Neural Networks

CISC 7026: Introduction to Deep Learning

University of Macau

1. We looked at linear and polynomial f
    1. Looked at both classification and regression
    2. They have problems
        1. Input features scale poorly
        2. Bad performance around edges
    3. Neural networks fix many of these problems
    4. What is a neural network?
        1. Draw linear model as neural network
    5. Based on theory of the brain
        1. Invented ages ago
        2. Only recently have we learned to harness them

6. Neuron theory
    1. Connectivity
    2. Activation function
7. Parallels between real/artificial neuron
8. Matrix/graph duality
9. Single layer perceptron
10. Issues with one layer
    1. Not universal function approximator
11. Backprops
    1. Provides a way to train nn
        1. Assigns “fault” for each neuron

2. Recall closed form for linear model
    1. We use the gradient of the linear model
3. We use a similar approach

1. Limitations of linear models
2. History and overview of neural networks
3. Neurons
4. Perceptron
5. Multilayer Perceptron
6. Backpropagation
7. Gradient descent

We previously looked at linear and polynomial models for regression

We previously looked at linear and polynomial models for regression

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta}\boldsymbol{x} = \theta_0 + \theta_1 x_1 + \theta_2 x_2, ...$$

We previously looked at linear and polynomial models for regression

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta} \boldsymbol{x} = \theta_0 + \theta_1 x_1 + \theta_2 x_2, ...$$

$$\boldsymbol{\theta} = \left(\boldsymbol{X}^\top \boldsymbol{X} \right)^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

Linear models are useful for simple problems

Linear models are useful for simple problems

Issues with very complex problems

Linear models are useful for simple problems

Issues with very complex problems

1. Poor scalability

Linear models are useful for simple problems

Issues with very complex problems

1. Poor scalability
2. Prone to overfitting

Linear models are useful for simple problems

Issues with very complex problems

1. Poor scalability
2. Prone to overfitting
3. Polynomials do not generalize well

Issues with very complex problems

1. **Poor scalability**
2. Polynomials do not generalize well

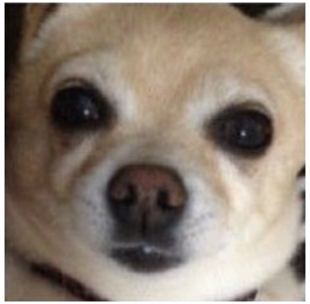
Polynomials fit tabular data well

Polynomials fit tabular data well

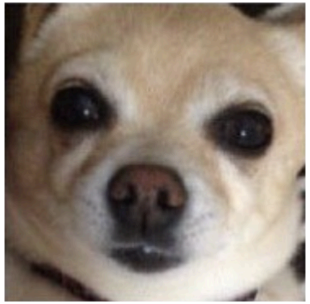
However, they scale poorly to higher-dimensional data like image pixels

Polynomials fit tabular data well

However, they scale poorly to higher-dimensional data like image pixels

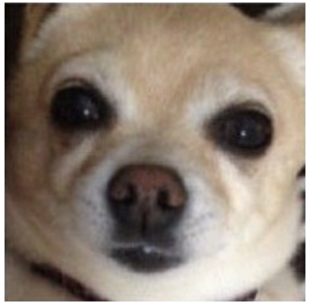


$$256 \times 256 \text{ pixels} = 65536 \text{ pixels}$$



$$256 \times 256 \text{ pixels} = 65536 \text{ pixels}$$

What does the design matrix look like for an n-degree polynomial?



$$256 \times 256 \text{ pixels} = 65536 \text{ pixels}$$

What does the design matrix look like for an n-degree polynomial?

$$\boldsymbol{X} = \begin{bmatrix} x_1^n & x_1^{n-1} & \dots & x_1^1 & 1 \\ x_2^n & x_2^{n-1} & \dots & x_2^1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_p^n & x_p^{n-1} & \dots & x_p^1 & 1 \\ x_1^{n-1} x_2 & x^{n-2} x_2^2 & \dots & 0 & 1 \\ \vdots & \vdots & & \vdots & \vdots \end{bmatrix}$$

$$\boldsymbol{X} = \begin{bmatrix} x_1^n & x_1^{n-1} & \dots & x_1^1 & 1 \\ x_2^n & x_2^{n-1} & \dots & x_2^1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_p^n & x_p^{n-1} & \dots & x_p^1 & 1 \\ x_1^{n-1} x_2 & x^{n-2} x_2^2 & \dots & 0 & 1 \\ \vdots & \vdots & & \vdots & \vdots \end{bmatrix}$$

Question: How big is the matrix for 65,536 pixels and $n = 3$?

Answer: $65, 536^3 \approx 10^{14}$ parameters

$$\boldsymbol{X} = \begin{bmatrix} x_1^n & x_1^{n-1} & \dots & x_1^1 & 1 \\ x_2^n & x_2^{n-1} & \dots & x_2^1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_p^n & x_p^{n-1} & \dots & x_p^1 & 1 \\ x_1^{n-1} x_2 & x^{n-2} x_2^2 & \dots & 0 & 1 \\ \vdots & \vdots & & \vdots & \vdots \end{bmatrix}$$

Question: How big is the matrix for 65,536 pixels and $n = 3$?

Answer: $65,536^3 \approx 10^{14}$ parameters

Question: How big is the matrix for 65,536 pixels and $n = 3$?

Answer: $65,536^3 \approx 10^{14}$ parameters

Question: How big is the matrix for 65,536 pixels and $n = 3$?

Answer: $65,536^3 \approx 10^{14}$ parameters

We must invert $\boldsymbol{X}^\top \boldsymbol{X}$, requiring $O(n^3)$ time

Question: How big is the matrix for 65,536 pixels and $n = 3$?

Answer: $65,536^3 \approx 10^{14}$ parameters

We must invert $\boldsymbol{X}^\top \boldsymbol{X}$, requiring $O(n^3)$ time

Largest matrix ever inverted is $\approx 10^{12}$

Question: How big is the matrix for 65,536 pixels and $n = 3$?

Answer: $65,536^3 \approx 10^{14}$ parameters

We must invert $\boldsymbol{X}^\top \boldsymbol{X}$, requiring $O(n^3)$ time

Largest matrix ever inverted is $\approx 10^{12}$

For comparison, GPT-4 has 10^{12} parameters

Question: How big is the matrix for 65,536 pixels and $n = 3$?

Answer: $65,536^3 \approx 10^{14}$ parameters

We must invert $\boldsymbol{X}^\top \boldsymbol{X}$, requiring $O(n^3)$ time

Largest matrix ever inverted is $\approx 10^{12}$

For comparison, GPT-4 has 10^{12} parameters

Polynomial regression scales poorly to high dimensional data

Issues with very complex problems

1. **Poor scalability**
2. Polynomials do not generalize well

Issues with very complex problems

1. Poor scalability
2. **Polynomials do not generalize well**

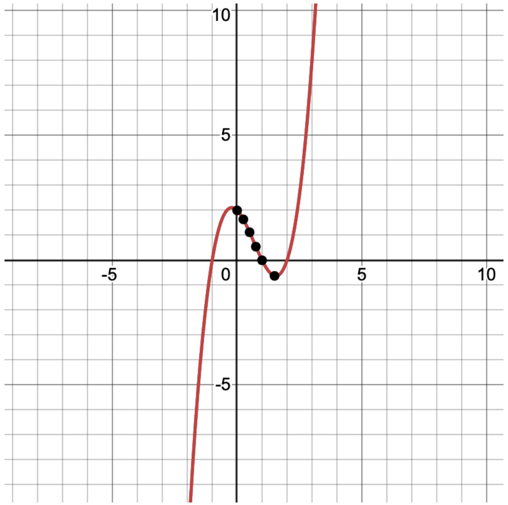
Polynomials tend towards $-\infty, \infty$ outside of the support

Polynomials tend towards $-\infty, \infty$ outside of the support

$$f(x) = x^3 - 2x^2 - x + 2$$

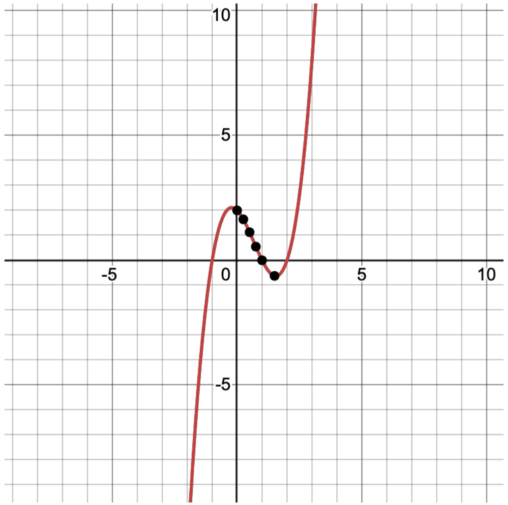
Polynomials tend towards $-\infty, \infty$ outside of the support

$$f(x) = x^3 - 2x^2 - x + 2$$



Polynomials tend towards $-\infty, \infty$ outside of the support

$$f(x) = x^3 - 2x^2 - x + 2$$



If breed of dog missing from training set, we still want to classify it as dog!

Linear and polynomial regression have issues

Linear and polynomial regression have issues

1. Poor scalability

Linear and polynomial regression have issues

1. Poor scalability
2. Polynomials do not generalize well

Relax

Can we improve upon the linear/polynomial model?

Can we improve upon the linear/polynomial model?

Yes, with neural networks

Can we improve upon the linear/polynomial model?

Yes, with neural networks

TODO: diagram/flowchart history of ML

Brain: Biological neurons → Biological neural network

Brain: Biological neurons $\rightarrow$ Biological neural network

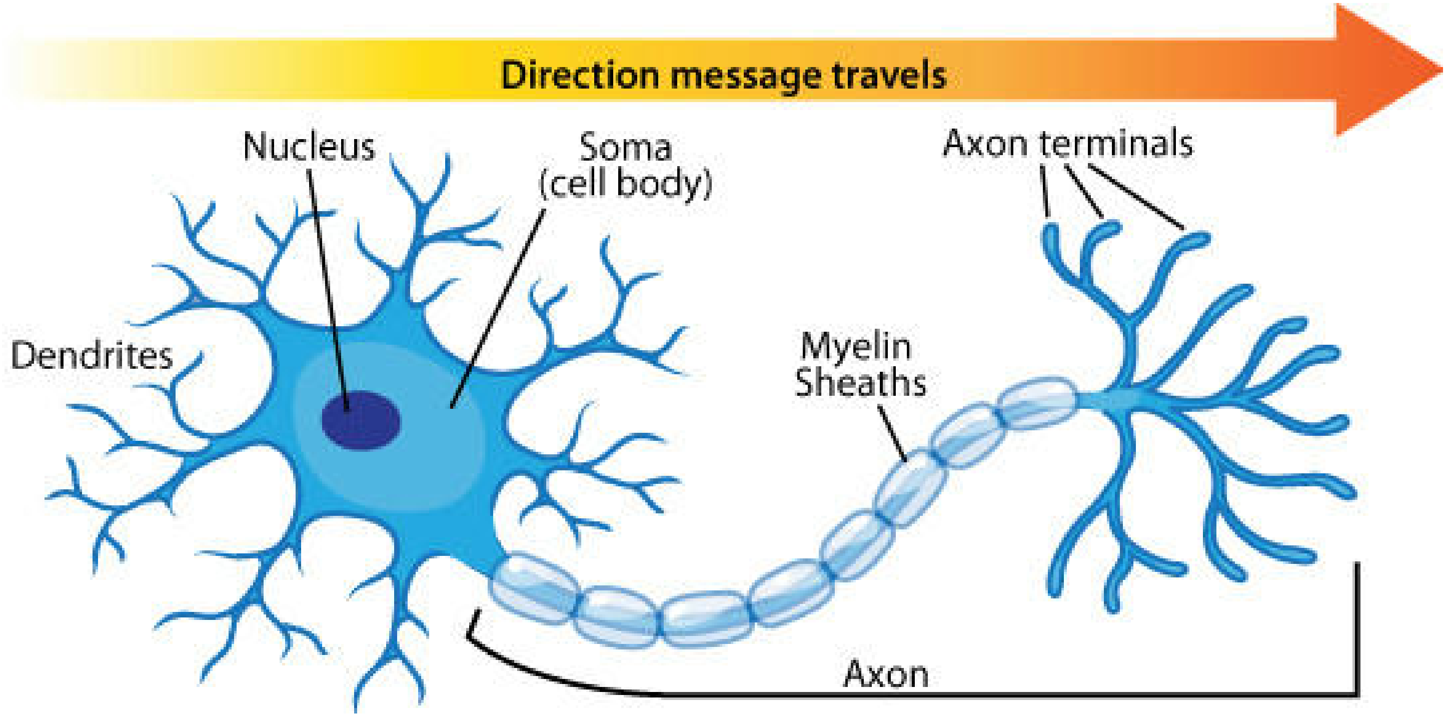
Computer: Artificial neurons $\rightarrow$ Artificial neural network

Brain: Biological neurons $\rightarrow$ Biological neural network

Computer: Artificial neurons $\rightarrow$ Artificial neural network

Neurons send and receive electrical impulses along axons and dendrites

Neurons send and receive electrical impulses along axons and dendrites



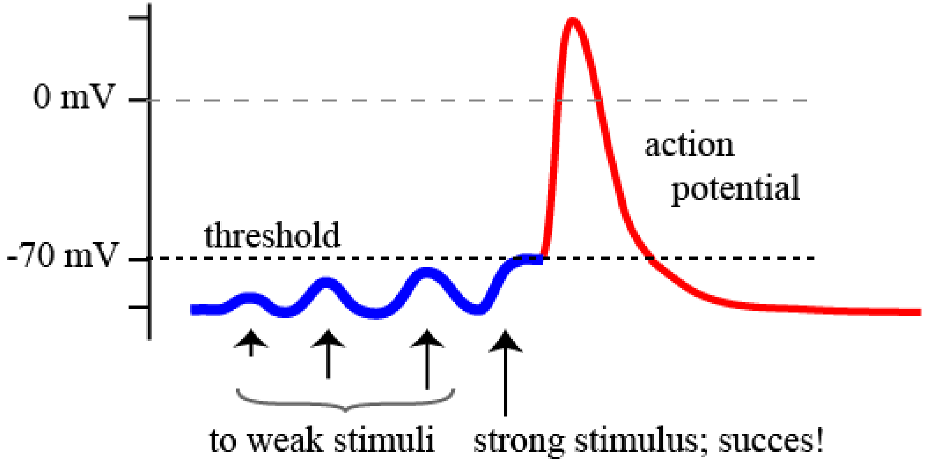
How does a neuron send an impulse ("fire")?

How does a neuron send an impulse ("fire")?

Incoming impulses (via dendrites) change the electric potential of the neuron

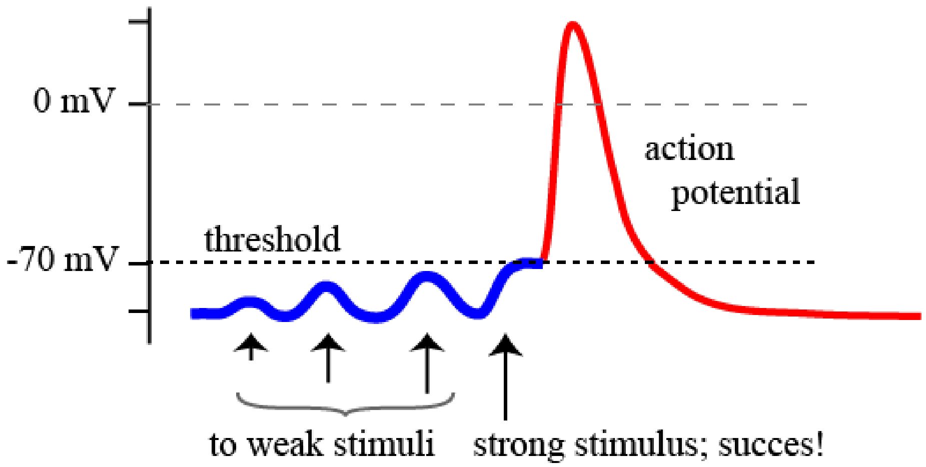
How does a neuron send an impulse (“fire”)?

Incoming impulses (via dendrites) change the electric potential of the neuron

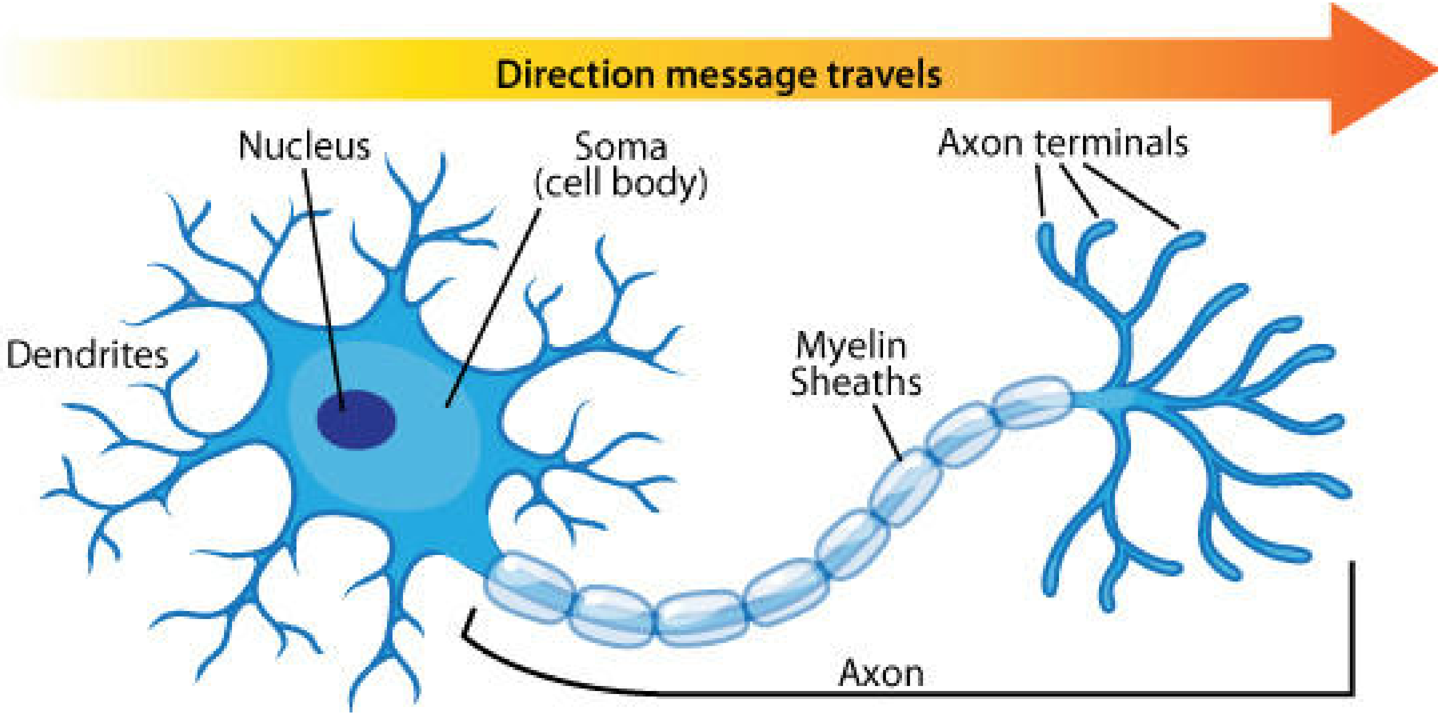


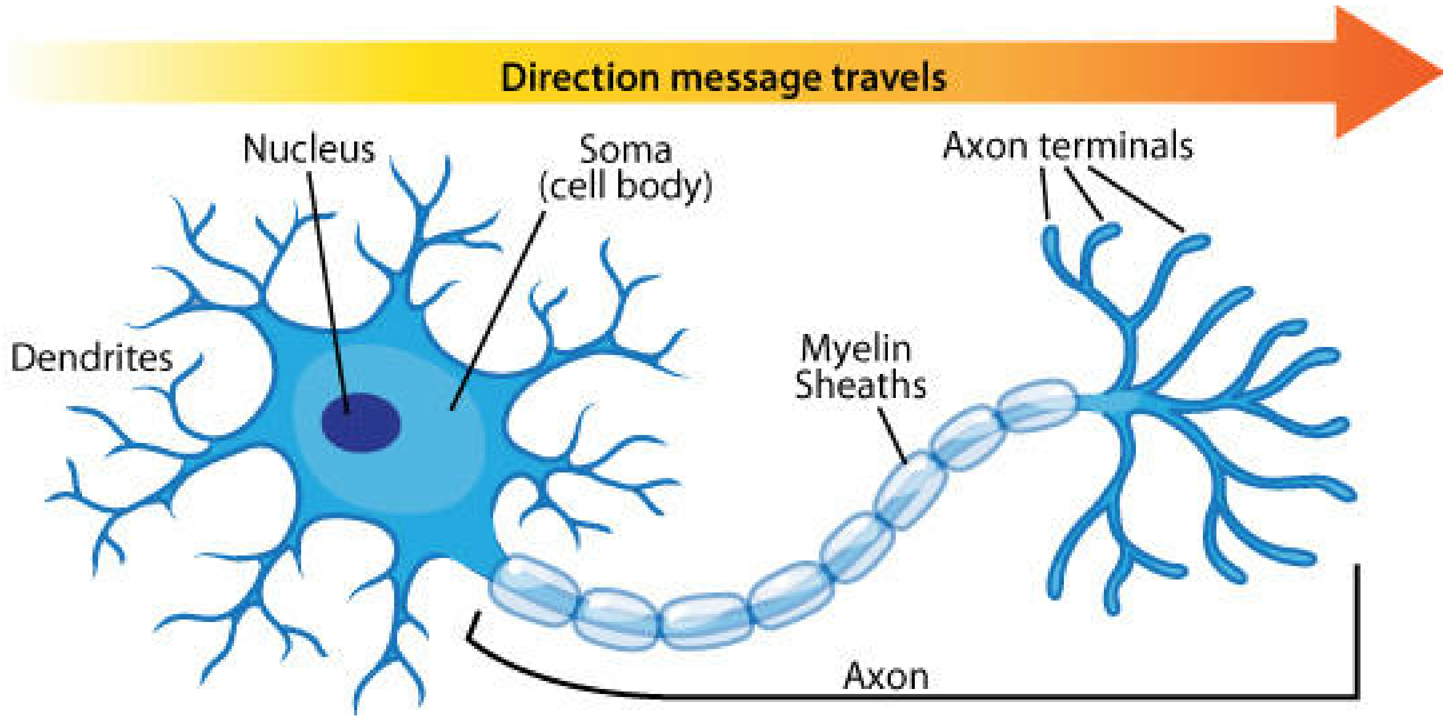
How does a neuron send an impulse ("fire")?

Incoming impulses (via dendrites) change the electric potential of the neuron



Pain triggers initial nerve impulse, sets of impulse chain into the brain

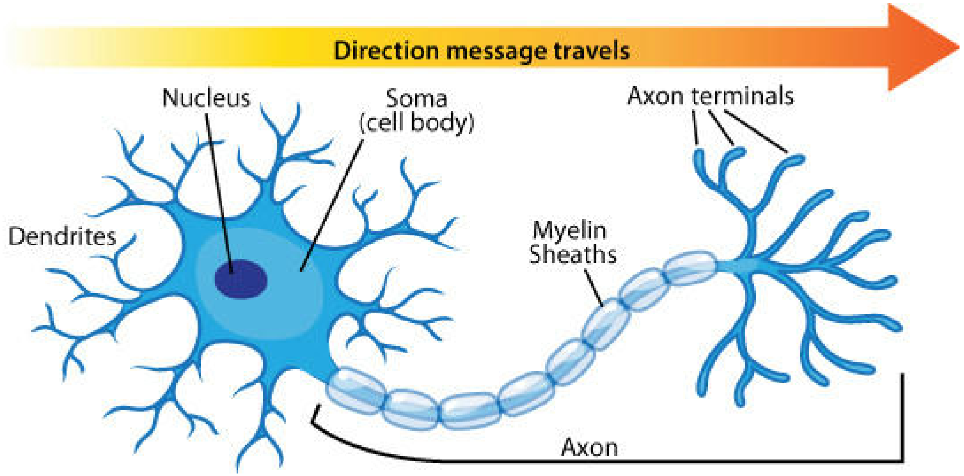

Direction message travels
Nucleus
Soma
(cell body)
Axon terminals
Dendrites
Myelin
Sheaths
Axon



Question: How would you model a neuron mathematically?

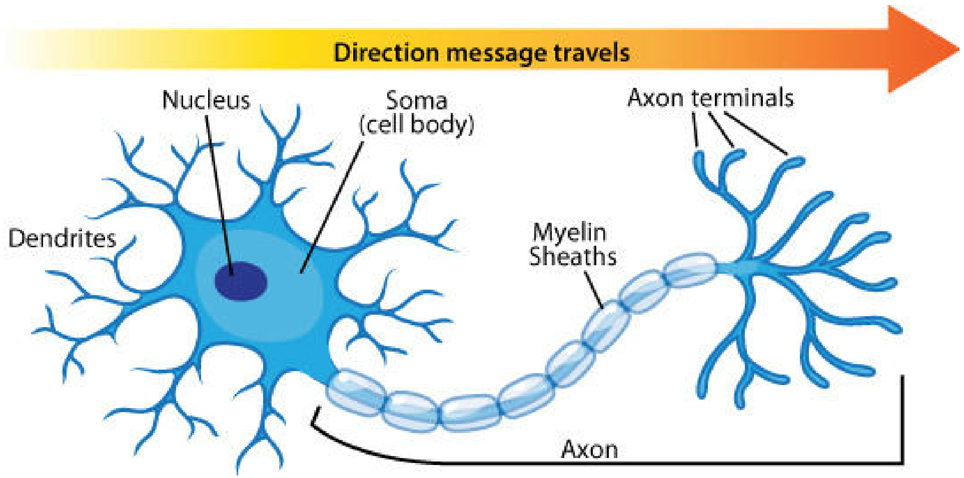
Let us define a neuron as a function

Let us define a neuron as a function



Neuron has a structure of dendrites

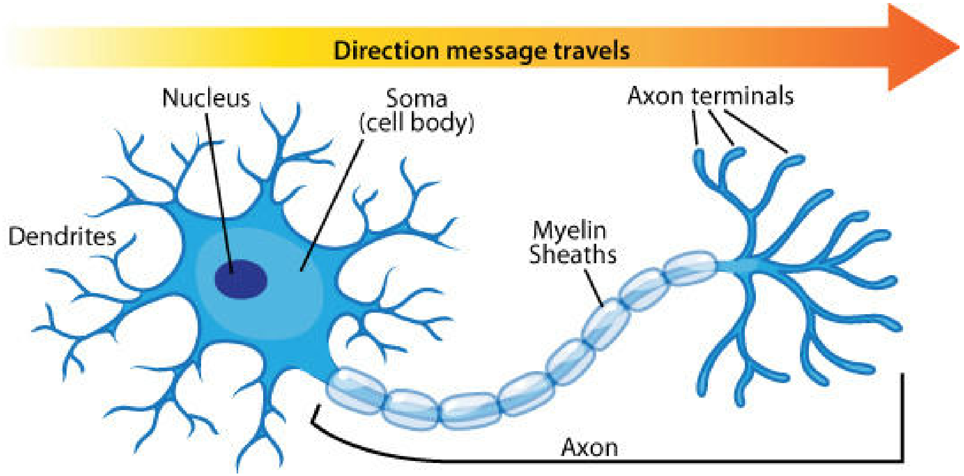
Let us define a neuron as a function



Neuron has a structure of dendrites

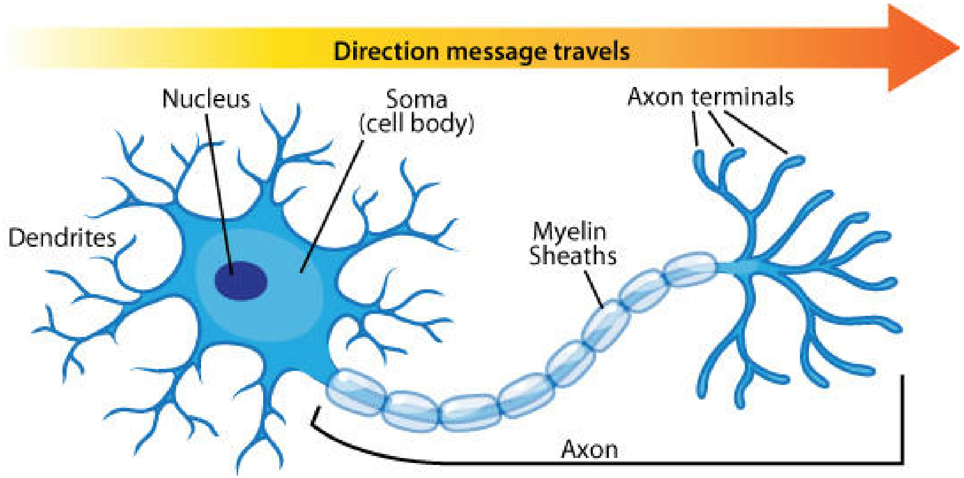
$$f\left(\begin{bmatrix}\theta_1 \\ \theta_2 \\ \vdots \\ \theta_n\end{bmatrix}\right) = f\left(\begin{bmatrix}1 \\ 0 \\ \vdots \\ 1\end{bmatrix}\right)$$

Let us define a neuron as a function



Each incoming dendrite has some voltage potential

Let us define a neuron as a function

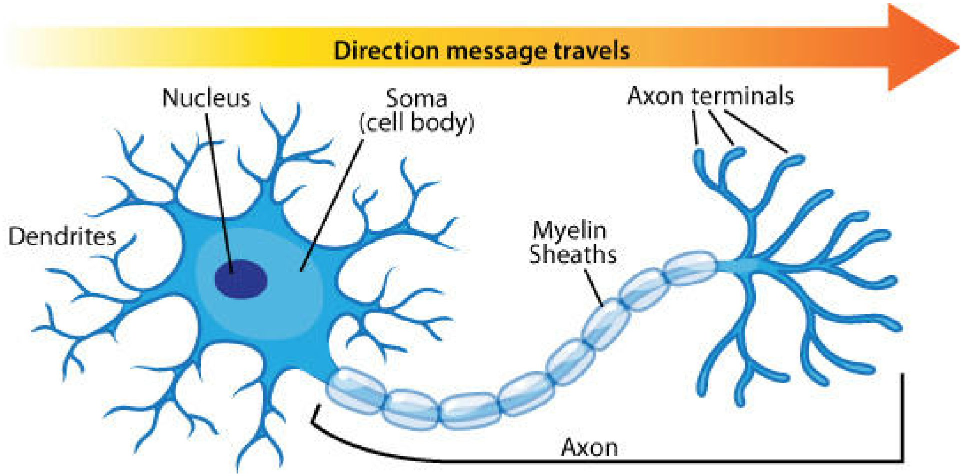


Each incoming dendrite has some voltage potential

$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}\right)$$

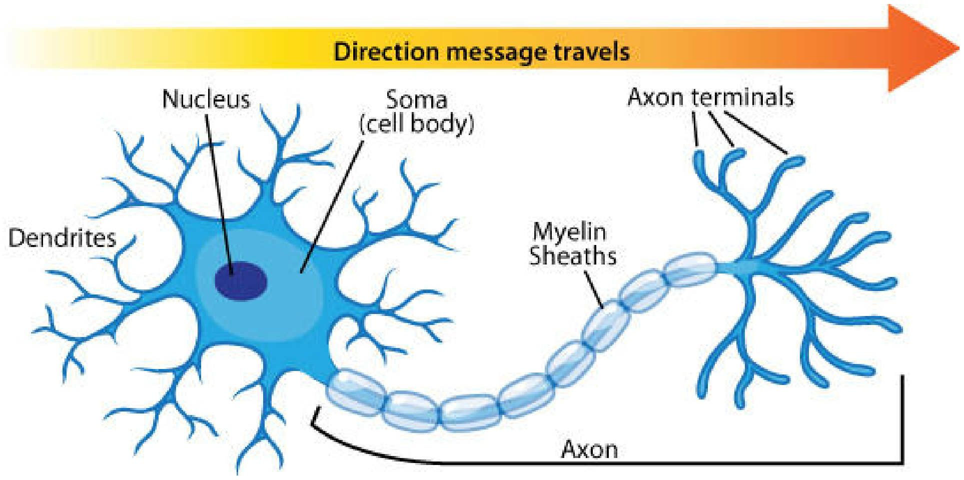
$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 0.5 \\ \vdots \\ -0.3 \end{bmatrix}$$

Let us define a neuron as a function



Voltage potentials sum together to give us the voltage in the cell body

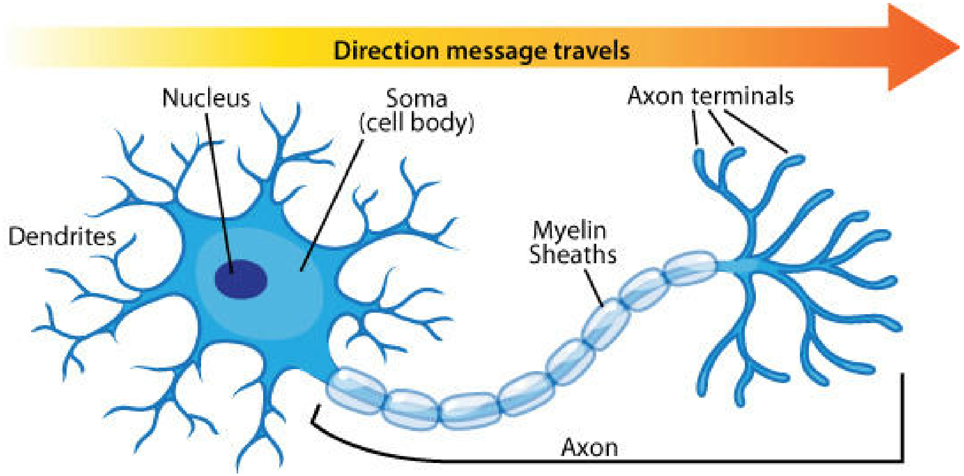
Let us define a neuron as a function



Voltage potentials sum together to give us the voltage in the cell body

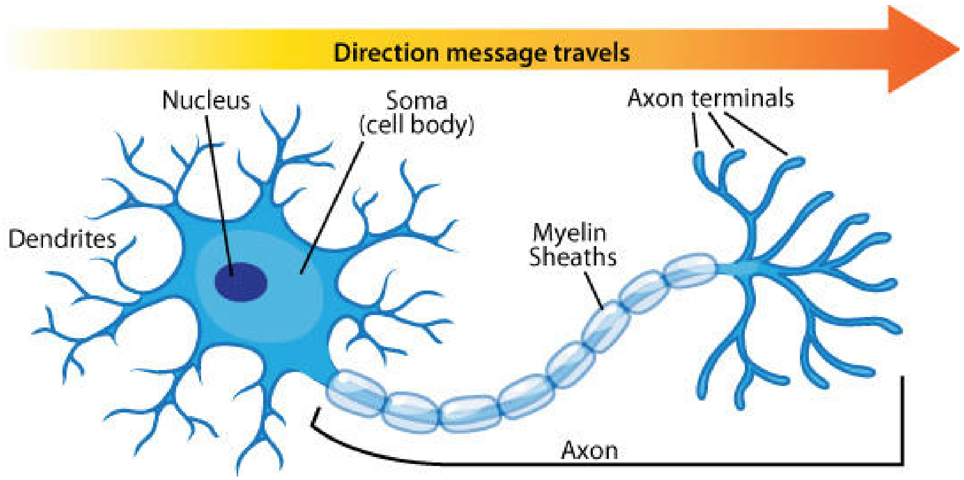
$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}\right) = \sum_{i=1}^{n} x_i \theta_i$$

Let us define a neuron as a function



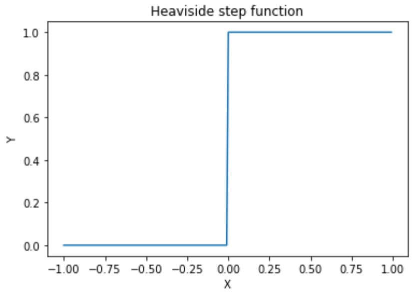
The axon fires only if the voltage is over a threshold

Let us define a neuron as a function

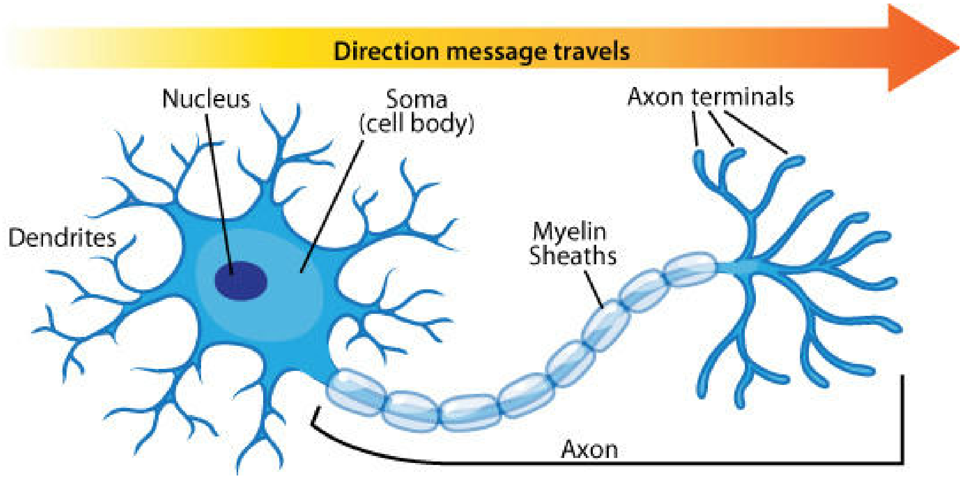


The axon fires only if the voltage is over a threshold

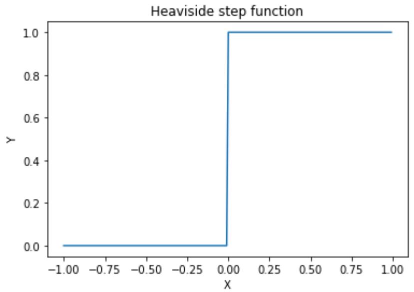
$$\sigma(x) =$$



Let us define a neuron as a function



The axon fires only if the voltage is over a threshold

$$\sigma(x) =$$ 

$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \right) = \sigma\left(\sum_{i=1}^{n} x_i \theta_i \right)$$

This is almost the artificial neuron!

$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}\right) = \sigma\left(\sum_{i=1}^{n} x_i \theta_i\right)$$

This is almost the artificial neuron!

$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}\right) = \sigma\left(\sum_{i=1}^{n} x_i \theta_i\right)$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\sum_{i=1}^{n} x_i \theta_i\right)$$

Question: Does it look familiar to any other functions we have seen?

This is almost the artificial neuron!

$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \right) = \sigma \left(\sum_{i=1}^{n} x_i \theta_i \right)$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma \left(\sum_{i=1}^{n} x_i \theta_i \right)$$

Question: Does it look familiar to any other functions we have seen?

Answer: The linear model!

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\sum_{i=1}^{n} x_i \theta_i\right)$$

Artificial neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\sum_{i=1}^{n} x_i \theta_i\right)$$

Artificial neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

Linear model

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\sum_{i=1}^{n} x_i \theta_i\right)$$

Artificial neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

Linear model

It is the linear model with an activation function!

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\sum_{i=1}^{n} x_i \theta_i\right)$$

Artificial neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

Linear model

It is the linear model with an activation function!

We add a bias term to the neuron, for the same reason we add a bias term to the linear model

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\sum_{i=1}^{n} x_i \theta_i\right)$$

Artificial neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

Linear model

It is the linear model with an activation function!

We add a bias term to the neuron, for the same reason we add a bias term to the linear model

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\theta_0 + \sum_{i=1}^{n} x_i \theta_i\right)$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\sum_{i=1}^{n} x_i \theta_i\right)$$

Artificial neuron

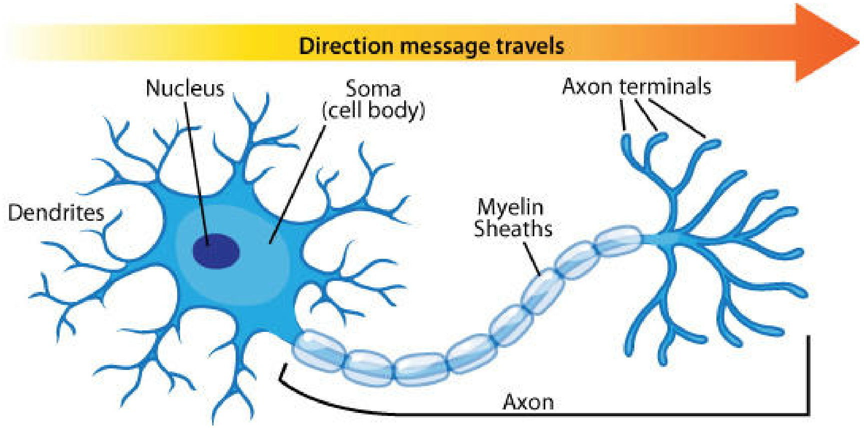
$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

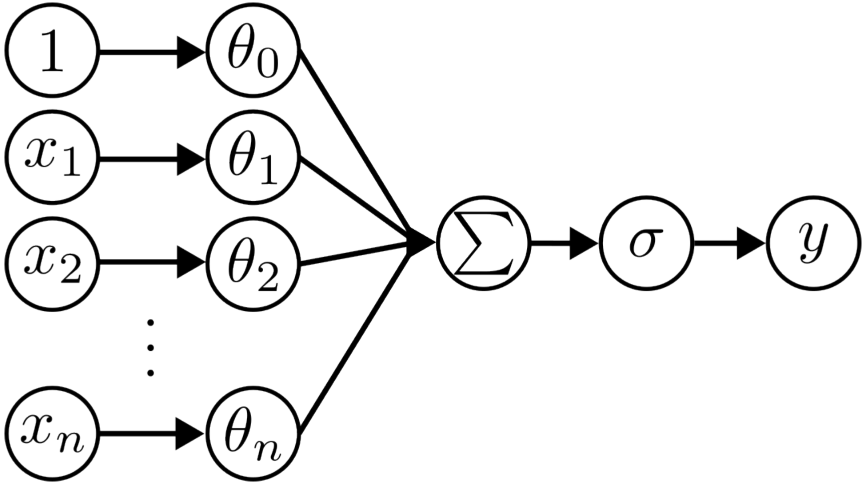
Linear model

It is the linear model with an activation function!

We add a bias term to the neuron, for the same reason we add a bias term to the linear model

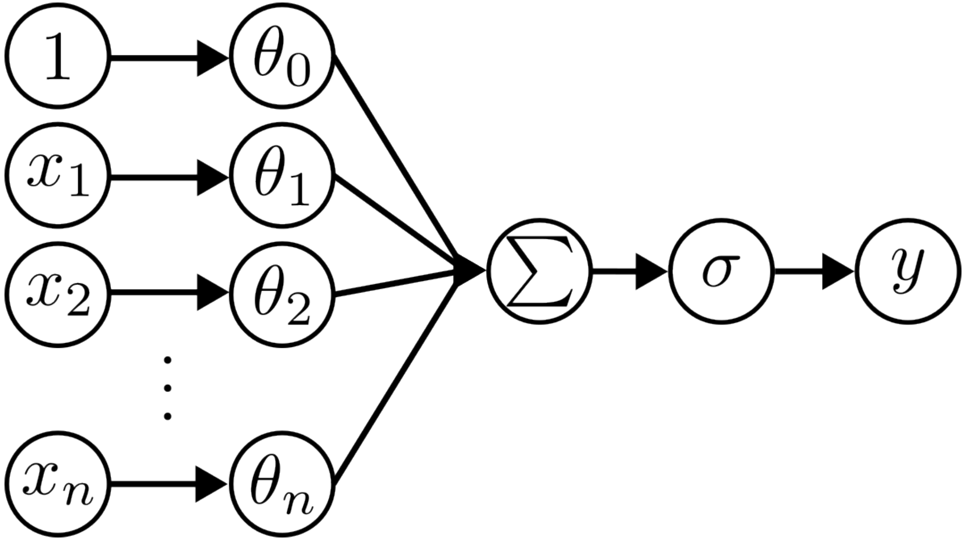
$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\theta_0 + \sum_{i=1}^{n} x_i \theta_i\right)$$

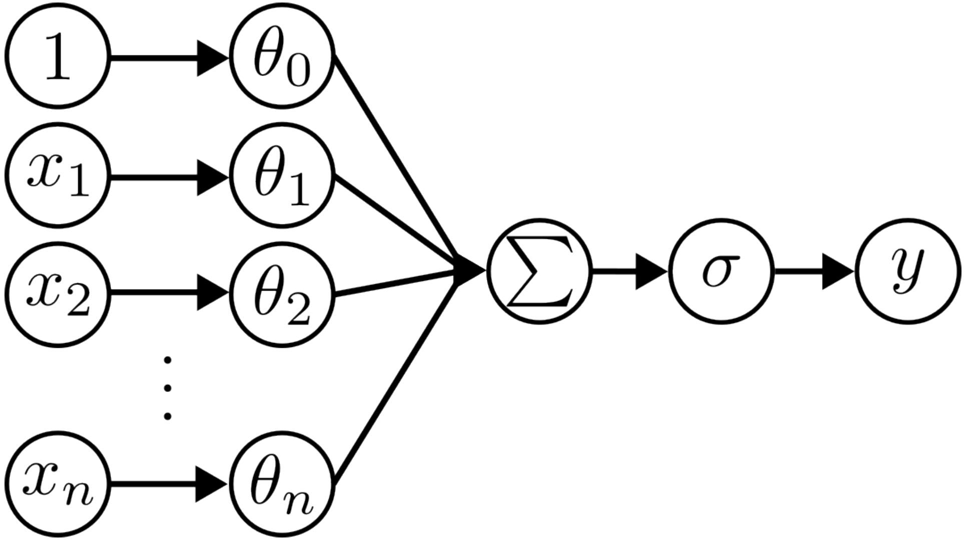




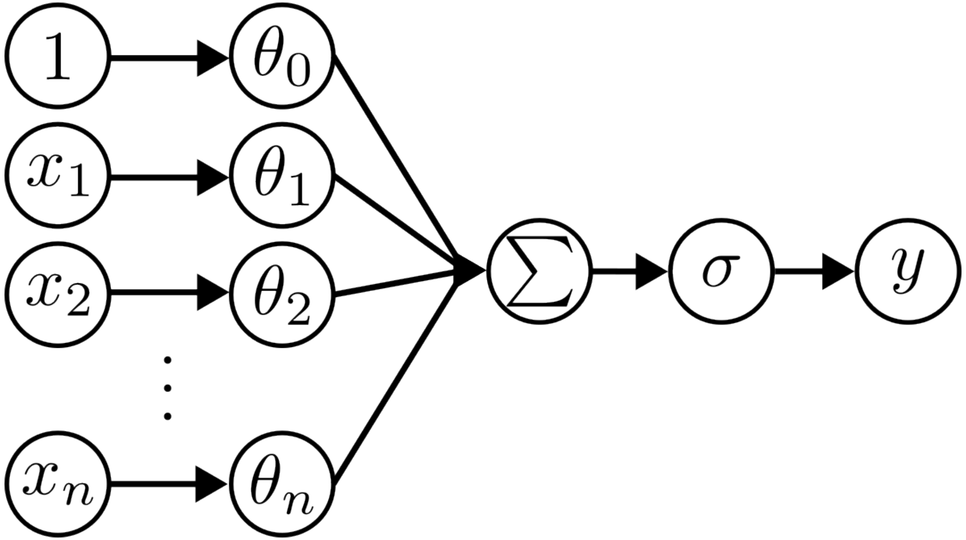
$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}\right) = \sigma\left(\theta_0 + \sum_{i=1}^{n} x_i \theta_i\right)$$

Relax


1
x_1
x_2
x_n
θ_0
θ_1
θ_2
θ_n
$\sum$
σ
y

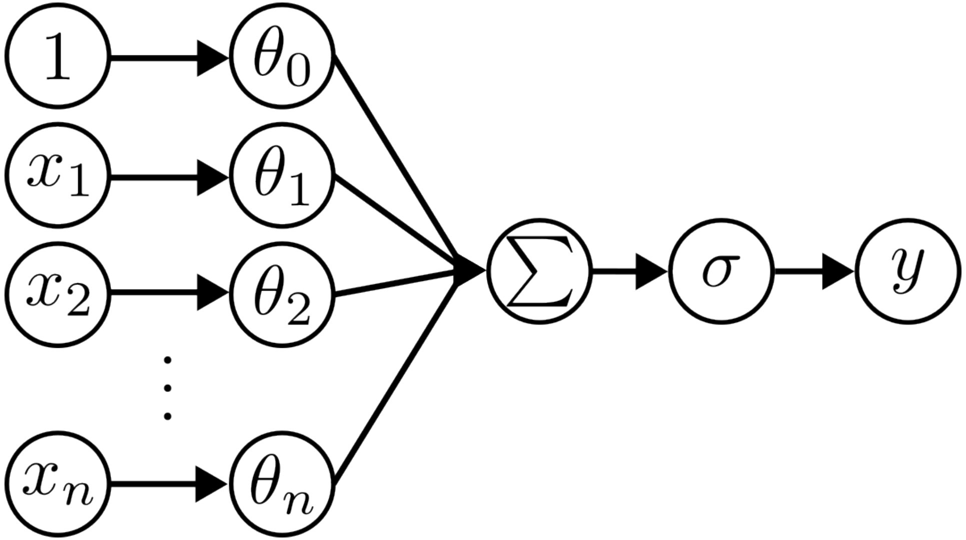


Recall that in machine learning we deal with functions



Recall that in machine learning we deal with functions

What kinds of functions can our neuron represent?

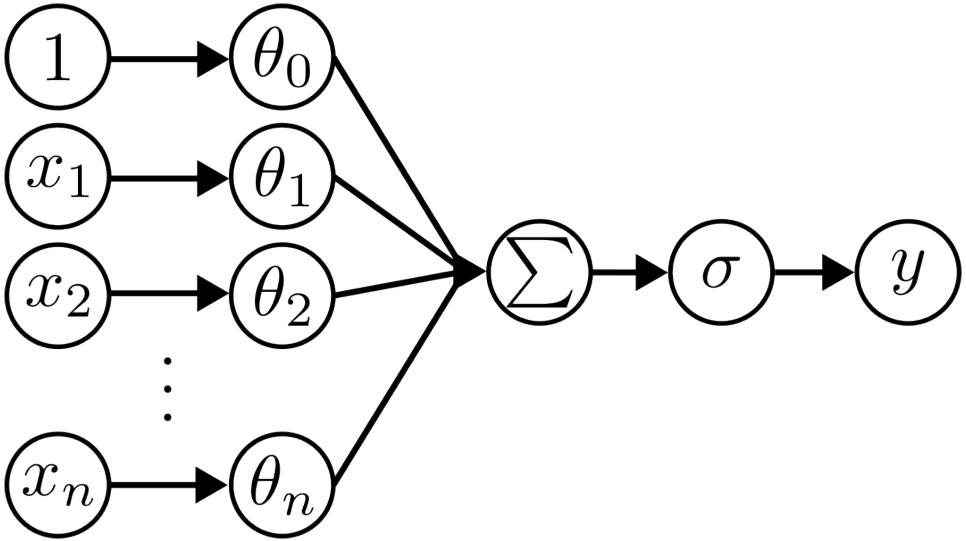


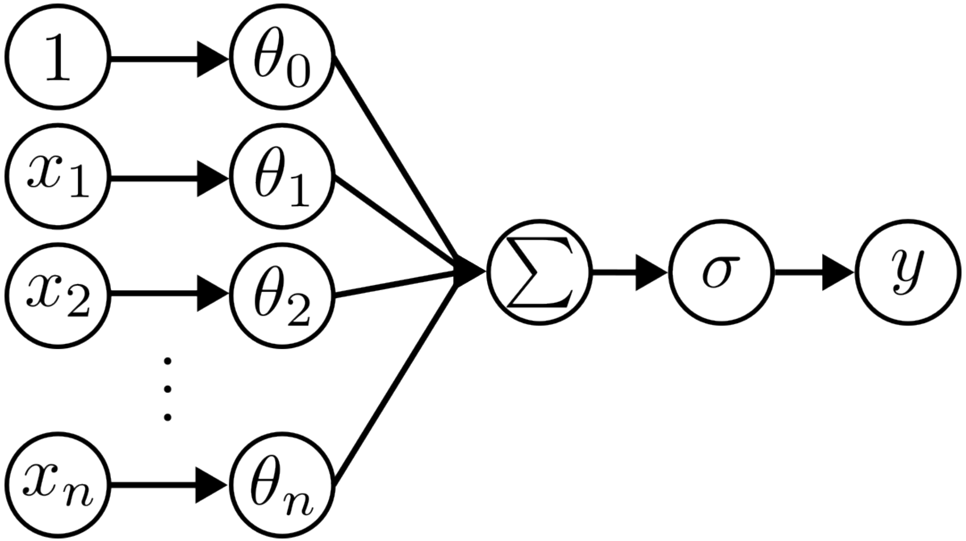
Recall that in machine learning we deal with functions

What kinds of functions can our neuron represent?

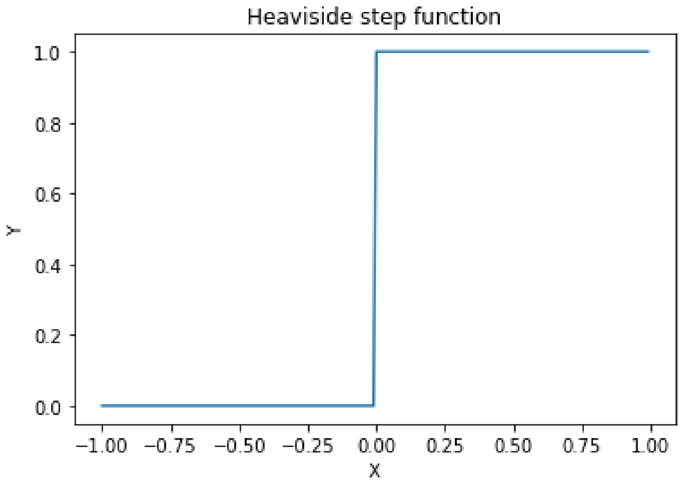
Let us start with a logical AND function

Recall the activation function (Heaviside step)





Recall the activation function (Heaviside step)



$$H(x) = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ if } x \leq 0 \end{cases}$$

Implement AND using an artificial neuron

Implement AND using an artificial neuron

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1\theta_1 + x_2\theta_2)$$

Implement AND using an artificial neuron

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1\theta_1 + x_2\theta_2)$$

$$\boldsymbol{\theta} = [\theta_0 \ \ \theta_1 \ \ \theta_2]^\top = [-1 \ \ 1 \ \ 1]^\top$$

Implement AND using an artificial neuron

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1\theta_1 + x_2\theta_2)$$

$$\boldsymbol{\theta} = [\theta_0 \;\; \theta_1 \;\; \theta_2]^\top = [-1 \;\; 1 \;\; 1]^\top$$

x_1	x_2	y	$f(x_1, x_2, \boldsymbol{\theta})$	$\hat{y}$
0	0	0	$H(-1 + 1 \cdot 0 + 1 \cdot 0) = H(-1)$	0
0	1	0	$H(-1 + 1 \cdot 0 + 1 \cdot 1) = H(0)$	0
1	0	0	$H(-1 + 1 \cdot 1 + 1 \cdot 0) = H(0)$	0
1	1	1	$H(-1 + 1 \cdot 1 + 1 \cdot 1) = H(1)$	1

Implement OR using an artificial neuron

Implement OR using an artificial neuron

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1\theta_1 + x_2\theta_2)$$

Implement OR using an artificial neuron

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1 \theta_1 + x_2 \theta_2)$$

$$\boldsymbol{\theta} = [\theta_0 \;\; \theta_1 \;\; \theta_2]^\top = [0 \;\; 1 \;\; 1]^\top$$

Implement OR using an artificial neuron

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1\theta_1 + x_2\theta_2)$$

$$\boldsymbol{\theta} = [\theta_0 \;\; \theta_1 \;\; \theta_2]^\top = [0 \;\; 1 \;\; 1]^\top$$

x_1	x_2	y	$f(x_1, x_2, \boldsymbol{\theta})$	$\hat{y}$
0	0	0	$H(0 + 1 \cdot 0 + 1 \cdot 0) = H(0)$	0
0	1	0	$H(0 + 1 \cdot 1 + 1 \cdot 0) = H(1)$	1
1	0	1	$H(0 + 1 \cdot 0 + 1 \cdot 1) = H(1)$	1
1	1	1	$H(1 + 1 \cdot 1 + 1 \cdot 1) = H(2)$	1

Implement XOR using an artificial neuron

Implement XOR using an artificial neuron

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1\theta_1 + x_2\theta_2)$$

Implement XOR using an artificial neuron

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1\theta_1 + x_2\theta_2)$$

$$\boldsymbol{\theta} = [\theta_0 \;\; \theta_1 \;\; \theta_2]^\top = [? \;\; ? \;\; ?]^\top$$

Implement XOR using an artificial neuron

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1\theta_1 + x_2\theta_2)$$

$$\boldsymbol{\theta} = [\theta_0 \;\; \theta_1 \;\; \theta_2]^\top = [? \;\; ? \;\; ?]^\top$$

x_1	x_2	y	$f(x_1, x_2, \boldsymbol{\theta})$	$\hat{y}$
0	0	0	This is IMPOSSIBLE!	
0	1	1		
1	0	1		
1	1	0		

Why can't we represent XOR using a neuron?

Why can't we represent XOR using a neuron?

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1\theta_1 + x_2\theta_2)$$

Why can't we represent XOR using a neuron?

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1 \theta_1 + x_2 \theta_2)$$

We can only represent $H(\text{linear function})$

Why can't we represent XOR using a neuron?

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1\theta_1 + x_2\theta_2)$$

We can only represent $H(\text{linear function})$

XOR is not a linear combination of x_1, x_2!

Why can't we represent XOR using a neuron?

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1\theta_1 + x_2\theta_2)$$

We can only represent $H(\text{linear function})$

XOR is not a linear combination of x_1, x_2!

We want to represent any function, not just linear functions

Why can't we represent XOR using a neuron?

$$f(x_1, x_2, \boldsymbol{\theta}) = H(\theta_0 + x_1\theta_1 + x_2\theta_2)$$

We can only represent $H(\text{linear function})$

XOR is not a linear combination of x_1, x_2!

We want to represent any function, not just linear functions

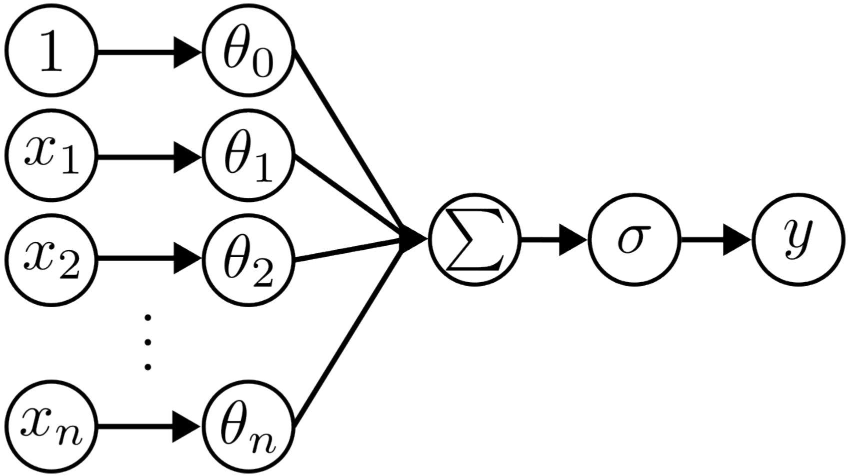
Let us think back to biology, maybe it has an answer

Brain: Biological neurons $\rightarrow$ Biological neural network

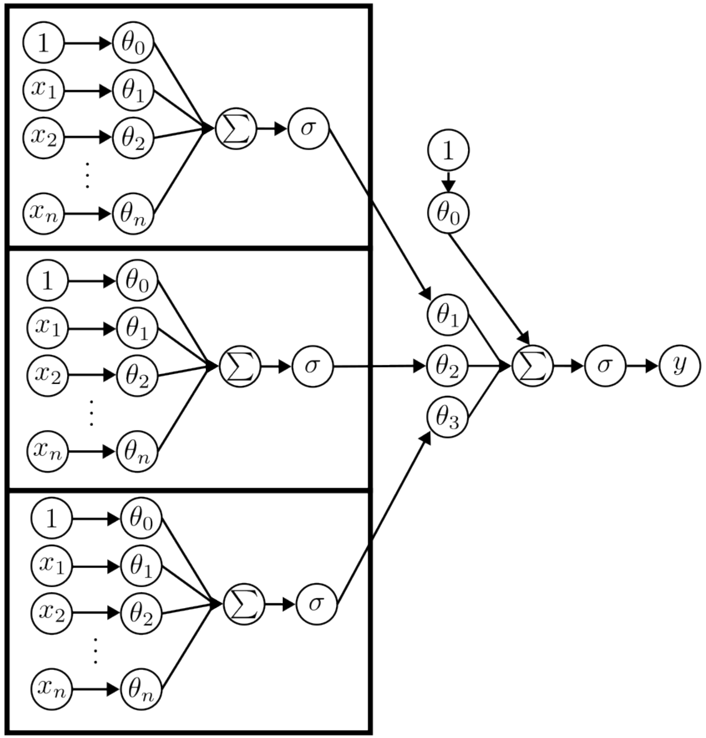
Brain: Biological neurons $\rightarrow$ Biological neural network

Computer: Artificial neurons $\rightarrow$ Artificial neural network

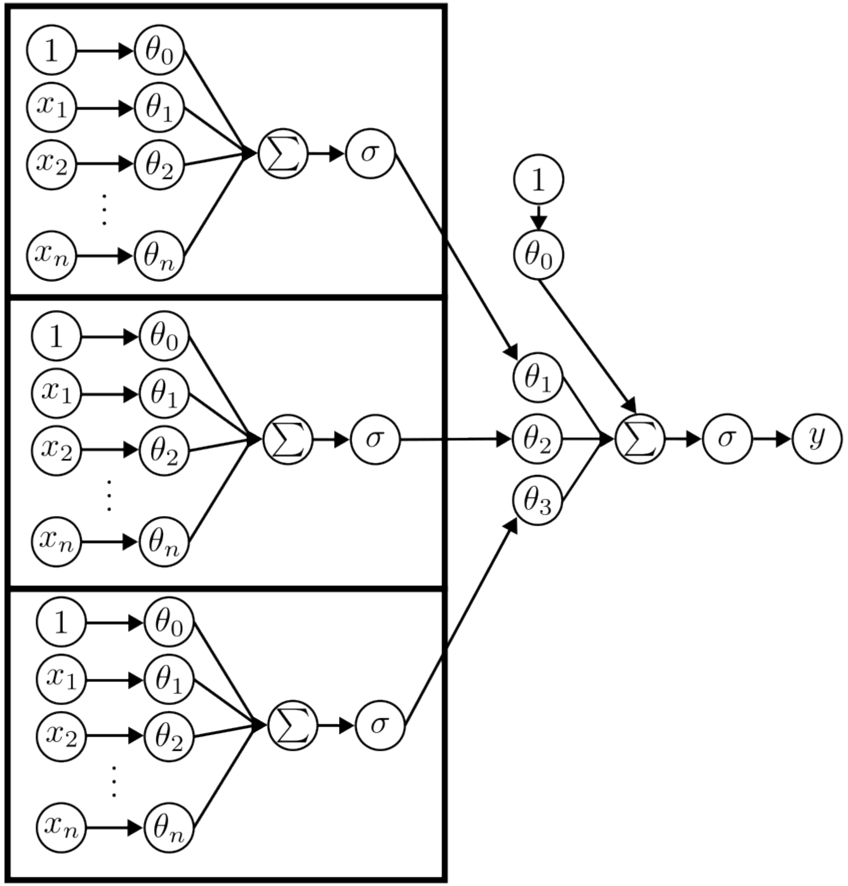
Connect artificial neurons into a network



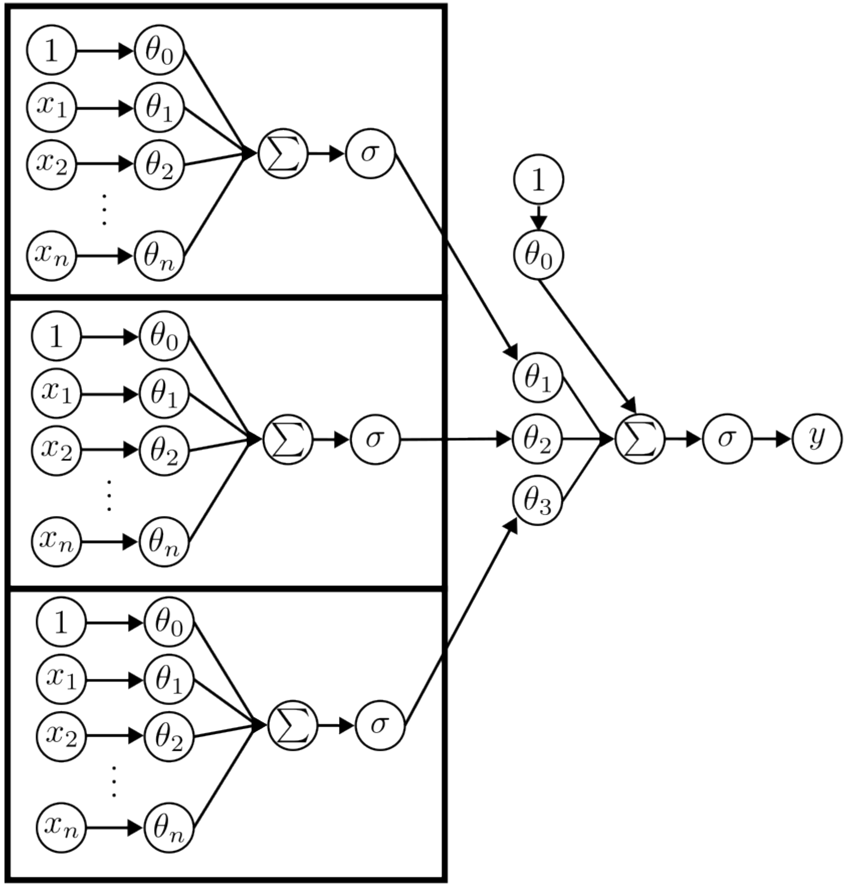
Neuron



Neural Network

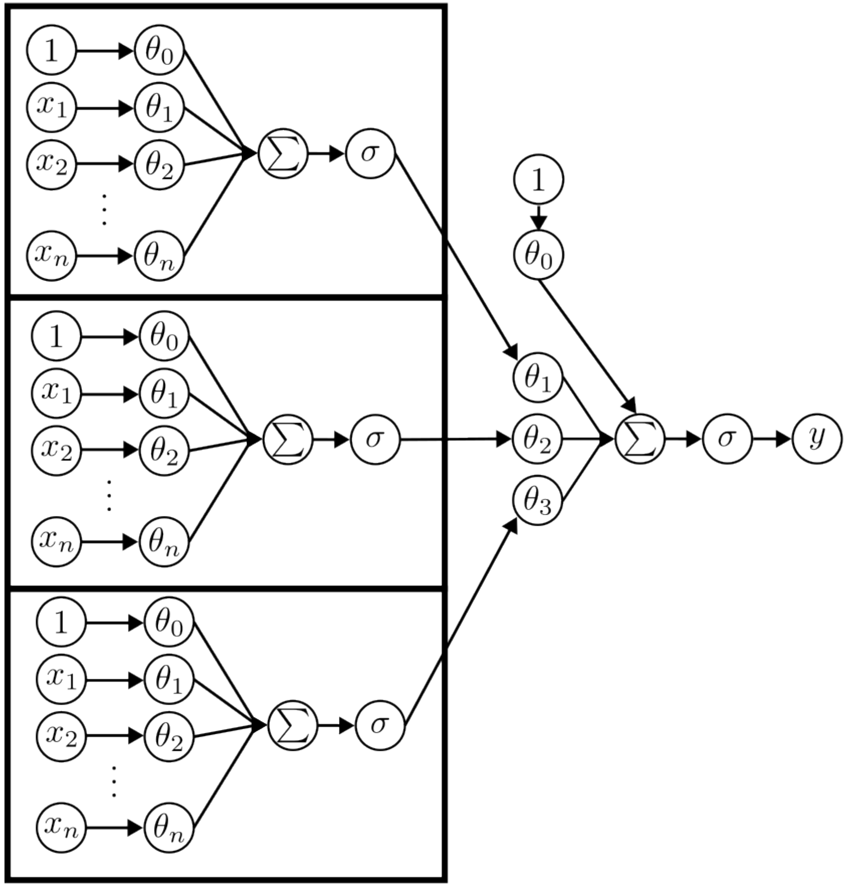


Adding neurons in **parallel** creates a **wide** neural network



Adding neurons in **parallel** creates a **wide** neural network

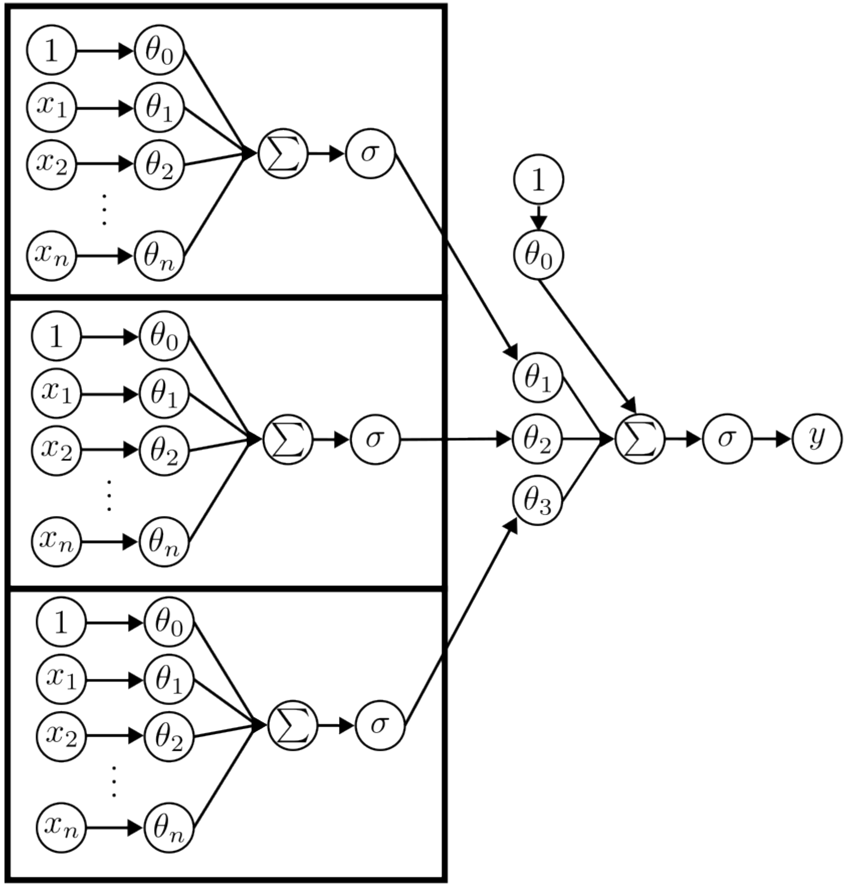
Adding neurons in **series** creates a **deep** neural network



Adding neurons in **parallel** creates a **wide** neural network

Adding neurons in **series** creates a **deep** neural network

Today's powerful neural networks are both **wide** and **deep**



Adding neurons in **parallel** creates a **wide** neural network

Adding neurons in **series** creates a **deep** neural network

Today's powerful neural networks are both **wide** and **deep**

Let us try to implement XOR using a wide and deep neural network

Implement XOR using a deep and wide neural network

Implement XOR using a deep and wide neural network

$$
\begin{aligned}
f(x_1, x_2, \boldsymbol{\theta}) = H\big(\theta_{3,0} & \\
+\theta_{3,1} & \quad \cdot \quad H\big(\theta_{1,0} + x_1\theta_{1,1} + x_2\theta_{1,2}\big) \\
+\theta_{3,2} & \quad \cdot \quad H\big(\theta_{2,0} + x_1\theta_{2,1} + x_2\theta_{2,2}\big)\big)
\end{aligned}
$$

Implement XOR using a deep and wide neural network

$$
\begin{aligned}
f(x_1, x_2, \boldsymbol{\theta}) = H\big(\theta_{3,0} & \\
+\theta_{3,1} \quad & \cdot \quad H\big(\theta_{1,0} + x_1 \theta_{1,1} + x_2 \theta_{1,2}\big) \\
+\theta_{3,2} \quad & \cdot \quad H\big(\theta_{2,0} + x_1 \theta_{2,1} + x_2 \theta_{2,2}\big)\big)
\end{aligned}
$$

$$
\boldsymbol{\theta} = \begin{bmatrix} \theta_{1,0} & \theta_{1,1} & \theta_{1,2} \\ \theta_{2,0} & \theta_{2,1} & \theta_{2,2} \\ \theta_{3,0} & \theta_{3,1} & \theta_{3,2} \end{bmatrix} = \begin{bmatrix} -0.5 & 1 & 1 \\ -1.5 & 1 & 1 \\ -0.5 & 1 & -2 \end{bmatrix}
$$

Implement XOR using a deep and wide neural network

$$\begin{aligned} f(x_1, x_2, \boldsymbol{\theta}) = H\big(\theta_{3,0} & \\ + \theta_{3,1} & \quad \cdot \quad H\big(\theta_{1,0} + x_1 \theta_{1,1} + x_2 \theta_{1,2}\big) \\ + \theta_{3,2} & \quad \cdot \quad H\big(\theta_{2,0} + x_1 \theta_{2,1} + x_2 \theta_{2,2}\big)\big) \end{aligned}$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_{1,0} & \theta_{1,1} & \theta_{1,2} \\ \theta_{2,0} & \theta_{2,1} & \theta_{2,2} \\ \theta_{3,0} & \theta_{3,1} & \theta_{3,2} \end{bmatrix} = \begin{bmatrix} -0.5 & 1 & 1 \\ -1.5 & 1 & 1 \\ -0.5 & 1 & -2 \end{bmatrix}$$

What other functions can we represent using a deep and wide neural network?

What other functions can we represent using a deep and wide neural network?

Consider a one-dimensional arbitrary function $g(x) = y$

What other functions can we represent using a deep and wide neural network?

Consider a one-dimensional arbitrary function $g(x) = y$

We can approximate g using our neural network f

What other functions can we represent using a deep and wide neural network?

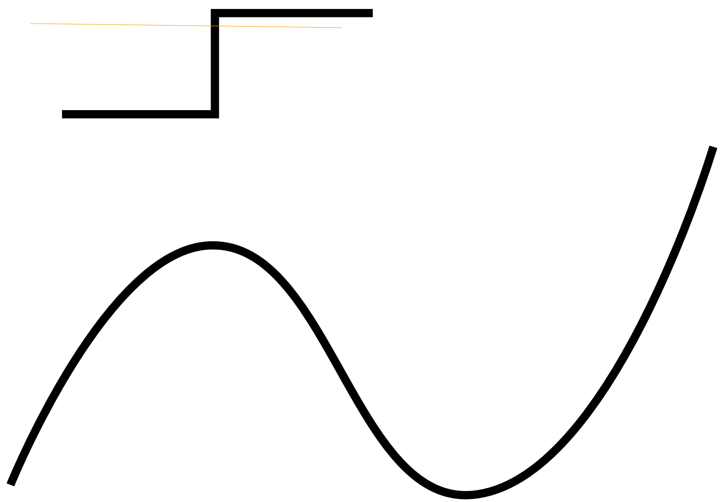
Consider a one-dimensional arbitrary function $g(x) = y$

We can approximate g using our neural network f

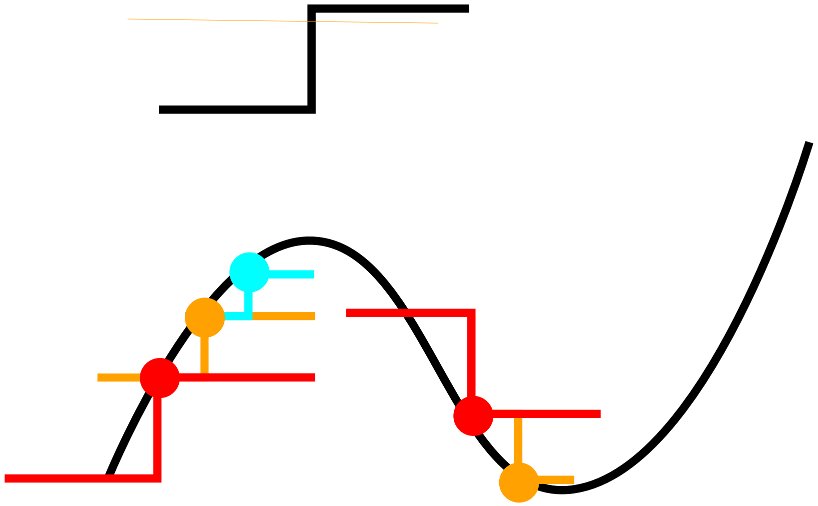
$$
\begin{aligned}
f(x_1, x_2, \boldsymbol{\theta}) = H\big(\theta_{3,0} & & \\
+\theta_{3,1} & \quad \cdot \quad & H\big(\theta_{1,0} + x_1 \theta_{1,1} + x_2 \theta_{1,2}\big) \\
+\theta_{3,2} & \quad \cdot \quad & H\big(\theta_{2,0} + x_1 \theta_{2,1} + x_2 \theta_{2,2}\big)\big)
\end{aligned}
$$

Proof Sketch: Approximate a function $g(x)$ using a linear combination of Heaviside functions

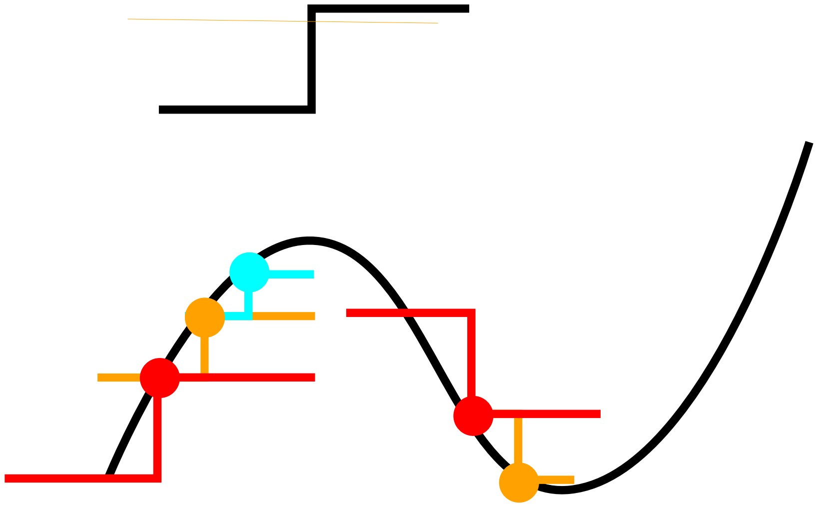
Proof Sketch: Approximate a function $g(x)$ using a linear combination of Heaviside functions



Proof Sketch: Approximate a function $g(x)$ using a linear combination of Heaviside functions



Proof Sketch: Approximate a function $g(x)$ using a linear combination of Heaviside functions



Roughly, $\left[\lim_{n \mapsto \infty} \theta_{2,0} + \theta_{2,1} \sum_{j=1}^{n} \sigma\left(\theta_{1,0} + \theta_{1,j} x\right)\right] = g(x); \quad \forall g$

More formally, a wide and deep neural network is a **universal function approximator**

More formally, a wide and deep neural network is a **universal function approximator**

It can approximate **any** continuous function to precision ε

More formally, a wide and deep neural network is a **universal function approximator**

It can approximate **any** continuous function to precision ε

$$| \; g(\boldsymbol{x}) - f(\boldsymbol{x}, \boldsymbol{\theta}) \; | < \varepsilon$$

More formally, a wide and deep neural network is a **universal function approximator**

It can approximate **any** continuous function to precision ε

$$| \; g(\boldsymbol{x}) - f(\boldsymbol{x}, \boldsymbol{\theta}) \; | < \varepsilon$$

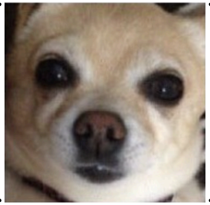
As we increase the width and depth of the network, ε shrinks

More formally, a wide and deep neural network is a **universal function approximator**

It can approximate **any** continuous function to precision ε

$$| \ g(\boldsymbol{x}) - f(\boldsymbol{x}, \boldsymbol{\theta}) \ | < \varepsilon$$

As we increase the width and depth of the network, ε shrinks

$g\Bigg($  $\Bigg) = \text{Dog}$

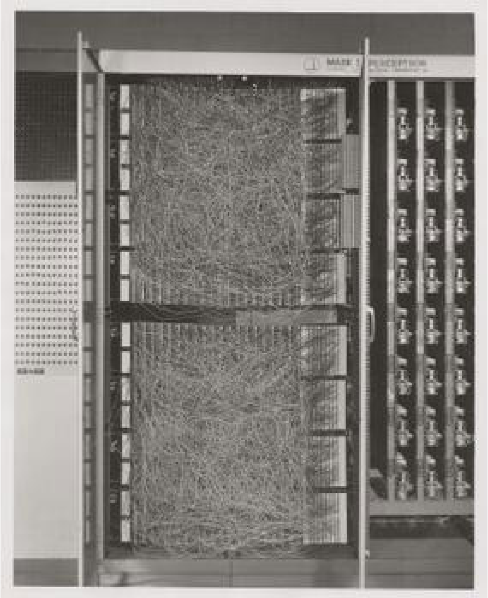
$g\Bigg($  $\Bigg) = \text{Muffin}$

Very powerful finding! The basis of deep learning.

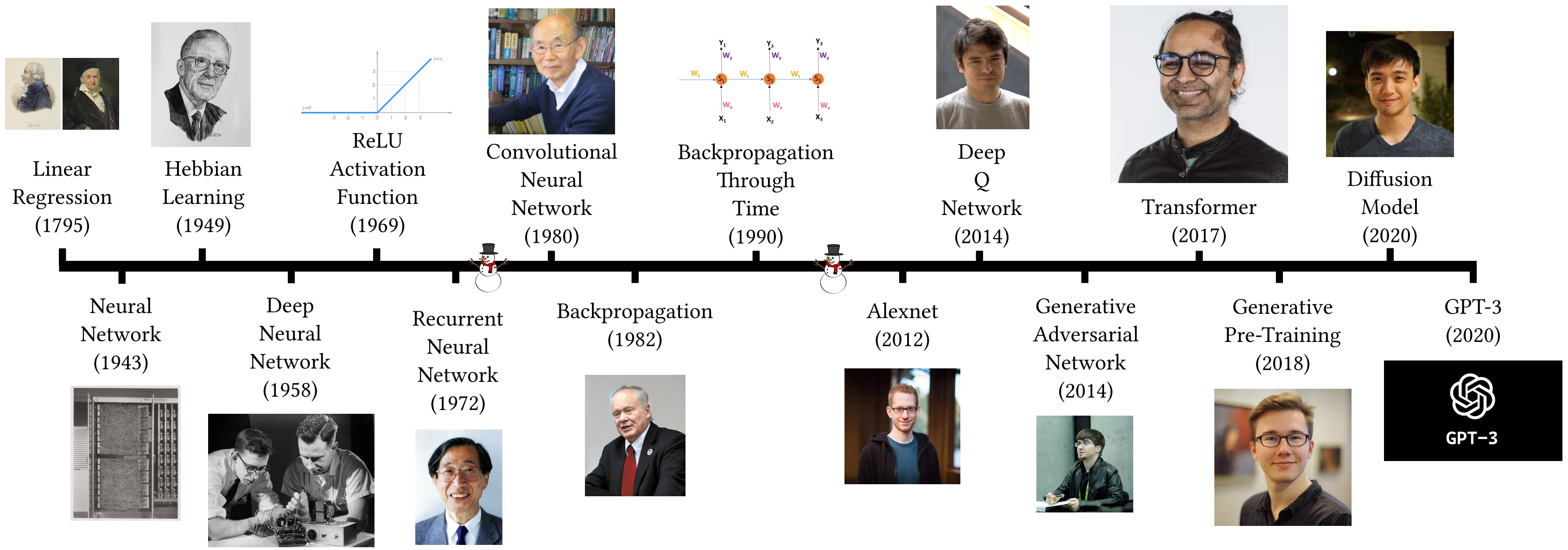
Relax

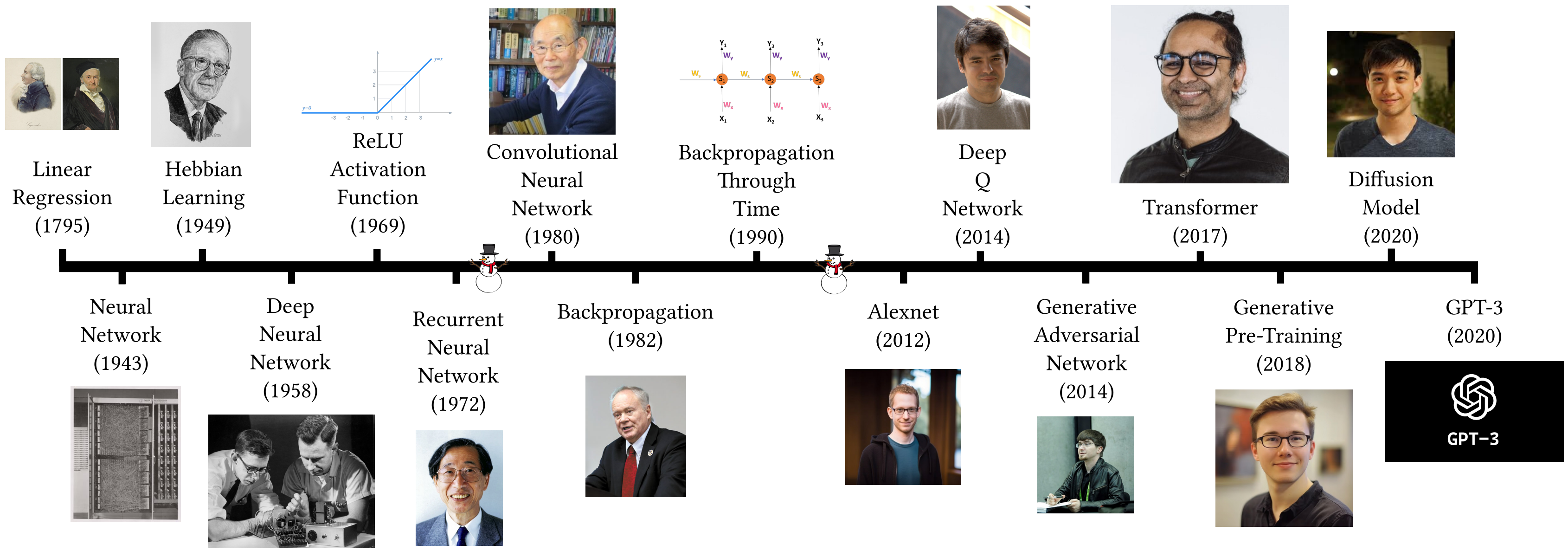
We call this form of a neural network a **feedforward network** or **perceptron** (invented in 1943)

We call this form of a neural network a **feedforward network** or **perceptron** (invented in 1943)

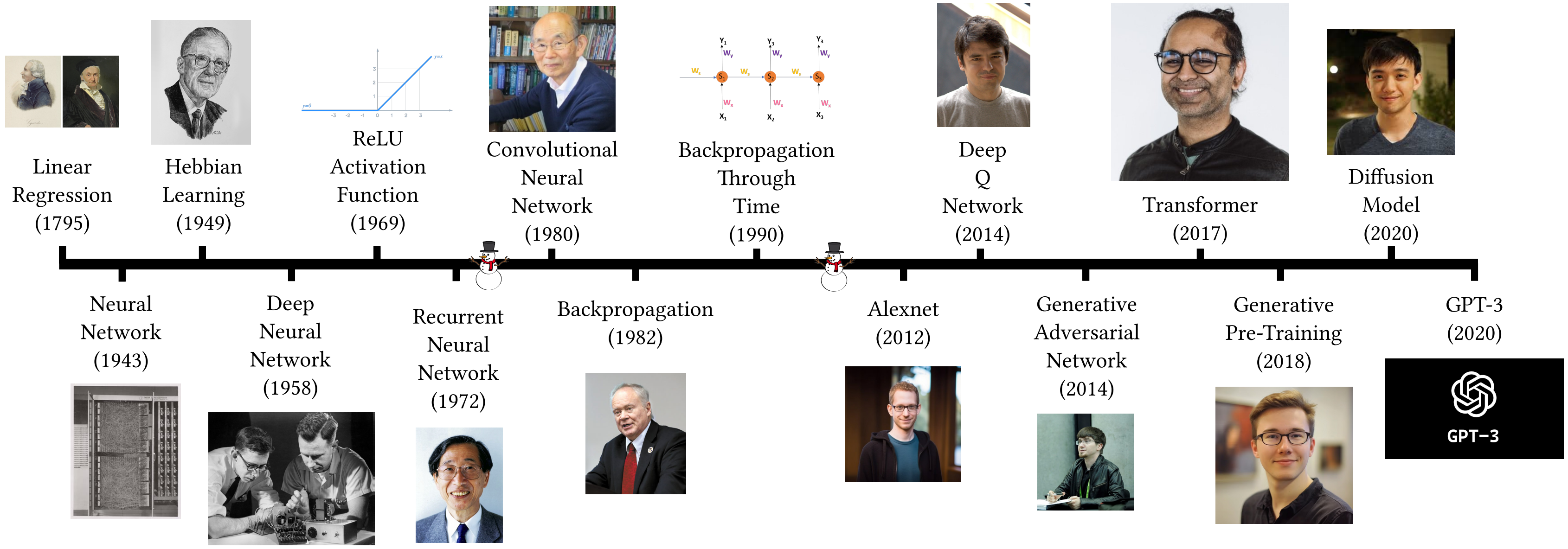


20×20 grid of pixels to process images


Linear Regression (1795)
Neural Network (1943)
Hebbian Learning (1949)
Deep Neural Network (1958)
ReLU Activation Function (1969)
Recurrent Neural Network (1972)
Convolutional Neural Network (1980)
Backpropagation (1982)
Backpropagation Through Time (1990)
Alexnet (2012)
Deep Q Network (2014)
Generative Adversarial Network (2014)
Transformer (2017)
Generative Pre-Training (2018)
Diffusion Model (2020)
GPT-3 (2020)
GPT-3



If the deep neural network was invented in 1958, why did it take 70 years for us to care about deep learning?



If the deep neural network was invented in 1958, why did it take 70 years for us to care about deep learning?

Many small improvements over time eventually made NNs feasible

The neural network we created today is called a feedforward network or perceptron

The neural network we created today is called a feedforward network or perceptron

When the network is deep, we call it a Multi-Layer Perceptron (MLP)

The neural network we created today is called a feedforward network or perceptron

When the network is deep, we call it a Multi-Layer Perceptron (MLP)

We often use the term “layers”, when referring to a specific depth of the neural network

- Four-layer MLP means a neural network with a depth of four