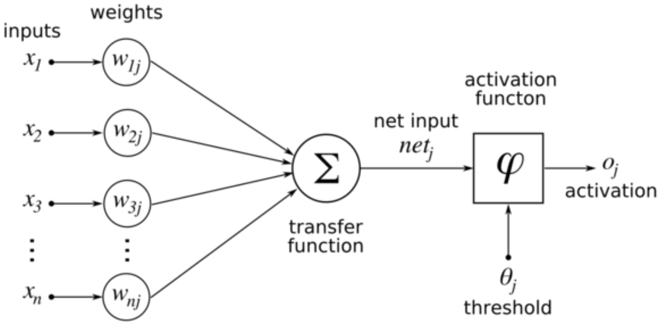


$$J(b) = -\sum_{i=1}^{m}\left(y^{(i)} \cdot \ln z^{(i)} + (1-y^{(i)}) \cdot \ln(1-z^{(i)})\right)$$

In practice...

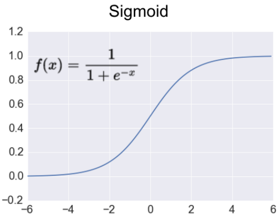


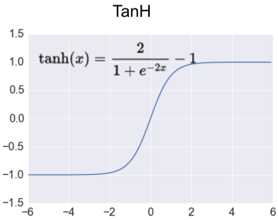
We want to mimic neurone firings

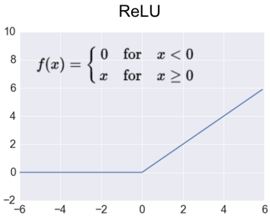


For gradient descent to work, we need activation functions that are

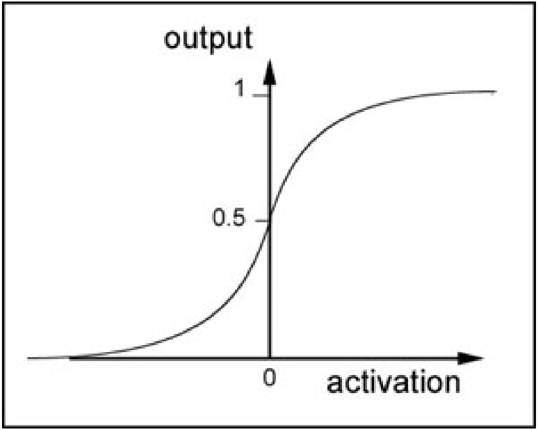
- Continuous
- Monotonically increasing
- **Differentiable**



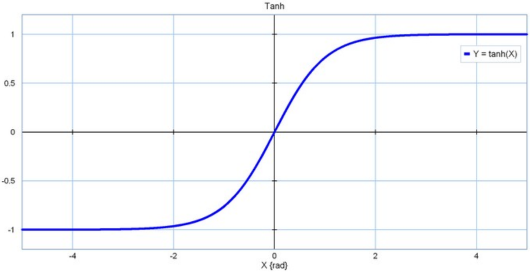




Sigmoid/Logistic

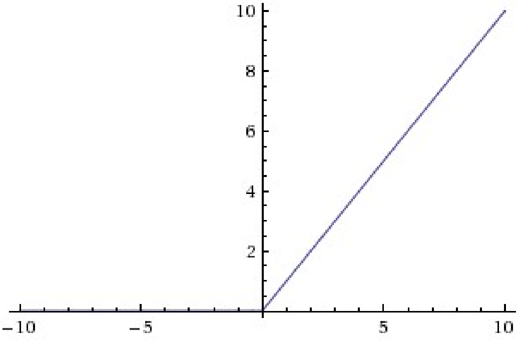


Hyperbolic Tangent (tanh)

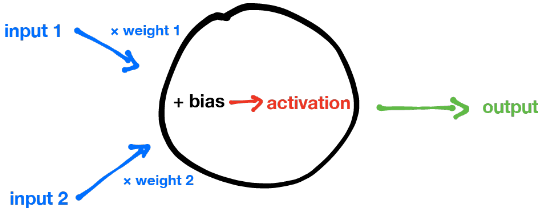


Rectified Linear Unit (ReLU)

$f(x) = \max(0, x)$



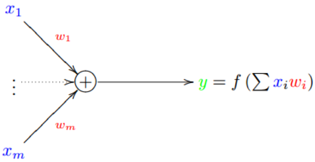
The Perceptron



- 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 ReLU(x))

The Perceptron

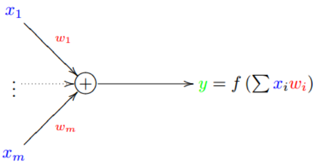
McCulloch and Pitts (1943) proposed the 'integrate and fire' model:



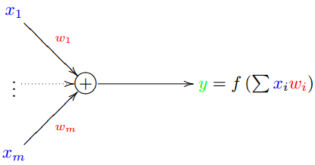
The Perceptron

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:



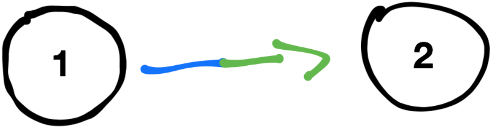
McCulloch and Pitts (1943) proposed the 'integrate and fire' model:



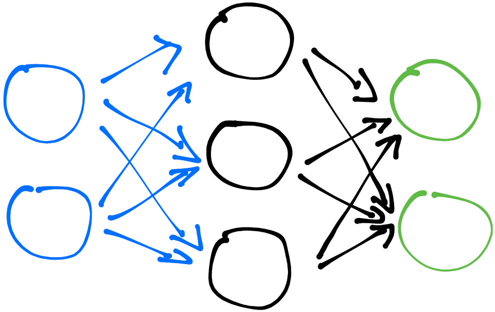
$$\sum_i x_i w_i = (1)(-2) + (4)(3) + (-2)(2) = 6$$

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

The Neural Network

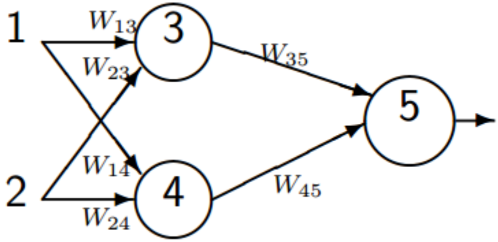


- The output of a neuron becomes the input for another neuron



- 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

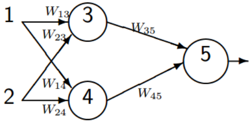
Feedforward Example



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

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

Feedforward



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

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

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

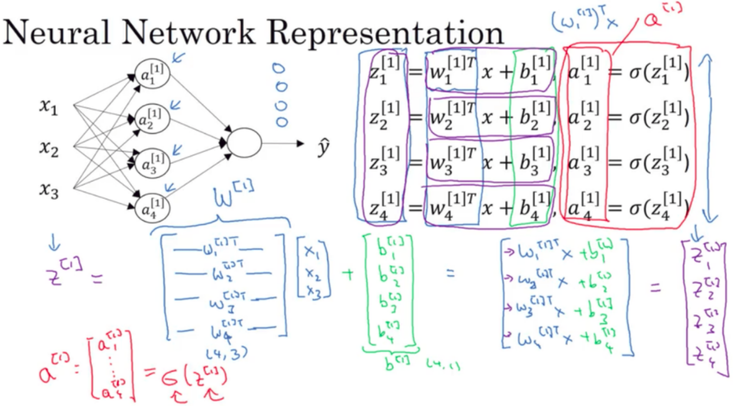
$$w_{14}(1) + w_{24}(2) = (1)(1) + (4)(2) = 9$$

$$z_3 = 0$$

$$z_4 = 1$$

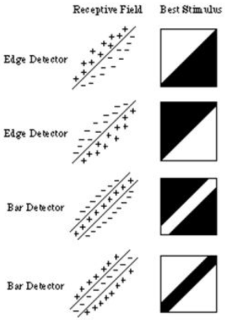
$$z_5 = f(w_{35}(0) + w_{45}(1)) = f((-1)(1)) = 0$$

Matrix Notation



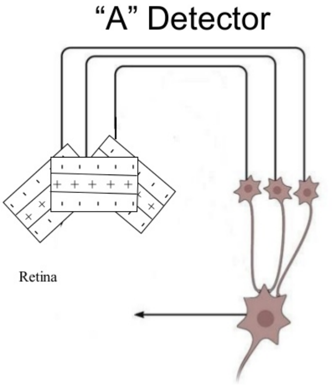
Why having more perceptrons works?

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



Some intuition

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



What does a neural network do?

- Neural networks are function approximators
  - So what?
  - Everything we are interested in is a function!
- What is a function?
  - Anything that maps an input to a single output

$$f : X \rightarrow Y$$

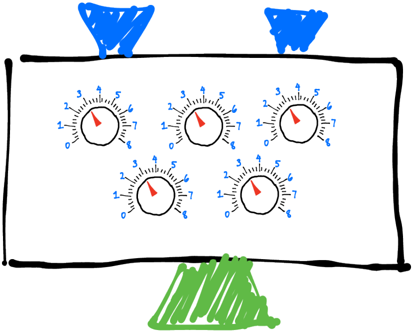
- Take for example a self driving car

$$f : X \to Y$$

  - 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

Plato

- Somewhere out there, there's a perfect function that tells you exactly what to do for some sensor input (**Platonist view**)
  - We want to approximate that function using a neural network
  - Using training data we've obtained from somewhere



- 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

Learning

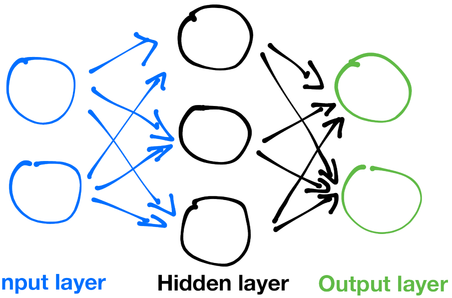
- So this architecture can (theoretically) approximate any function.
- But how do we actually find the correct parameters?
- Gradient Descent!

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



Neural network:

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

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

Coding Demo

Questions

Questions?