Neural Networks

CISC 7026: Introduction to Deep Learning

University of Macau

Notation Change

Notation change: Previously x_i, y_i referred to data i

Notation Change

Notation change: Previously x_i, y_i referred to data i

From now on, we will write $x_{[i]}$ to refer to data i

Notation Change

Notation change: Previously x_i, y_i referred to data i

From now on, we will write $x_{[i]}$ to refer to data i

x_i will refer to element i of a vector $\boldsymbol{x}$ or matrix $\boldsymbol{X}$

Notation Change

Notation change: Previously x_i, y_i referred to data i

From now on, we will write $x_{[i]}$ to refer to data i

x_i will refer to element i of a vector $\boldsymbol{x}$ or matrix $\boldsymbol{X}$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}, \quad \boldsymbol{X} = \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,n} \end{bmatrix}$$

1. Review
2. Multivariate linear regression
3. Limitations of linear regression
4. History of neural networks
5. Biological neurons
6. Artificial neurons
7. Wide neural networks
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron

1. **Review**
2. Multivariate linear regression
3. Limitations of linear regression
4. History of neural networks
5. Biological neurons
6. Artificial neurons
7. Wide neural networks
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron

Review

Since you are very educated, we focused on how education affects life expectancy

Review

Since you are very educated, we focused on how education affects life expectancy

Studies show a causal effect of education on health

Review

Since you are very educated, we focused on how education affects life expectancy

Studies show a causal effect of education on health

- *The causal effects of education on health outcomes in the UK Biobank.* Davies et al. *Nature Human Behaviour.*

Review

Since you are very educated, we focused on how education affects life expectancy

Studies show a causal effect of education on health

- *The causal effects of education on health outcomes in the UK Biobank.* Davies et al. *Nature Human Behaviour.*
- By staying in school, you are likely to live longer

Review

Since you are very educated, we focused on how education affects life expectancy

Studies show a causal effect of education on health

- *The causal effects of education on health outcomes in the UK Biobank.* Davies et al. *Nature Human Behaviour.*
- By staying in school, you are likely to live longer
- Being rich also helps, but education alone has a **causal** relationship with life expectancy

Review

Task: Given your education, predict your life expectancy

Review

Task: Given your education, predict your life expectancy

$X \in \mathbb{R}_+$: Years in school

Review

Task: Given your education, predict your life expectancy

$X \in \mathbb{R}_+$: Years in school

$Y \in \mathbb{R}_+$: Age of death

Review

Task: Given your education, predict your life expectancy

$X \in \mathbb{R}_+$: Years in school

$Y \in \mathbb{R}_+$: Age of death

$\Theta \in \mathbb{R}^2$: Parameters

Review

Task: Given your education, predict your life expectancy

$X \in \mathbb{R}_+$: Years in school

$Y \in \mathbb{R}_+$: Age of death

$\Theta \in \mathbb{R}^2$: Parameters

$$f : X \times \Theta \mapsto Y$$

Review

Task: Given your education, predict your life expectancy

$X \in \mathbb{R}_+$: Years in school

$Y \in \mathbb{R}_+$: Age of death

$\Theta \in \mathbb{R}^2$: Parameters

$$f : X \times \Theta \mapsto Y$$

Approach: Learn the parameters θ such that

$$f(x, \theta) = y; \quad x \in X, y \in Y$$

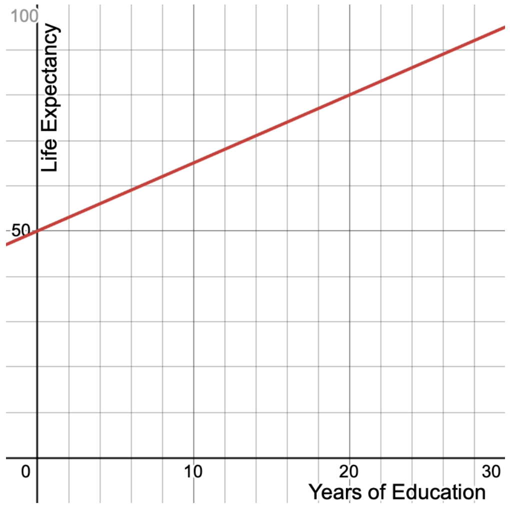
Review

Started with a linear function f

Review

Started with a linear function f

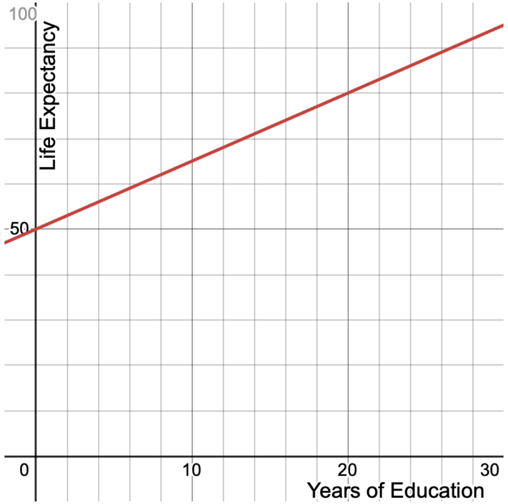
$$f(x, \boldsymbol{\theta}) = f\left(x, \begin{bmatrix} \theta_1 \\ \theta_0 \end{bmatrix}\right) = \theta_1 x + \theta_0$$



Review

Started with a linear function f

$$f(x, \boldsymbol{\theta}) = f\left(x, \begin{bmatrix} \theta_1 \\ \theta_0 \end{bmatrix}\right) = \theta_1 x + \theta_0$$

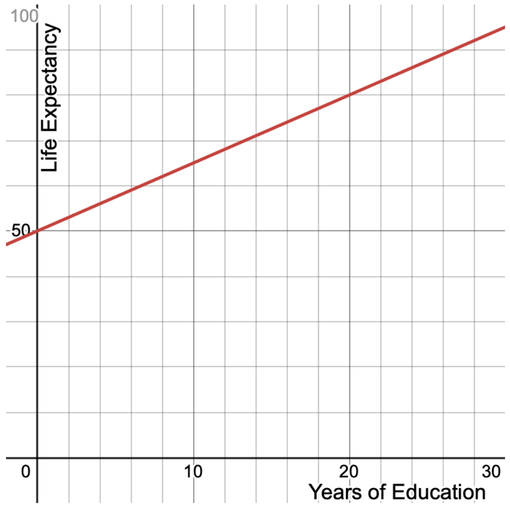


Then, we derived the square error function

Review

Started with a linear function f

$$f(x, \boldsymbol{\theta}) = f\left(x, \begin{bmatrix} \theta_1 \\ \theta_0 \end{bmatrix}\right) = \theta_1 x + \theta_0$$



Then, we derived the square error function

$$\text{error}(f(x, \boldsymbol{\theta}), y) = (f(x, \boldsymbol{\theta}) - y)^2$$

Review

We wrote the loss function for a single datapoint $x_{[i]}, y_{[i]}$ using the square error

$$\mathcal{L}\left(x_{[i]}, y_{[i]}, \boldsymbol{\theta}\right) = \operatorname{error}\left(f\left(x_{[i]}, \boldsymbol{\theta}\right), y_{[i]}\right) = \left(f\left(x_{[i]}, \boldsymbol{\theta}\right) - y_{[i]}\right)^2$$

Review

We wrote the loss function for a single datapoint $x_{[i]}, y_{[i]}$ using the square error

$$\mathcal{L}\left(x_{[i]}, y_{[i]}, \boldsymbol{\theta}\right) = \text{error}\left(f\left(x_{[i]}, \boldsymbol{\theta}\right), y_{[i]}\right) = \left(f\left(x_{[i]}, \boldsymbol{\theta}\right) - y_{[i]}\right)^2$$

But we wanted to learn a model over **all** the data, not a single datapoint

Review

We wrote the loss function for a single datapoint $x_{[i]}, y_{[i]}$ using the square error

$$\mathcal{L}\left(x_{[i]}, y_{[i]}, \boldsymbol{\theta}\right) = \text{error}\left(f\left(x_{[i]}, \boldsymbol{\theta}\right), y_{[i]}\right) = \left(f\left(x_{[i]}, \boldsymbol{\theta}\right) - y_{[i]}\right)^2$$

But we wanted to learn a model over **all** the data, not a single datapoint

We wanted to make **new** predictions, to **generalize**

Review

We wrote the loss function for a single datapoint $x_{[i]}, y_{[i]}$ using the square error

$$\mathcal{L}\left(x_{[i]}, y_{[i]}, \boldsymbol{\theta}\right) = \mathrm{error}\left(f\left(x_{[i]}, \boldsymbol{\theta}\right), y_{[i]}\right) = \left(f\left(x_{[i]}, \boldsymbol{\theta}\right) - y_{[i]}\right)^2$$

But we wanted to learn a model over **all** the data, not a single datapoint

We wanted to make **new** predictions, to **generalize**

$$\boldsymbol{x} = \begin{bmatrix} x_{[1]} & x_{[2]} & \cdots & x_{[n]} \end{bmatrix}^\top, \boldsymbol{y} = \begin{bmatrix} y_{[1]} & y_{[2]} & \cdots & y_{[n]} \end{bmatrix}^\top$$

Review

We wrote the loss function for a single datapoint $x_{[i]}, y_{[i]}$ using the square error

$$\mathcal{L}\left(x_{[i]}, y_{[i]}, \boldsymbol{\theta}\right) = \text{error}\left(f\left(x_{[i]}, \boldsymbol{\theta}\right), y_{[i]}\right) = \left(f\left(x_{[i]}, \boldsymbol{\theta}\right) - y_{[i]}\right)^2$$

But we wanted to learn a model over **all** the data, not a single datapoint

We wanted to make **new** predictions, to **generalize**

$$\boldsymbol{x} = \begin{bmatrix} x_{[1]} & x_{[2]} & \cdots & x_{[n]} \end{bmatrix}^\top, \boldsymbol{y} = \begin{bmatrix} y_{[1]} & y_{[2]} & \cdots & y_{[n]} \end{bmatrix}^\top$$

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \text{error}\left(f\left(x_{[i]}, \boldsymbol{\theta}\right), y_{[i]}\right) = \sum_{i=1}^{n} \left(f\left(x_{[i]}, \boldsymbol{\theta}\right) - y_{[i]}\right)^2$$

Review

Our objective was to find the parameters that minimized the loss function over the dataset

Review

Our objective was to find the parameters that minimized the loss function over the dataset

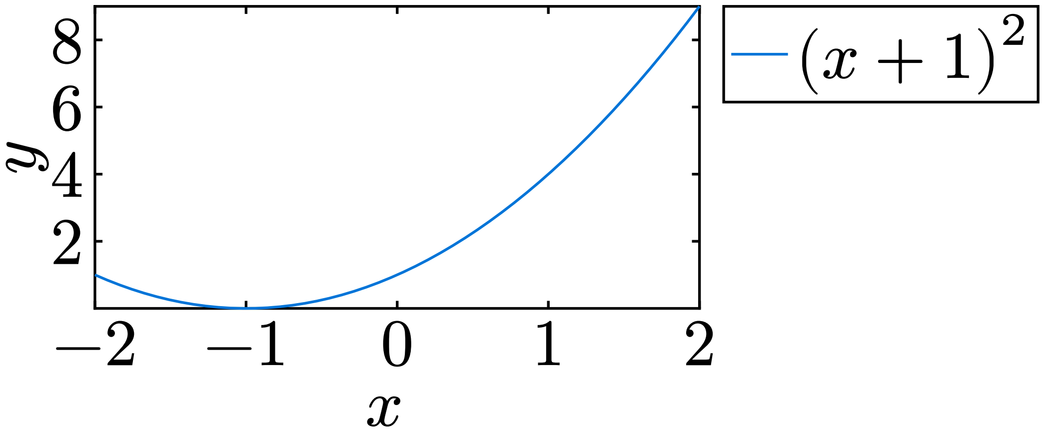
We introduced the arg min operator

Review

Our objective was to find the parameters that minimized the loss function over the dataset

We introduced the arg min operator

$$f(x) = (x + 1)^2$$

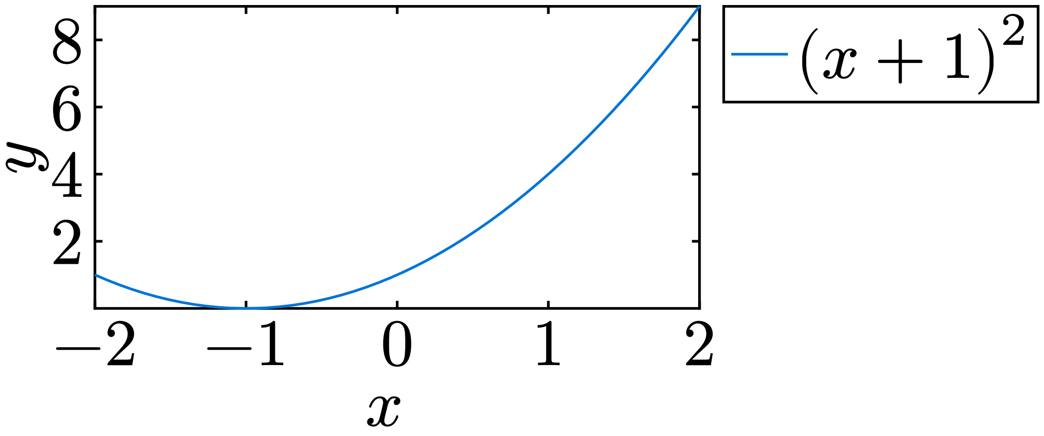


Review

Our objective was to find the parameters that minimized the loss function over the dataset

We introduced the arg min operator

$$f(x) = (x+1)^2$$



$$\arg\min_x f(x) = -1$$

Review

With the $\arg\min$ operator, we formally wrote our optimization objective

Review

With the arg min operator, we formally wrote our optimization objective

$$\underset{\boldsymbol{\theta}}{\arg\min}\, \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\arg\min} \sum_{i=1}^{n} \text{error}\Big(f\big(x_{[i]}, \boldsymbol{\theta}\big), y_{[i]}\Big)$$

$$= \underset{\boldsymbol{\theta}}{\arg\min} \sum_{i=1}^{n} \Big(f\big(x_{[i]}, \boldsymbol{\theta}\big) - y_{[i]}\Big)^2$$

Review

We defined the design matrix $\boldsymbol{X}_D$

Review

We defined the design matrix $\boldsymbol{X}_D$

$$\boldsymbol{X}_D = [\boldsymbol{x} \;\; \boldsymbol{1}] = \begin{bmatrix} x_{[1]} & 1 \\ x_{[2]} & 1 \\ \vdots & \vdots \\ x_{[n]} & 1 \end{bmatrix}$$

Review

We defined the design matrix $\boldsymbol{X}_D$

$$\boldsymbol{X}_D = [\boldsymbol{x} \;\; \mathbf{1}] = \begin{bmatrix} x_{[1]} & 1 \\ x_{[2]} & 1 \\ \vdots & \vdots \\ x_{[n]} & 1 \end{bmatrix}$$

We use the design matrix to find an **analytical** solution to the optimization objective

Review

We defined the design matrix $\boldsymbol{X}_D$

$$\boldsymbol{X}_D = [\boldsymbol{x} \;\; \mathbf{1}] = \begin{bmatrix} x_{[1]} & 1 \\ x_{[2]} & 1 \\ \vdots & \vdots \\ x_{[n]} & 1 \end{bmatrix}$$

We use the design matrix to find an **analytical** solution to the optimization objective

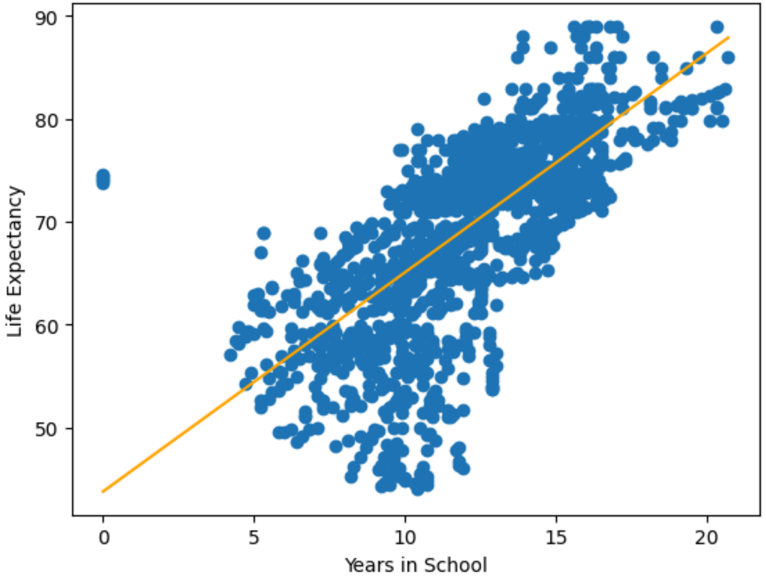
$$\boldsymbol{\theta} = \left(\boldsymbol{X}_D^\top \boldsymbol{X}_D\right)^{-1} \boldsymbol{X}_D^\top \boldsymbol{y}$$

Review

With this analytical solution, we were able to learn a linear model

Review

With this analytical solution, we were able to learn a linear model



Review

Then, we used a trick to extend linear regression to nonlinear models

Review

Then, we used a trick to extend linear regression to nonlinear models

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1]} & 1 \\ x_{[2]} & 1 \\ \vdots & \vdots \\ x_{[n]} & 1 \end{bmatrix} \Rightarrow \boldsymbol{X}_D = \begin{bmatrix} \log\left(1 + x_{[1]}\right) & 1 \\ \log\left(1 + x_{[2]}\right) & 1 \\ \vdots & \vdots \\ \log\left(1 + x_{[n]}\right) & 1 \end{bmatrix}$$

Review

We extended to polynomials, which are **universal function approximators**

Review

We extended to polynomials, which are **universal function approximators**

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1]} & 1 \\ x_{[2]} & 1 \\ \vdots & \vdots \\ x_{[n]} & 1 \end{bmatrix} \Rightarrow \boldsymbol{X}_D = \begin{bmatrix} x_{[1]}^m & x_{[1]}^{m-1} & \dots & x_{[1]} & 1 \\ x_{[2]}^m & x_{[2]}^{m-1} & \dots & x_{[2]} & 1 \\ \vdots & \vdots & \ddots & & \\ x_{[n]}^m & x_{[n]}^{m-1} & \dots & x_{[n]} & 1 \end{bmatrix}$$

Review

We extended to polynomials, which are **universal function approximators**

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1]} & 1 \\ x_{[2]} & 1 \\ \vdots & \vdots \\ x_{[n]} & 1 \end{bmatrix} \Rightarrow \boldsymbol{X}_D = \begin{bmatrix} x_{[1]}^m & x_{[1]}^{m-1} & \dots & x_{[1]} & 1 \\ x_{[2]}^m & x_{[2]}^{m-1} & \dots & x_{[2]} & 1 \\ \vdots & \vdots & \ddots & & \\ x_{[n]}^m & x_{[n]}^{m-1} & \dots & x_{[n]} & 1 \end{bmatrix}$$

$$f : X \times \Theta \mapsto \mathbb{R}$$

Review

We extended to polynomials, which are **universal function approximators**

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1]} & 1 \\ x_{[2]} & 1 \\ \vdots & \vdots \\ x_{[n]} & 1 \end{bmatrix} \Rightarrow \boldsymbol{X}_D = \begin{bmatrix} x_{[1]}^m & x_{[1]}^{m-1} & \dots & x_{[1]} & 1 \\ x_{[2]}^m & x_{[2]}^{m-1} & \dots & x_{[2]} & 1 \\ \vdots & \vdots & \ddots & & \\ x_{[n]}^m & x_{[n]}^{m-1} & \dots & x_{[n]} & 1 \end{bmatrix}$$

$$f : X \times \Theta \mapsto \mathbb{R}$$

$$\Theta \in \mathbb{R}^2 \Rightarrow \Theta \in \mathbb{R}^m$$

Review

Finally, we discussed overfitting

Review

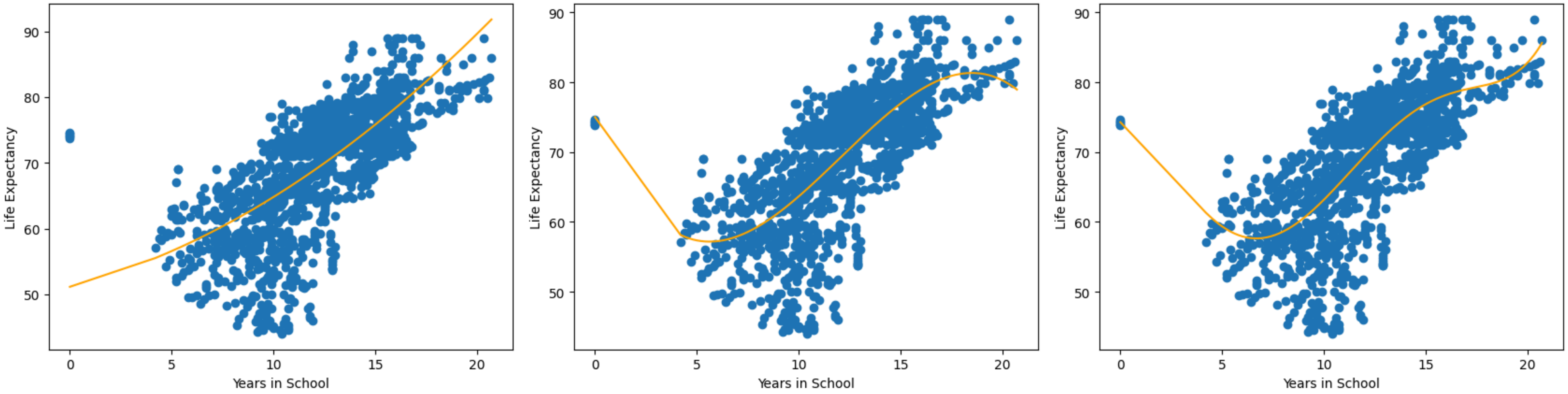
Finally, we discussed overfitting

$$f(x, \boldsymbol{\theta}) = \theta_n x^m + \theta_{m-1} x^{m-1}, ..., \theta_1 + x^1 + \theta_0$$

Review

Finally, we discussed overfitting

$$f(x, \boldsymbol{\theta}) = \theta_n x^m + \theta_{m-1} x^{m-1}, ..., \theta_1 + x^1 + \theta_0$$



$m = 2$ $m = 3$ $m = 5$

Review

We care about **generalization** in machine learning

Review

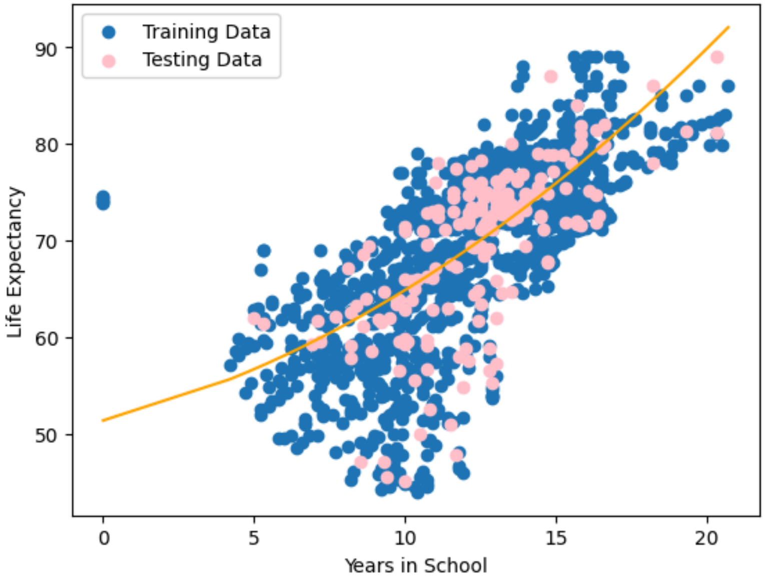
We care about **generalization** in machine learning

So we should always split our dataset into a training dataset and a testing dataset

Review

We care about **generalization** in machine learning

So we should always split our dataset into a training dataset and a testing dataset



1. **Review**
2. Multivariate linear regression
3. Limitations of linear regression
4. History of neural networks
5. Biological neurons
6. Artificial neurons
7. Wide neural networks
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron

1. Review
2. **Multivariate linear regression**
3. Limitations of linear regression
4. History of neural networks
5. Biological neurons
6. Artificial neurons
7. Wide neural networks
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron

Last time, we assumed a single-input system

Last time, we assumed a single-input system

Years of education: $X \in \mathbb{R}$

Last time, we assumed a single-input system

Years of education: $X \in \mathbb{R}$

But sometimes we want to consider multiple input dimensions

Last time, we assumed a single-input system

Years of education: $X \in \mathbb{R}$

But sometimes we want to consider multiple input dimensions

Years of education, BMI, GDP: $X \in \mathbb{R}^3$

Last time, we assumed a single-input system

Years of education: $X \in \mathbb{R}$

But sometimes we want to consider multiple input dimensions

Years of education, BMI, GDP: $X \in \mathbb{R}^3$

We can solve these problems using linear regression too

For multivariate problems, we will define the input dimension as d_x

For multivariate problems, we will define the input dimension as d_x

$$\boldsymbol{x} \in X; \quad X \in \mathbb{R}^{d_x}$$

For multivariate problems, we will define the input dimension as d_x

$$\boldsymbol{x} \in X; \quad X \in \mathbb{R}^{d_x}$$

We will write the vectors as

$$\boldsymbol{x}_{[i]} = \begin{bmatrix} x_{[i],1} \\ x_{[i],2} \\ \vdots \\ x_{[i],d_x} \end{bmatrix}$$

For multivariate problems, we will define the input dimension as d_x

$$\boldsymbol{x} \in X; \quad X \in \mathbb{R}^{d_x}$$

We will write the vectors as

$$\boldsymbol{x}_{[i]} = \begin{bmatrix} x_{[i],1} \\ x_{[i],2} \\ \vdots \\ x_{[i],d_x} \end{bmatrix}$$

$x_{[i],1}$ refers to the first dimension of training data i

The design matrix for a **multivariate** linear system is

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1],d_x} & x_{[1],d_x-1} & \dots & x_{[1],1} & 1 \\ x_{[2],d_x} & x_{[2],d_x-1} & \dots & x_{[2],1} & 1 \\ \vdots & \vdots & \ddots & \vdots & \\ x_{[n],d_x} & x_{[n],d_x-1} & \dots & x_{[n],1} & 1 \end{bmatrix}$$

The design matrix for a **multivariate** linear system is

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1],d_x} & x_{[1],d_x-1} & \dots & x_{[1],1} & 1 \\ x_{[2],d_x} & x_{[2],d_x-1} & \dots & x_{[2],1} & 1 \\ \vdots & \vdots & \ddots & \vdots & \\ x_{[n],d_x} & x_{[n],d_x-1} & \dots & x_{[n],1} & 1 \end{bmatrix}$$

Remember $x_{[n],d_x}$ refers to dimension d_x of training data n

The design matrix for a **multivariate** linear system is

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1],d_x} & x_{[1],d_x-1} & \dots & x_{[1],1} & 1 \\ x_{[2],d_x} & x_{[2],d_x-1} & \dots & x_{[2],1} & 1 \\ \vdots & \vdots & \ddots & \vdots & \\ x_{[n],d_x} & x_{[n],d_x-1} & \dots & x_{[n],1} & 1 \end{bmatrix}$$

Remember $x_{[n],d_x}$ refers to dimension d_x of training data n

The solution is the same as before

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

Agenda

1. Review
2. **Multivariate linear regression**
3. Limitations of linear regression
4. History of neural networks
5. Biological neurons
6. Artificial neurons
7. Wide neural networks
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron

Agenda

1. Review
2. Multivariate linear regression
3. **Limitations of linear regression**
4. History of neural networks
5. Biological neurons
6. Artificial neurons
7. Wide neural networks
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron

Linear models are useful for certain problems

Linear models are useful for certain problems

1. Analytical solution

Linear models are useful for certain problems

1. Analytical solution
2. Low data requirement

Linear models are useful for certain problems

1. Analytical solution
2. Low data requirement

But issues arise with other problems

Linear models are useful for certain problems

1. Analytical solution
2. Low data requirement

But issues arise with other problems

1. Poor scalability

Linear models are useful for certain problems

1. Analytical solution
2. Low data requirement

But issues arise with other problems

1. Poor scalability
2. Polynomials do not generalize well

Issues with very complex problems

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

So far, we have seen:

So far, we have seen:

One-dimensional polynomial functions

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1]}^m & x_{[1]}^{m-1} & \dots & x_{[1]} & 1 \\ x_{[2]}^m & x_{[2]}^{m-1} & \dots & x_{[2]} & 1 \\ \vdots & \vdots & \ddots & & \\ x_{[n]}^m & x_{[n]}^{m-1} & \dots & x_{[n]} & 1 \end{bmatrix}$$

So far, we have seen:

One-dimensional polynomial functions

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1]}^m & x_{[1]}^{m-1} & \dots & x_{[1]} & 1 \\ x_{[2]}^m & x_{[2]}^{m-1} & \dots & x_{[2]} & 1 \\ \vdots & \vdots & \ddots & & \\ x_{[n]}^m & x_{[n]}^{m-1} & \dots & x_{[n]} & 1 \end{bmatrix}$$

Multi-dimensional linear functions

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1],d_x} & x_{[1],d_x-1} & \dots & 1 \\ x_{[2],d_x} & x_{[2],d_x-1} & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ x_{[n],d_x} & x_{[n],d_x-1} & \dots & 1 \end{bmatrix}$$

So far, we have seen:

One-dimensional polynomial functions

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1]}^m & x_{[1]}^{m-1} & \dots & x_{[1]} & 1 \\ x_{[2]}^m & x_{[2]}^{m-1} & \dots & x_{[2]} & 1 \\ \vdots & \vdots & \ddots & & \\ x_{[n]}^m & x_{[n]}^{m-1} & \dots & x_{[n]} & 1 \end{bmatrix}$$

Multi-dimensional linear functions

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1],d_x} & x_{[1],d_x-1} & \dots & 1 \\ x_{[2],d_x} & x_{[2],d_x-1} & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ x_{[n],d_x} & x_{[n],d_x-1} & \dots & 1 \end{bmatrix}$$

Combine them to create multi-dimensional polynomial functions

So far, we have seen:

One-dimensional polynomial functions

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1]}^m & x_{[1]}^{m-1} & \dots & x_{[1]} & 1 \\ x_{[2]}^m & x_{[2]}^{m-1} & \dots & x_{[2]} & 1 \\ \vdots & \vdots & \ddots & & \\ x_{[n]}^m & x_{[n]}^{m-1} & \dots & x_{[n]} & 1 \end{bmatrix}$$

Multi-dimensional linear functions

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1],d_x} & x_{[1],d_x-1} & \dots & 1 \\ x_{[2],d_x} & x_{[2],d_x-1} & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ x_{[n],d_x} & x_{[n],d_x-1} & \dots & 1 \end{bmatrix}$$

Combine them to create multi-dimensional polynomial functions

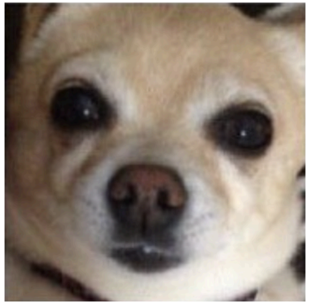
Let us do an example

Let us do an example

Task: predict how many ♥ a photo gets on social media

Let us do an example

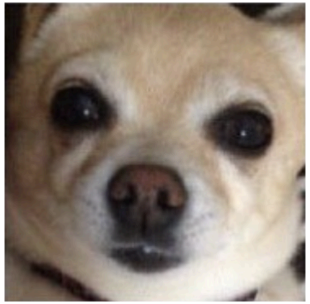
Task: predict how many ♥ a photo gets on social media



$$X \in \mathbb{Z}_+^{256 \times 256}$$

Let us do an example

Task: predict how many ♥ a photo gets on social media

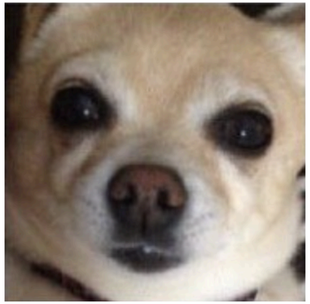


$$X \in \mathbb{Z}_+^{256 \times 256}$$

In this case, $d_x = 65,536$

Let us do an example

Task: predict how many ♥ a photo gets on social media



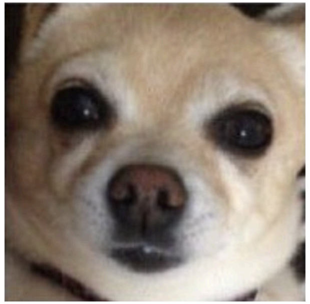
$$X \in \mathbb{Z}_+^{256 \times 256}$$

In this case, $d_x = 65,536$

This task is highly nonlinear, so we will use a polynomial model of order $m = 20$

Let us do an example

Task: predict how many ♥ a photo gets on social media



$$X \in \mathbb{Z}_+^{256 \times 256}$$

In this case, $d_x = 65,536$

This task is highly nonlinear, so we will use a polynomial model of order $m = 20$

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1], d_x}^m & \dots & x_{[1], 1}^m & \dots & x_{[1], d_x}^{m-1} & \dots & x_{[1], 1}^{m-1} & \dots & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{[n], d_x}^m & \dots & x_{[n], 1}^m & \dots & x_{[n], d_x}^{m-1} & \dots & x_{[n], 1}^{m-1} & \dots & \dots & 1 \\ x_{[1], d_x}^m & x_{[1], d_x}^{m-1} x_{[1], d_x - 1} & \dots & \dots & \dots & \dots & \dots & \dots & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

$$
\boldsymbol{X}_D = \begin{bmatrix}
x^m_{[1],d_x} & \cdots & x^m_{[1],1} & \cdots & x^{m-1}_{[1],d_x} & \cdots & x^{m-1}_{[1],1} & \cdots & \cdots & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
x^m_{[n],d_x} & \cdots & x^m_{[n],1} & \cdots & x^{m-1}_{[n],d_x} & \cdots & x^{m-1}_{[n],1} & \cdots & \cdots & 1 \\
x^m_{[1],d_x} & x^{m-1}_{[1],d_x} x_{[1],d_x-1} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{bmatrix}
$$

Question: How many rows in this matrix?

$$\boldsymbol{X}_D = \begin{bmatrix} x_{[1], d_x}^m & \dots & x_{[1], 1}^m & \dots & x_{[1], d_x}^{m-1} & \dots & x_{[1], 1}^{m-1} & \dots & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{[n], d_x}^m & \dots & x_{[n], 1}^m & \dots & x_{[n], d_x}^{m-1} & \dots & x_{[n], 1}^{m-1} & \dots & \dots & 1 \\ x_{[1], d_x}^m & x_{[1], d_x}^{m-1} x_{[1], d_x - 1} & \dots & \dots & \dots & \dots & \dots & \dots & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Question: How many rows in this matrix?

Hint: $d_x = 2, m = 3$: $x^3 + y^3 + x^2 y + y^2 x + xy + x + y + 1$

$$
\boldsymbol{X}_D = \begin{bmatrix}
x_{[1],d_x}^m & \cdots & x_{[1],1}^m & \cdots & x_{[1],d_x}^{m-1} & \cdots & x_{[1],1}^{m-1} & \cdots & \cdots & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
x_{[n],d_x}^m & \cdots & x_{[n],1}^m & \cdots & x_{[n],d_x}^{m-1} & \cdots & x_{[n],1}^{m-1} & \cdots & \cdots & 1 \\
x_{[1],d_x}^m & x_{[1],d_x}^{m-1} x_{[1],d_x-1} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{bmatrix}
$$

Question: How many rows in this matrix?

Hint: $d_x = 2, m = 3$: $x^3 + y^3 + x^2 y + y^2 x + xy + x + y + 1$

Answer: $(d_x)^m + 1 = 65536^{20} + 1 \approx 10^{96}$

To find $\boldsymbol{\theta}$, we must invert $\boldsymbol{X}_D^\top \boldsymbol{X}_D$

To find $\boldsymbol{\theta}$, we must invert $\boldsymbol{X}_D^\top \boldsymbol{X}_D$

$\boldsymbol{X}_D^\top \boldsymbol{X}_D : 10^{96} \times 10^{96}$

To find $\boldsymbol{\theta}$, we must invert $\boldsymbol{X}_D^\top \boldsymbol{X}_D$

$\boldsymbol{X}_D^\top \boldsymbol{X}_D : 10^{96} \times 10^{96}$

Question: What is the largest matrix ever inverted?

To find $\boldsymbol{\theta}$, we must invert $\boldsymbol{X}_D^\top \boldsymbol{X}_D$

$\boldsymbol{X}_D^\top \boldsymbol{X}_D : 10^{96} \times 10^{96}$

Question: What is the largest matrix ever inverted?

Answer: $10^6 \times 10^6$

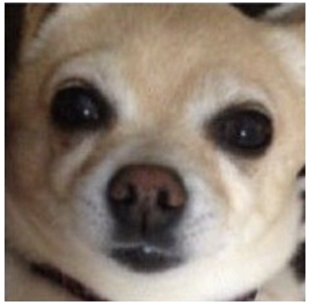
To find $\boldsymbol{\theta}$, we must invert $\boldsymbol{X}_D^\top \boldsymbol{X}_D$

$\boldsymbol{X}_D^\top \boldsymbol{X}_D : 10^{96} \times 10^{96}$

Question: What is the largest matrix ever inverted?

Answer: $10^6 \times 10^6$

We cannot predict how many ♥
the picture will get



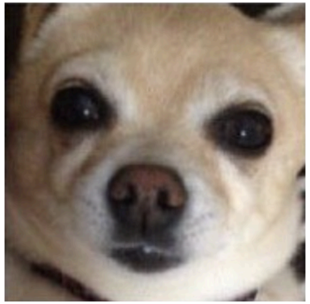
To find $\boldsymbol{\theta}$, we must invert $\boldsymbol{X}_D^\top \boldsymbol{X}_D$

$\boldsymbol{X}_D^\top \boldsymbol{X}_D : 10^{96} \times 10^{96}$

Question: What is the largest matrix ever inverted?

Answer: $10^6 \times 10^6$

We cannot predict how many ♥
the picture will get



Polynomial regression does not scale to large inputs

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**

What happens to polynomials outside of the support (dataset)?

What happens to polynomials outside of the support (dataset)?

We will evaluate polynomials in the limit

What happens to polynomials outside of the support (dataset)?

We will evaluate polynomials in the limit

$$\lim_{x \to \infty} \theta_m x^m + \theta_{m-1} x^{m-1} + \ldots$$

Equation of a polynomial

What happens to polynomials outside of the support (dataset)?

We will evaluate polynomials in the limit

$$\lim_{x \to \infty} \theta_m x^m + \theta_{m-1} x^{m-1} + \ldots$$

Equation of a polynomial

$$\lim_{x \to \infty} x^m \left(\theta_m + \frac{\theta_{m-1}}{x} + \ldots \right)$$

Factor out x^m

What happens to polynomials outside of the support (dataset)?

We will evaluate polynomials in the limit

$$\lim_{x \to \infty} \theta_m x^m + \theta_{m-1} x^{m-1} + \ldots$$ Equation of a polynomial

$$\lim_{x \to \infty} x^m \left(\theta_m + \frac{\theta_{m-1}}{x} + \ldots \right)$$ Factor out x^m

$$\lim_{x \to \infty} x^m \cdot \lim_{x \to \infty} \left(\theta_m + \frac{\theta_{m-1}}{x} + \ldots \right)$$ Split the limit (limit of products)

$$\lim_{x \to \infty} x^m \cdot \lim_{x \to \infty} \left(\theta_m + \frac{\theta_{m-1}}{x} + \ldots \right)$$

Split the limit (limit of products)

$$\left(\lim_{x \to \infty} x^m \right) \cdot (\theta_m + 0 + \ldots)$$

Evaluate right limit

$$\lim_{x \to \infty} x^m \cdot \lim_{x \to \infty} \left(\theta_m + \frac{\theta_{m-1}}{x} + \ldots \right)$$

Split the limit (limit of products)

$$\left(\lim_{x \to \infty} x^m \right) \cdot (\theta_m + 0 + \ldots)$$

Evaluate right limit

$$\theta_m \lim_{x \to \infty} x^m$$

Rewrite

$$\lim_{x \to \infty} x^m \cdot \lim_{x \to \infty} \left(\theta_m + \frac{\theta_{m-1}}{x} + \ldots \right)$$

Split the limit (limit of products)

$$\left(\lim_{x \to \infty} x^m \right) \cdot (\theta_m + 0 + \ldots)$$

Evaluate right limit

$$\theta_m \lim_{x \to \infty} x^m$$

Rewrite

$$\theta_m \lim_{x \to \infty} x^m = \infty$$

If $\theta_m > 0$

$$\lim_{x \to \infty} x^m \cdot \lim_{x \to \infty} \left(\theta_m + \frac{\theta_{m-1}}{x} + \ldots \right)$$

Split the limit (limit of products)

$$\left(\lim_{x \to \infty} x^m \right) \cdot (\theta_m + 0 + \ldots)$$

Evaluate right limit

$$\theta_m \lim_{x \to \infty} x^m$$

Rewrite

$$\theta_m \lim_{x \to \infty} x^m = \infty$$

If $\theta_m > 0$

$$\theta_m \lim_{x \to \infty} x^m = -\infty$$

If $\theta_m < 0$

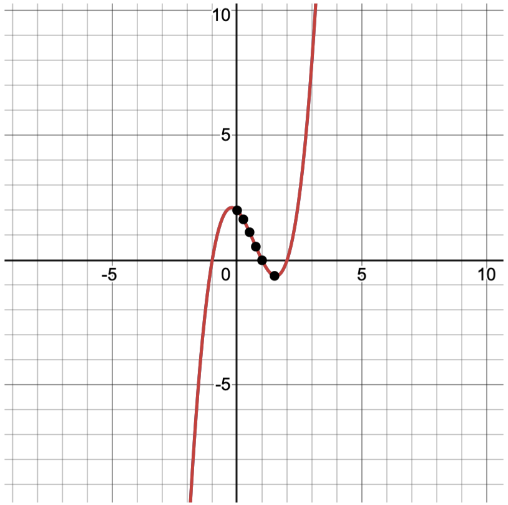
Polynomials quickly tend towards $-\infty, \infty$ outside of the support

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

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

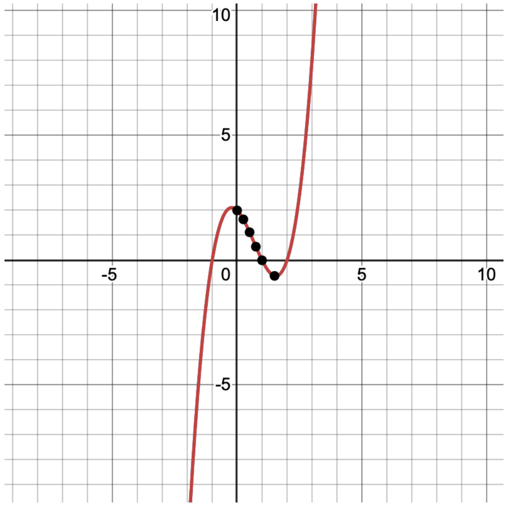
Polynomials quickly tend towards $-\infty, \infty$ outside of the support

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



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

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



Remember, to predict new data we want our functions to generalize

Linear regression has issues

Linear regression has issues

1. Poor scalability

Linear regression has issues

1. Poor scalability
2. Polynomials do not generalize well

1. Review
2. Multivariate linear regression
3. **Limitations of linear regression**
4. History of neural networks
5. Biological neurons
6. Artificial neurons
7. Wide neural networks
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron

1. Review
2. Multivariate linear regression
3. Limitations of linear regression
4. **History of neural networks**
5. Biological neurons
6. Artificial neurons
7. Wide neural networks
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron

Can we improve upon linear regression?

Can we improve upon linear regression?

Yes, with neural networks

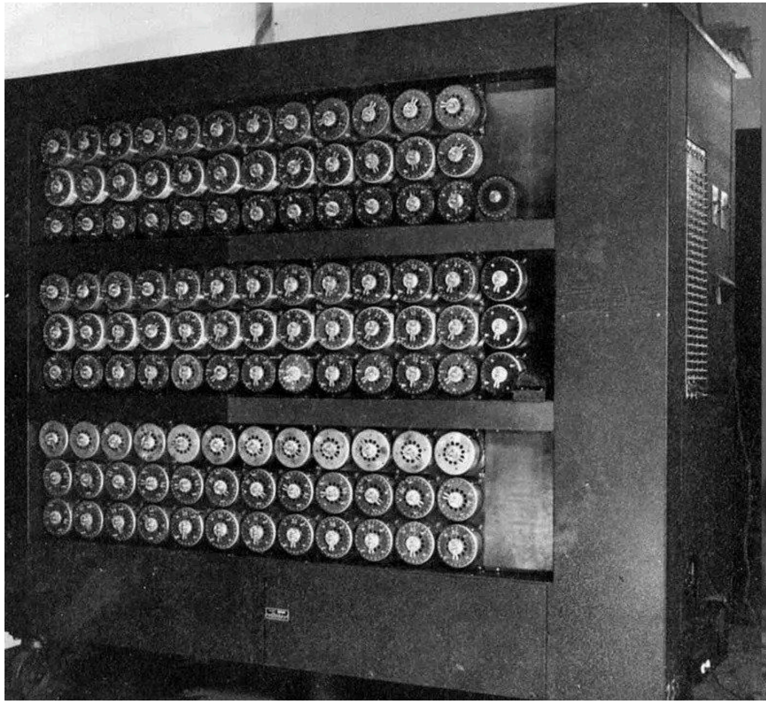
In 1939-1945, there was a World War

In 1939-1945, there was a World War

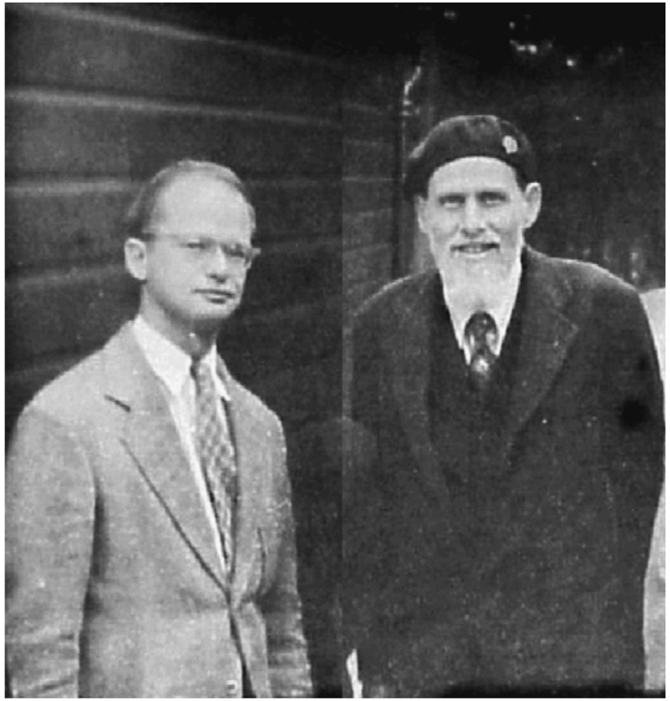
Militaries invested funding for research, and invented the computer

In 1939-1945, there was a World War

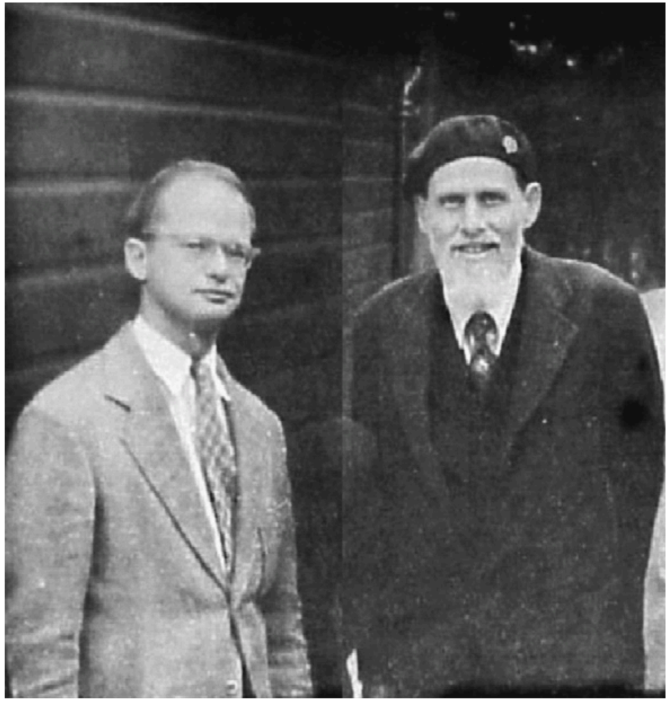
Militaries invested funding for research, and invented the computer

Meanwhile, a neuroscientist and mathematician (McCullough and Pitts) were trying to understand the human brain

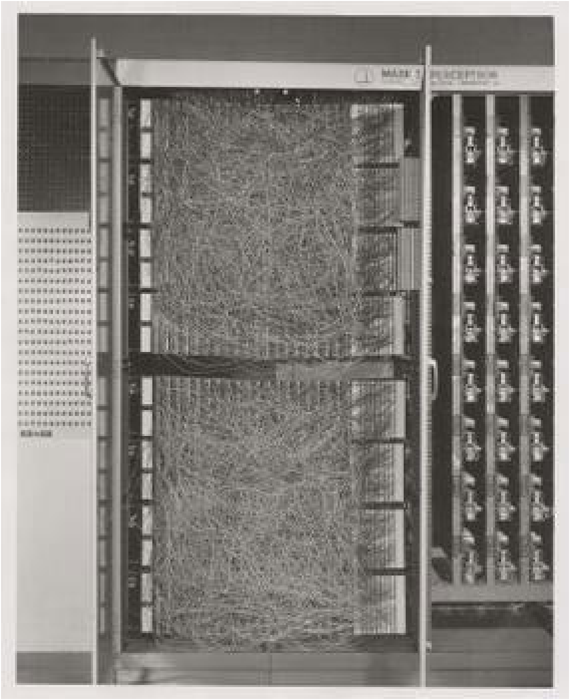


Meanwhile, a neuroscientist and mathematician (McCullough and Pitts) were trying to understand the human brain



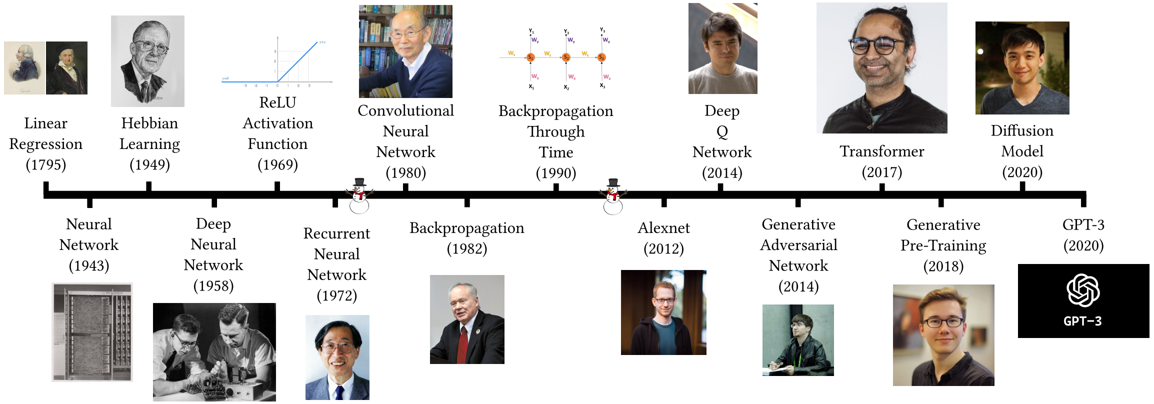
They designed the theory for the first neural network

A few years later, Rosenblatt implemented this neural network using a new invention – the computer

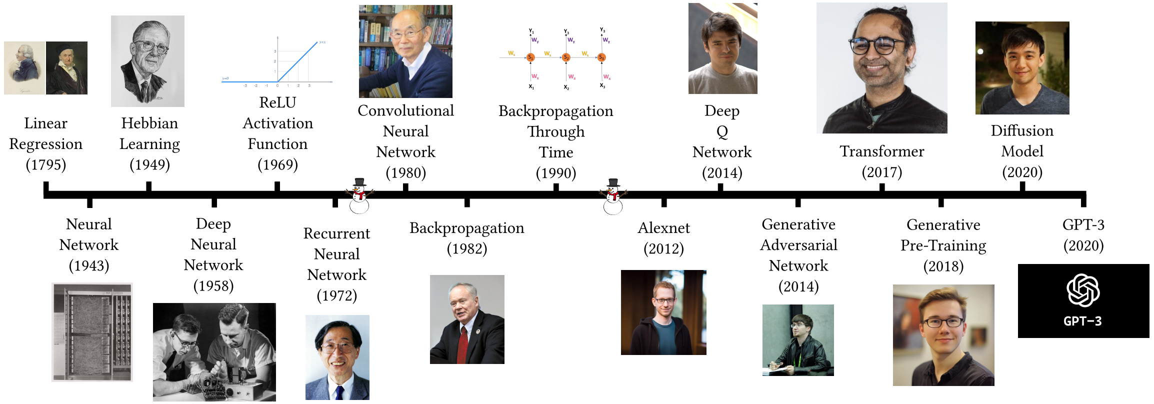


Through advances in theory and hardware, neural networks became slightly better

Through advances in theory and hardware, neural networks became slightly better



Through advances in theory and hardware, neural networks became slightly better



Around 2012, these improvements culminated in neural networks that perform like humans

So what is a neural network?

So what is a neural network?

It is a function, inspired by how the brain works

So what is a neural network?

It is a function, inspired by how the brain works

$$f : X \times \Theta \mapsto Y$$

Brains and neural networks rely on **neurons**

Brains and neural networks rely on **neurons**

Brain: Biological neurons $\rightarrow$ Biological neural network

Brains and neural networks rely on **neurons**

Brain: Biological neurons $\rightarrow$ Biological neural network

Computer: Artificial neurons $\rightarrow$ Artificial neural network

Brains and neural networks rely on **neurons**

Brain: Biological neurons $\rightarrow$ Biological neural network

Computer: Artificial neurons $\rightarrow$ Artificial neural network

First, let us review biological neurons

Brains and neural networks rely on **neurons**

Brain: Biological neurons $\rightarrow$ Biological neural network

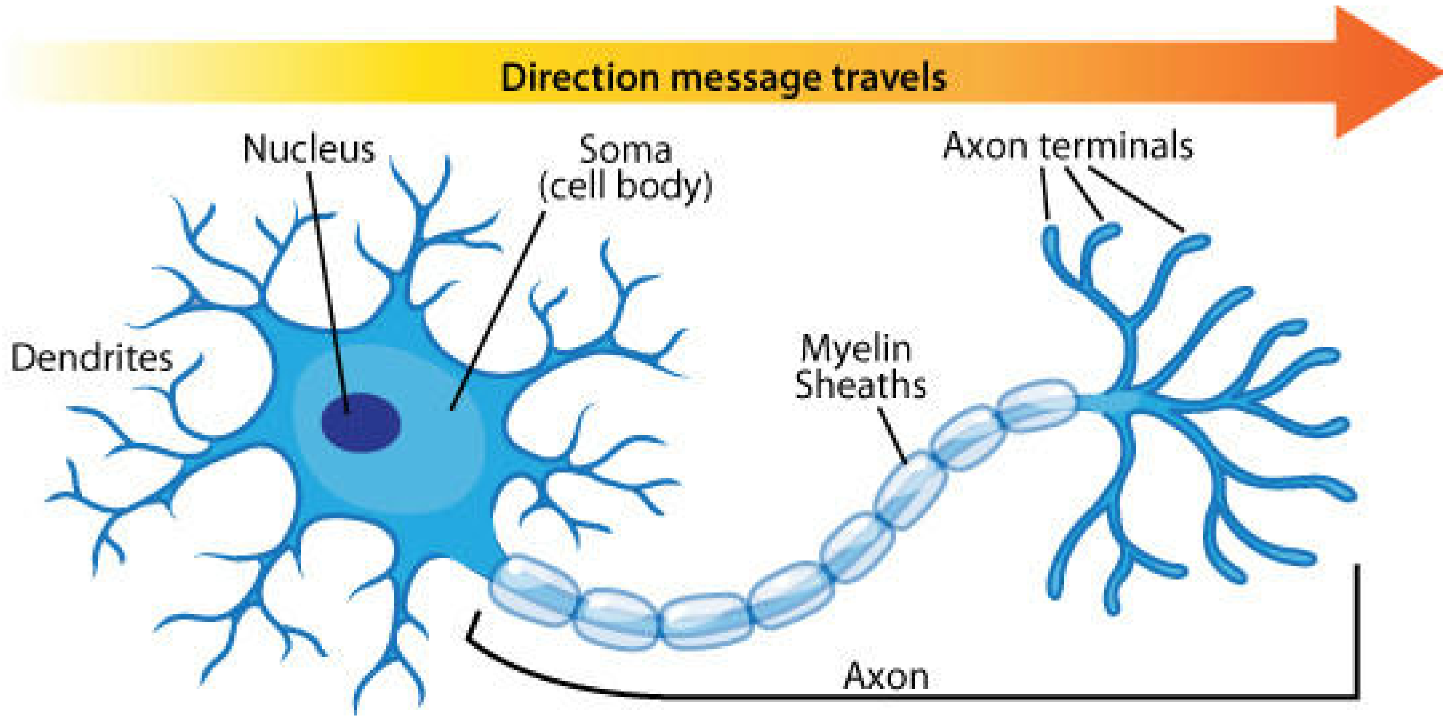
Computer: Artificial neurons $\rightarrow$ Artificial neural network

First, let us review biological neurons

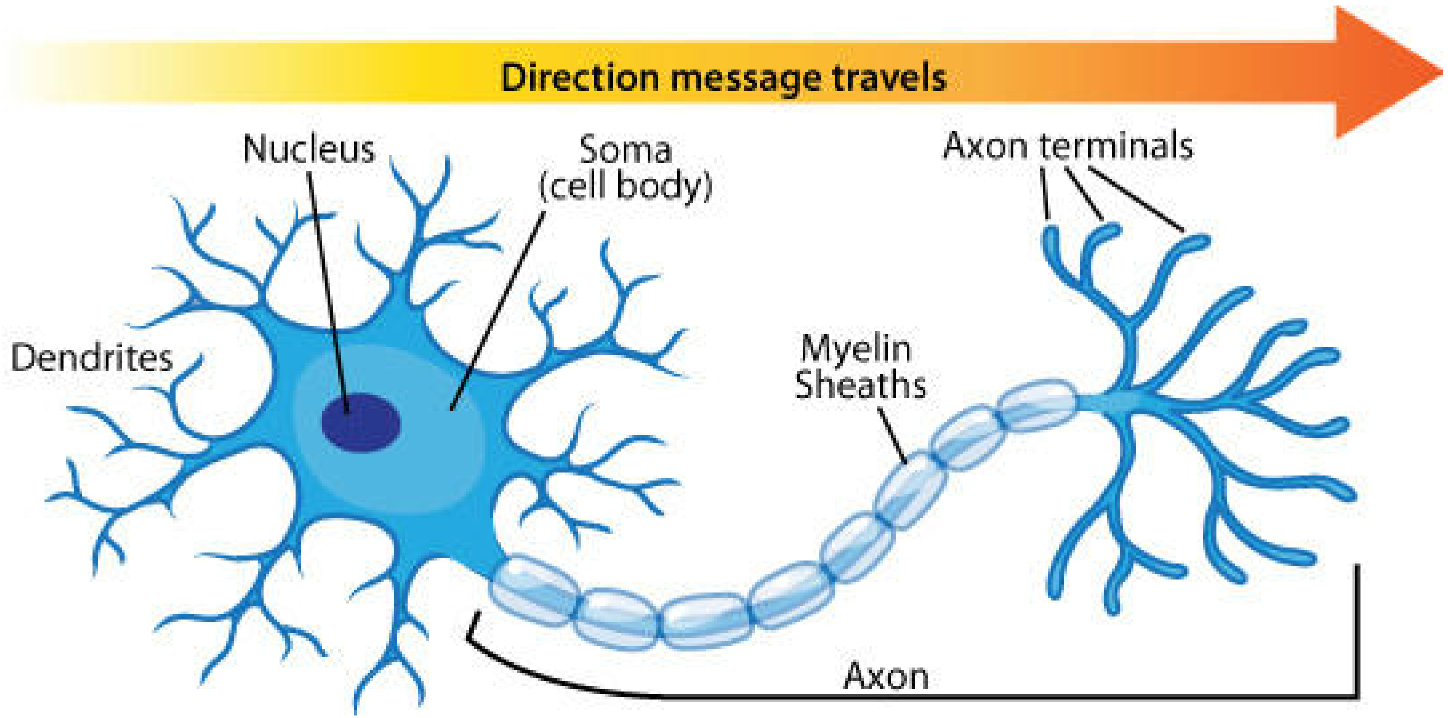
Note: I am not a neuroscientist! I may make simplifications or errors with biology

1. Review
2. Multivariate linear regression
3. Limitations of linear regression
4. **History of neural networks**
5. Biological neurons
6. Artificial neurons
7. Wide neural networks
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron

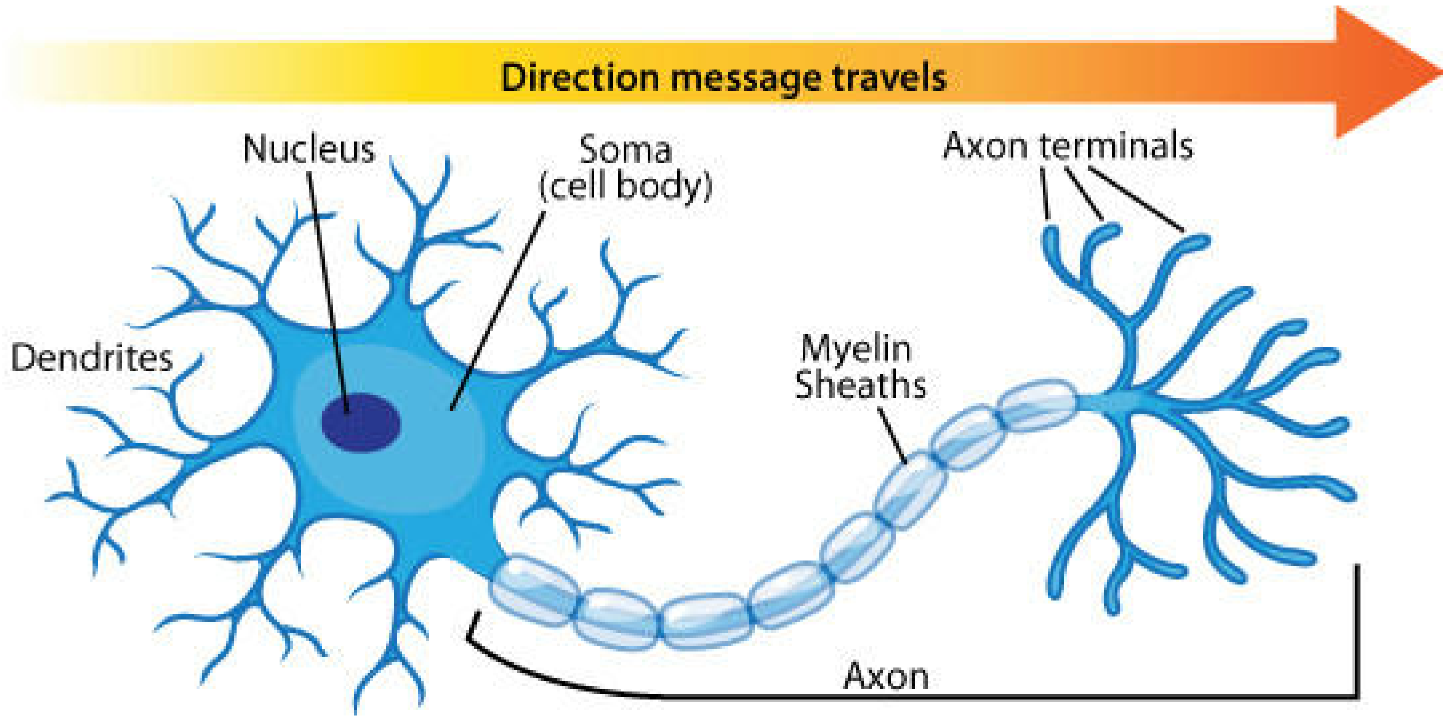
1. Review
2. Multivariate linear regression
3. Limitations of linear regression
4. History of neural networks
5. **Biological neurons**
6. Artificial neurons
7. Wide neural networks
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron



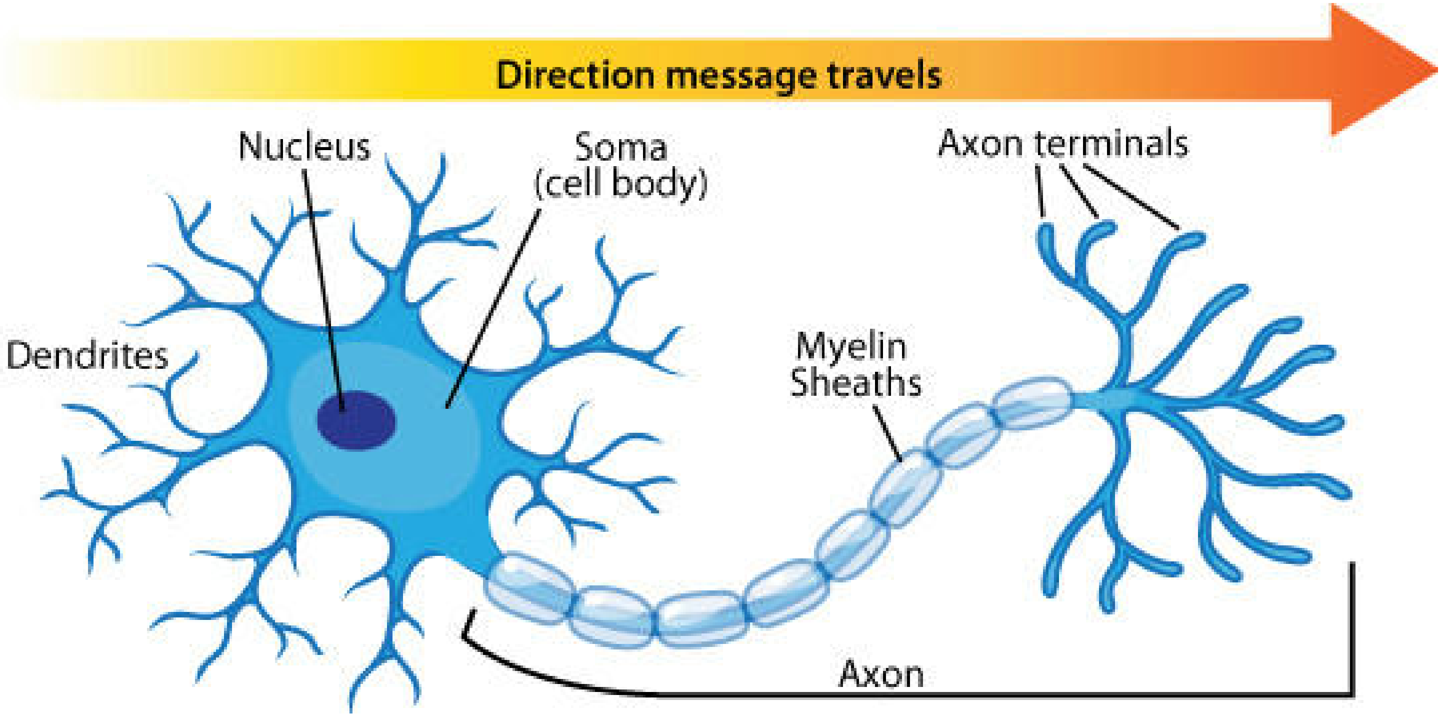
A simplified neuron consists of many parts



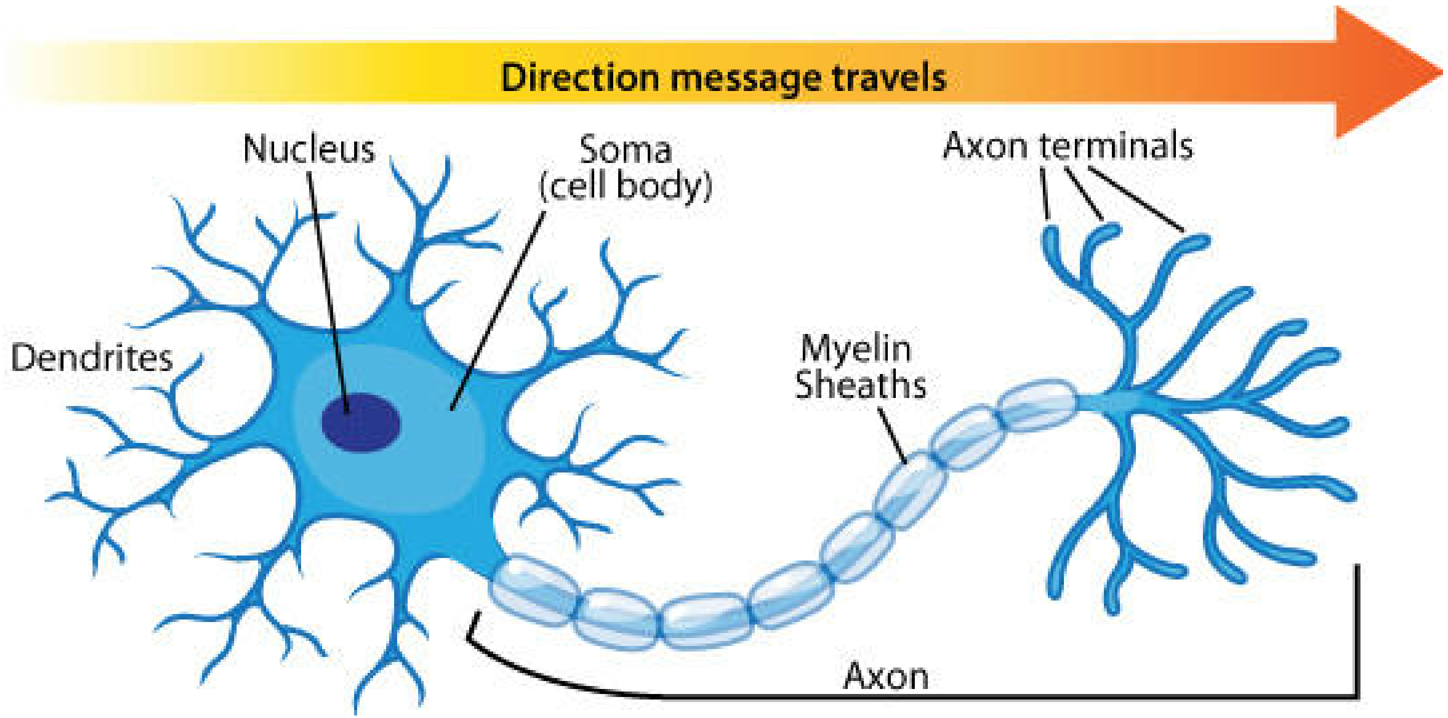
Neurons send and process messages from other neurons



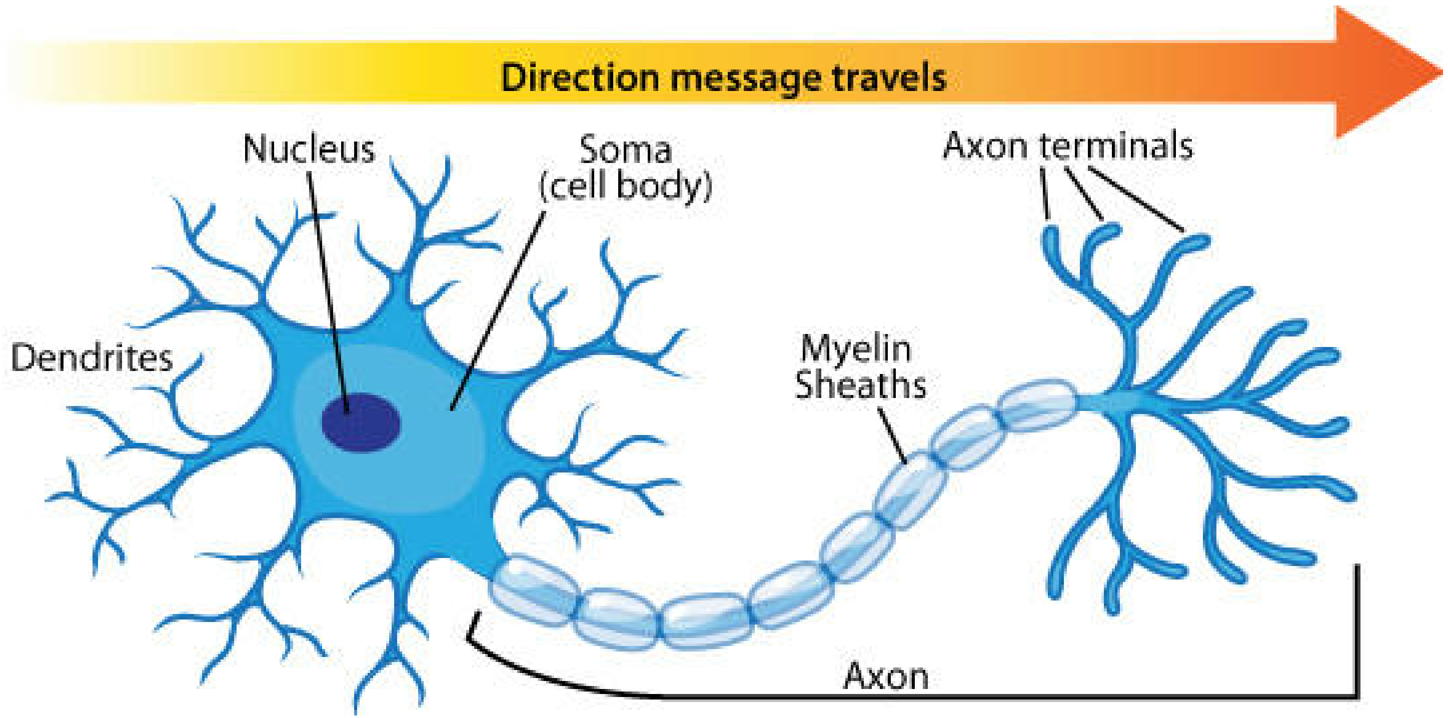
Incoming electrical signals travel along dendrites



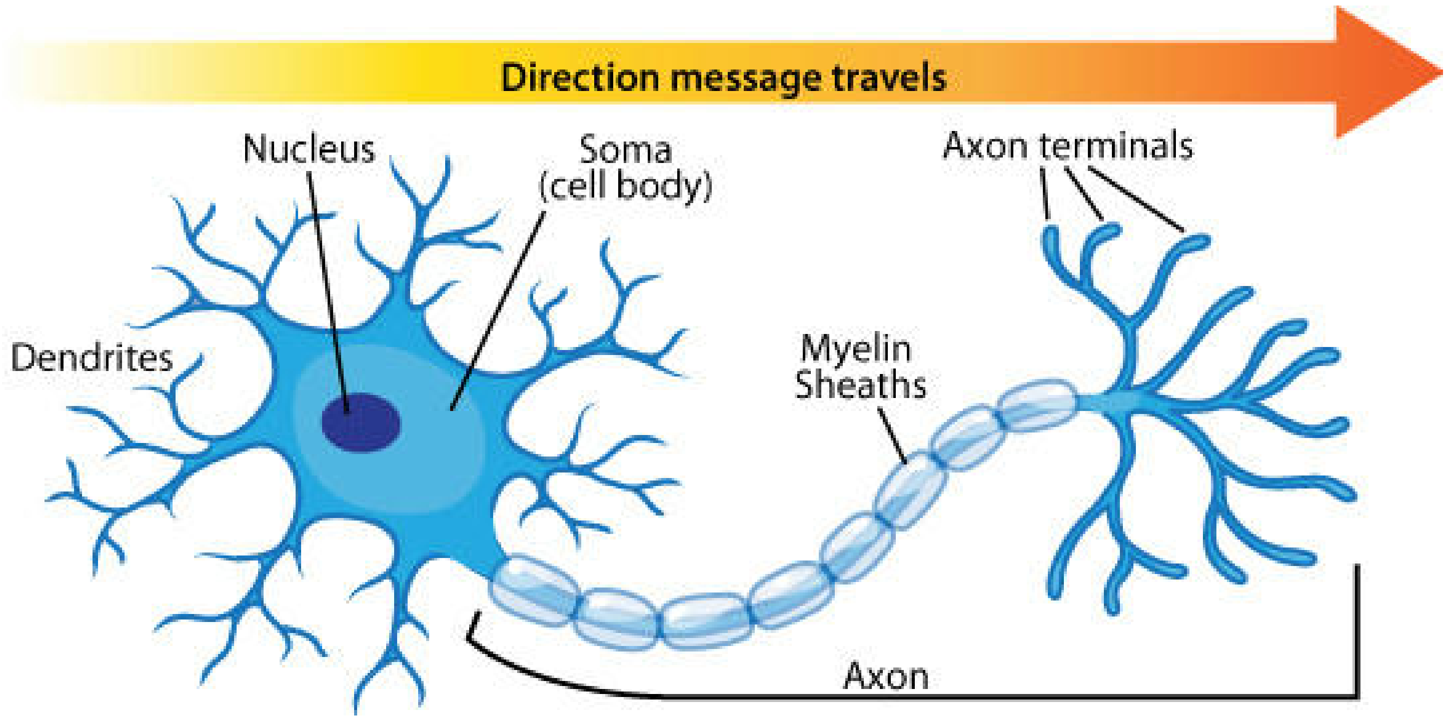
Dendrites are not all equal! Different dendrites have different diameters and structures



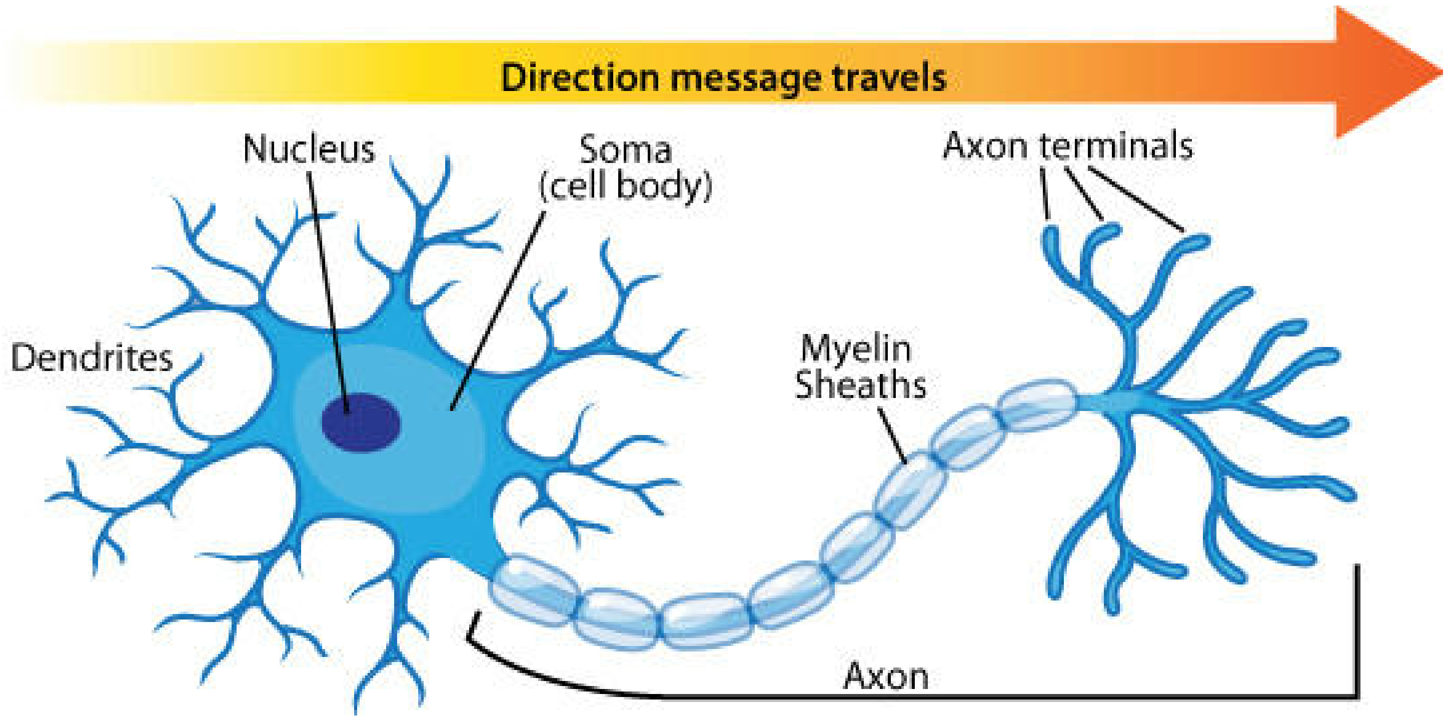
The axon outputs an electrical signal to other neurons



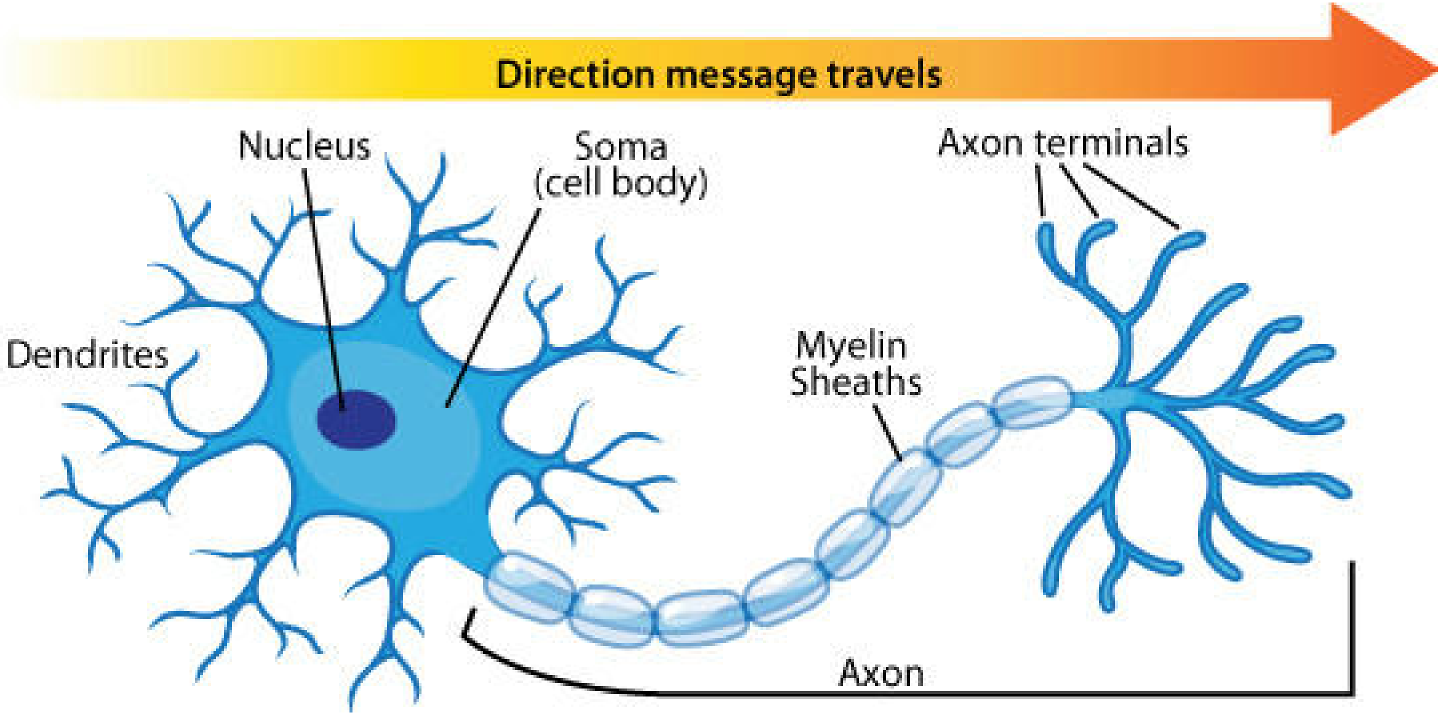
The axon terminals will connect to dendrites of other neurons



For our purposes, we can consider the axon terminals and dendrites to be the same thing



The neuron takes many inputs, and produces a single output

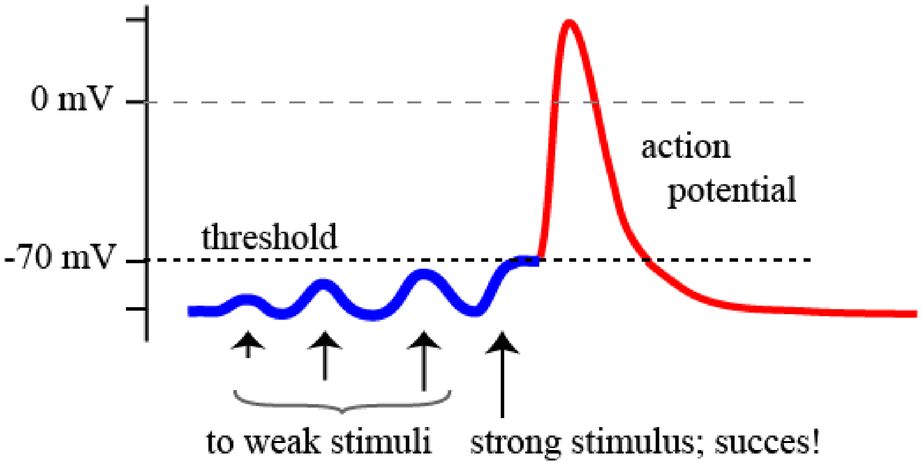


The neuron will only output a signal down the axon ("fire") at certain times

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

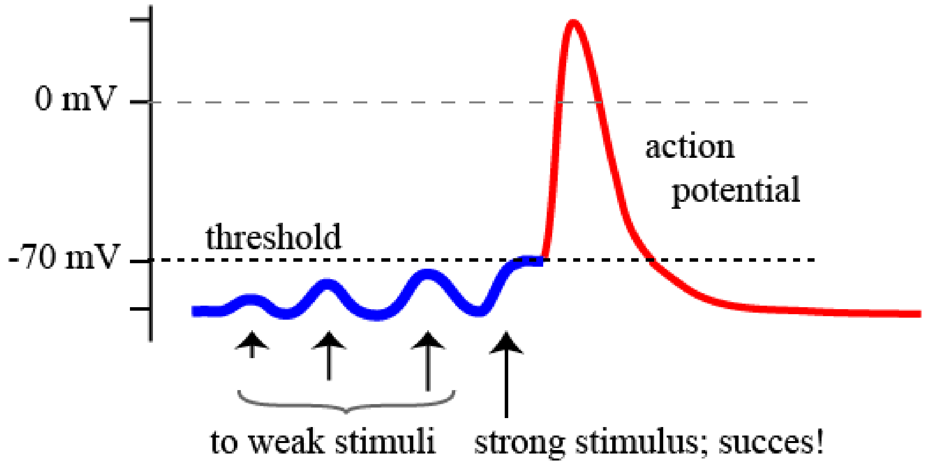
How does a neuron decide to send an impulse (“fire”)?

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



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

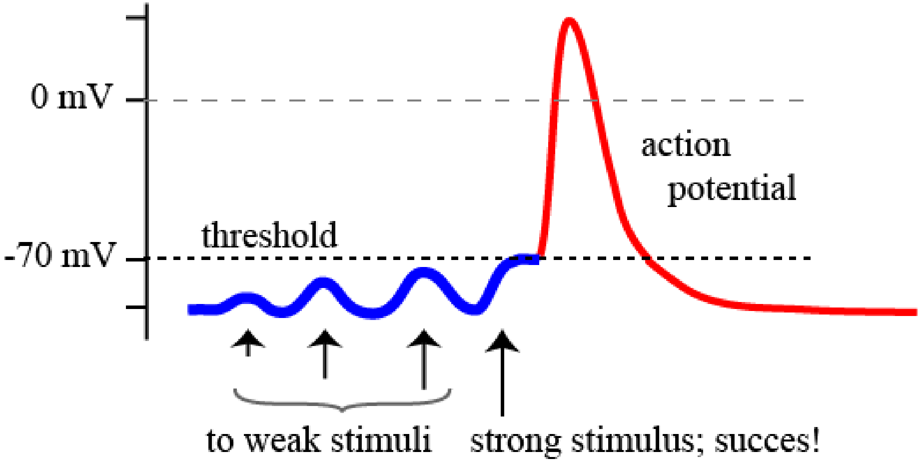
Incoming impulses (via dendrites) change the electric potential of the neuron



Recall that in a parallel circuit, we can sum voltages together

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

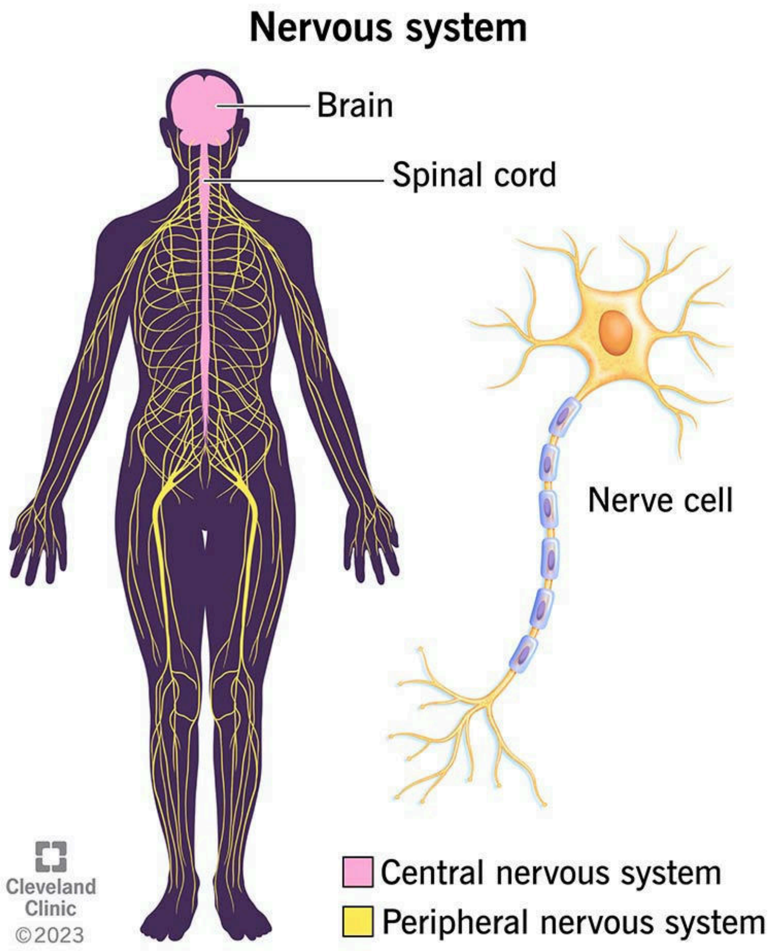
Incoming impulses (via dendrites) change the electric potential of the neuron



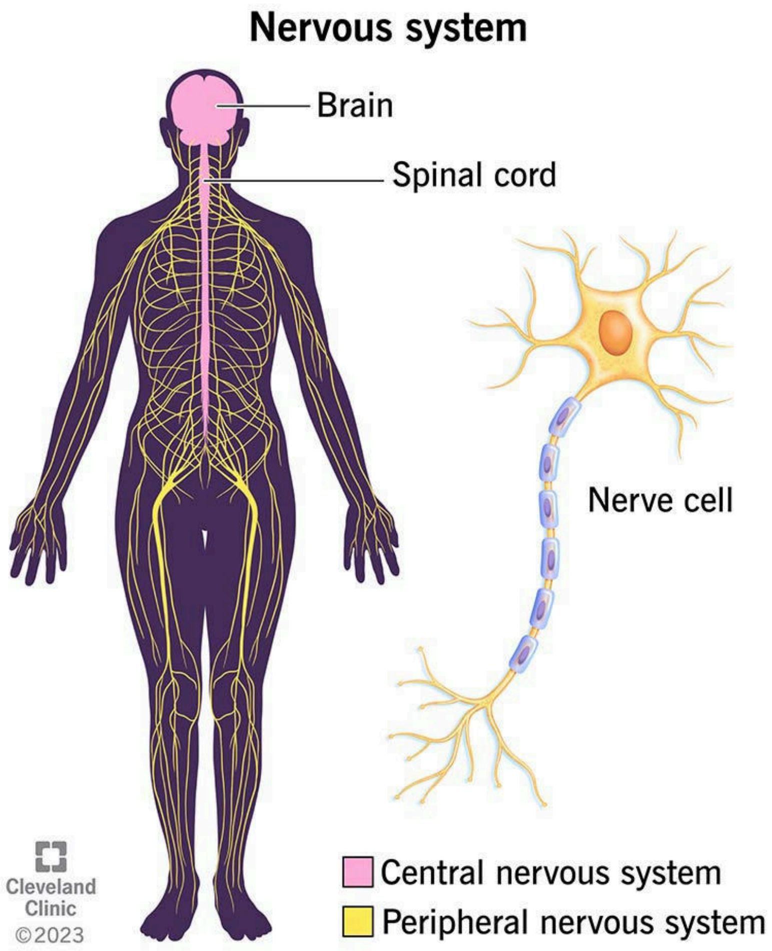
Recall that in a parallel circuit, we can sum voltages together

Many active dendrites will add together and trigger an impulse

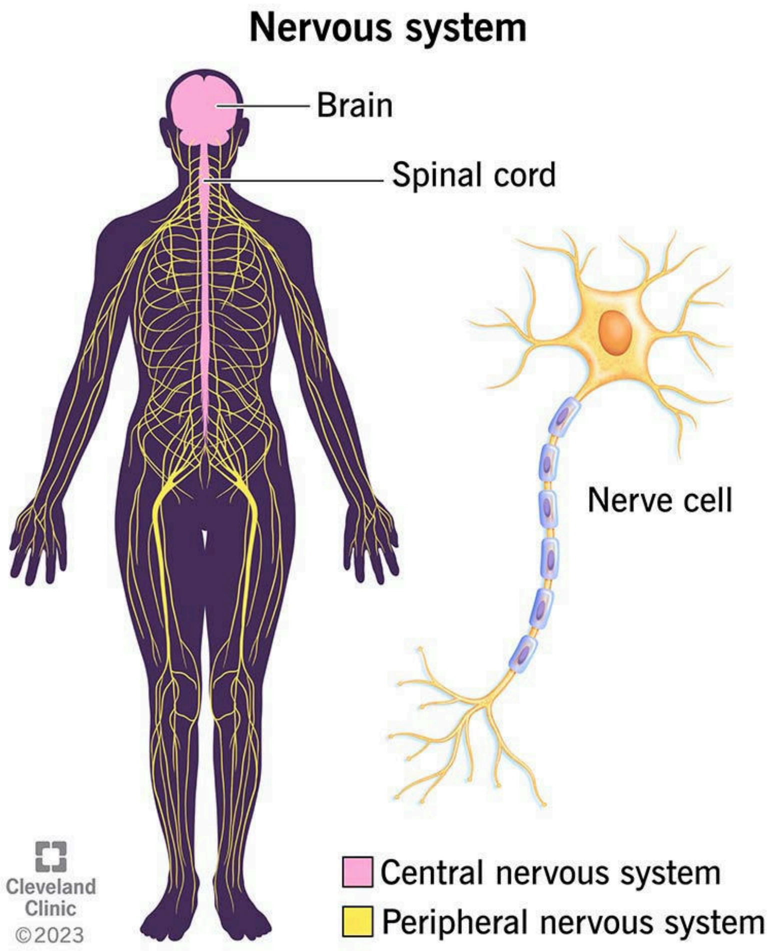
Pain triggers initial nerve impulse, starts a chain reaction into the brain



When the signal reaches the brain, we will think

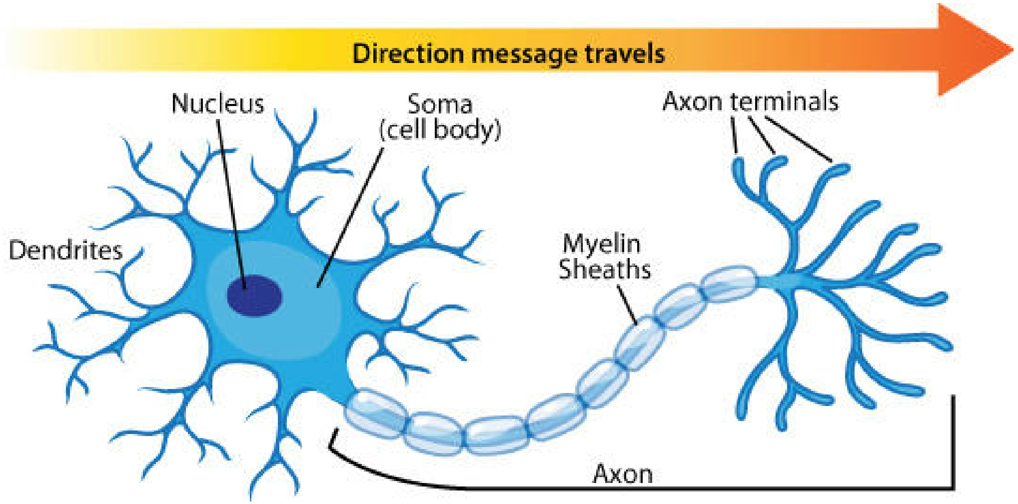


After thinking, we will take action

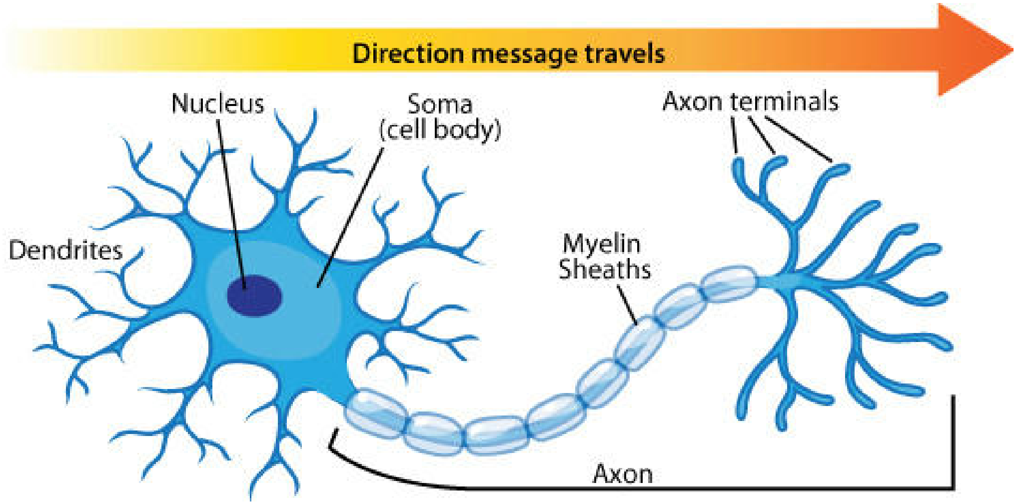


1. Review
2. Multivariate linear regression
3. Limitations of linear regression
4. History of neural networks
5. **Biological neurons**
6. Artificial neurons
7. Wide neural networks
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron

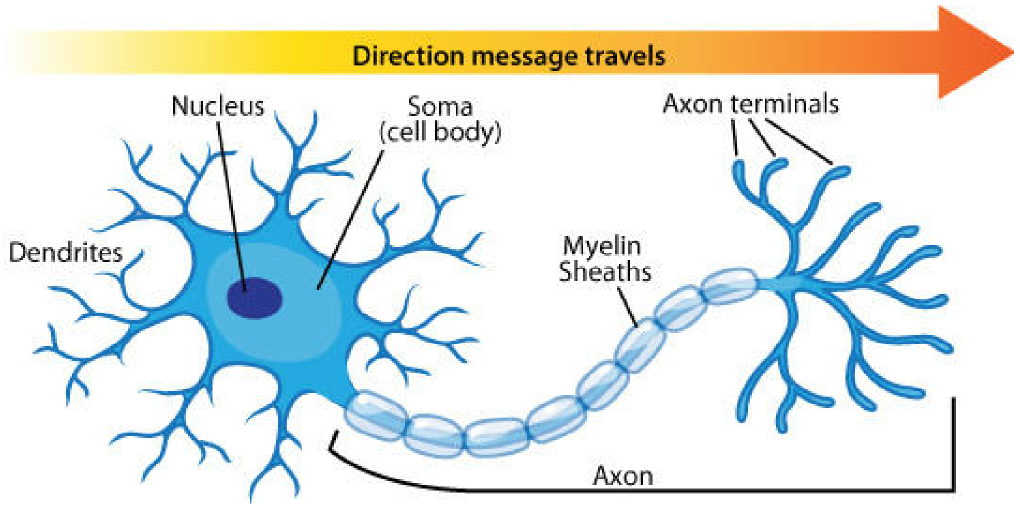
1. Review
2. Multivariate linear regression
3. Limitations of linear regression
4. History of neural networks
5. Biological neurons
6. **Artificial neurons**
7. Wide neural networks
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron



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



Question: How could we write a neuron as a function? $f : \_\_ \mapsto \_\_$



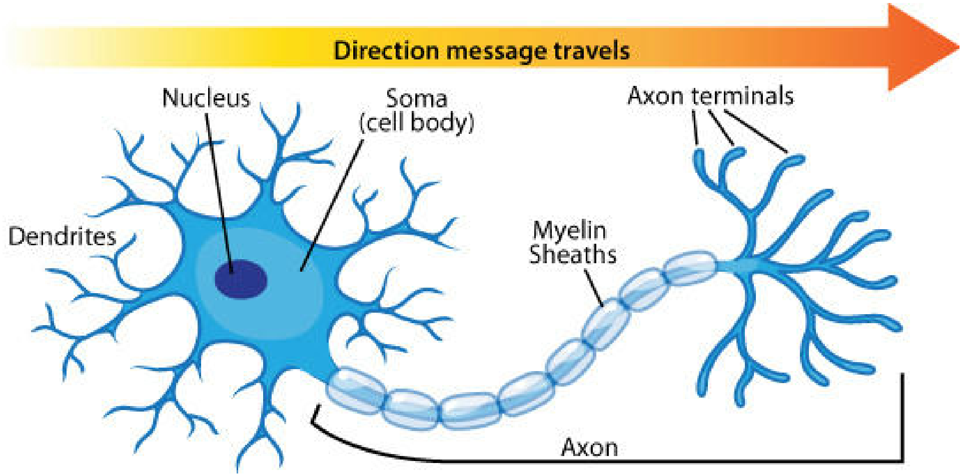
Question: How could we write a neuron as a function? $f : \_\_ \mapsto \_\_$

Answer:

$$f : \underbrace{\mathbb{R}^{d_x}}_{\text{Dendrite voltages}} \times \underbrace{\mathbb{R}^{d_x}}_{\text{Dendrite size}} \mapsto \underbrace{\mathbb{R}}_{\text{Axon voltage}}$$

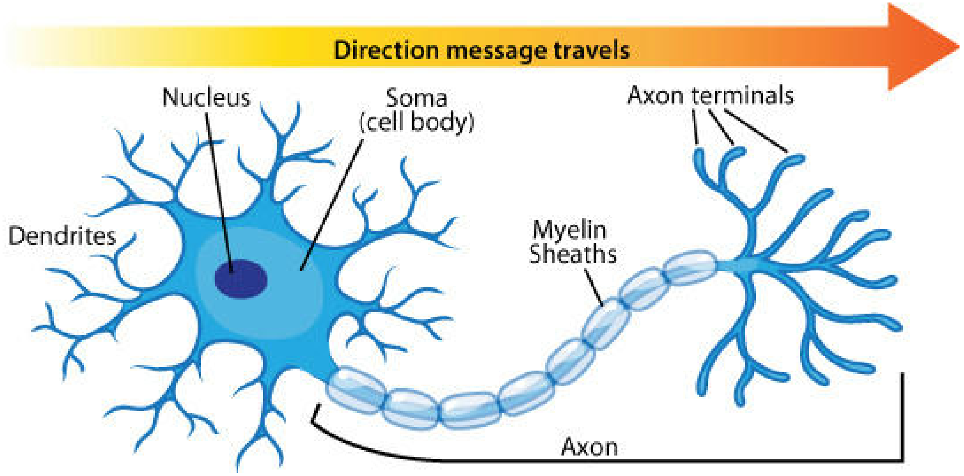
Let us implement an artifical neuron as a function

Let us implement an artifical neuron as a function



Neuron has a structure of dendrites

Let us implement an artifical neuron as a function

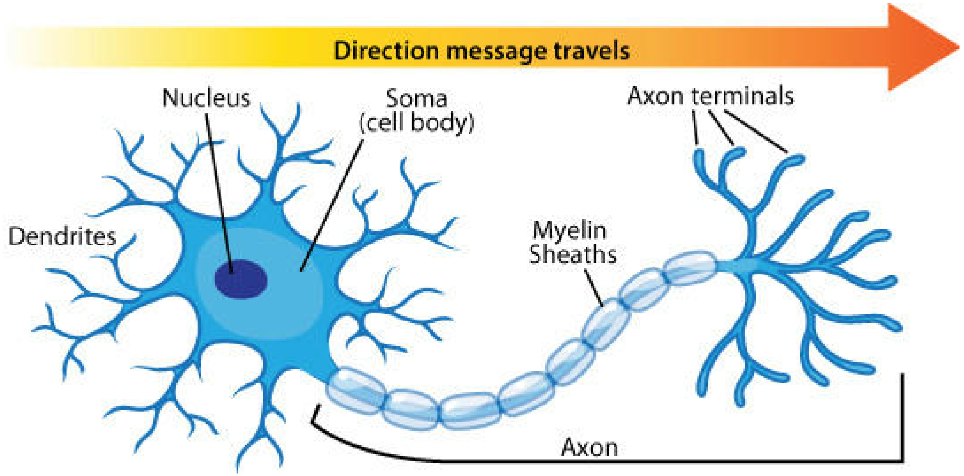


Neuron has a structure of dendrites

$$f\left(\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{d_x} \end{bmatrix}\right)$$

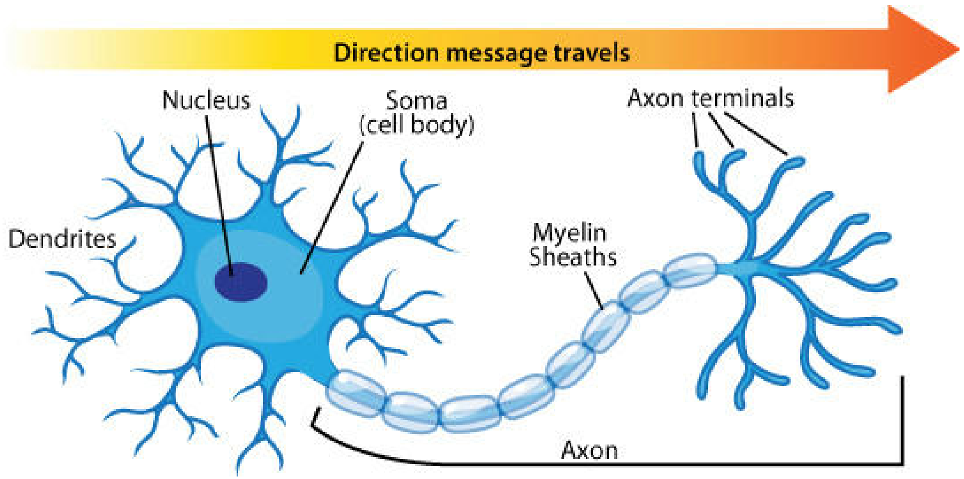
$$f(\boldsymbol{\theta})$$

Let us implement an artifical neuron as a function



Each incoming dendrite has some voltage potential

Let us implement an artifical neuron as a function

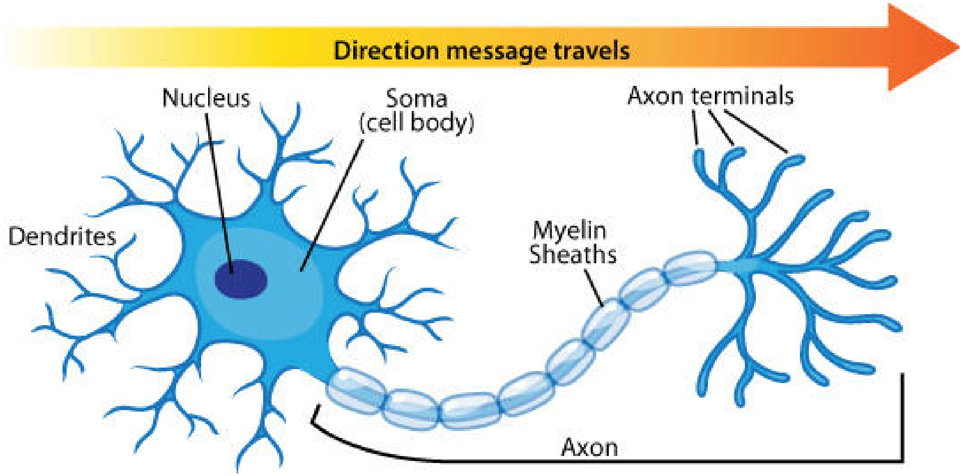


Each incoming dendrite has some voltage potential

$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_{d_x} \end{bmatrix}, \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_{d_x} \end{bmatrix}\right)$$

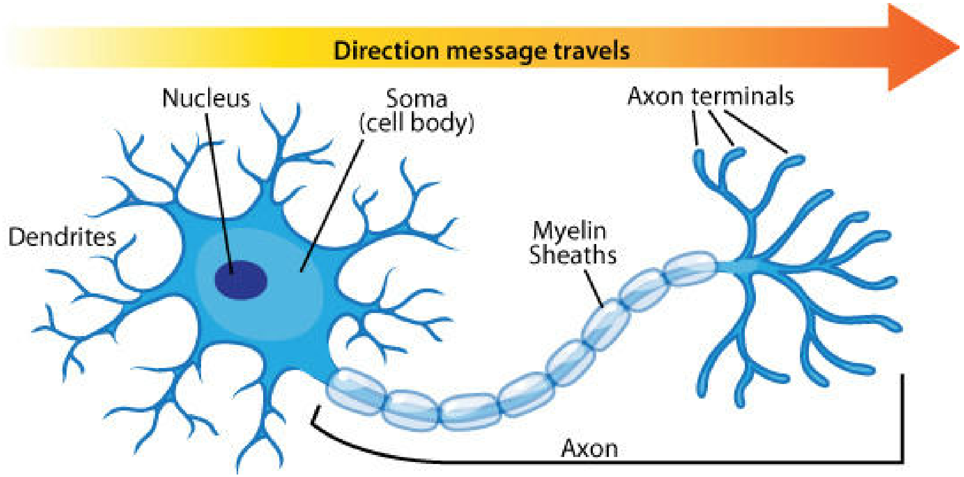
$$f(\boldsymbol{x}, \boldsymbol{\theta})$$

Let us implement an artifical neuron as a function



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

Let us implement an artifical neuron as a function

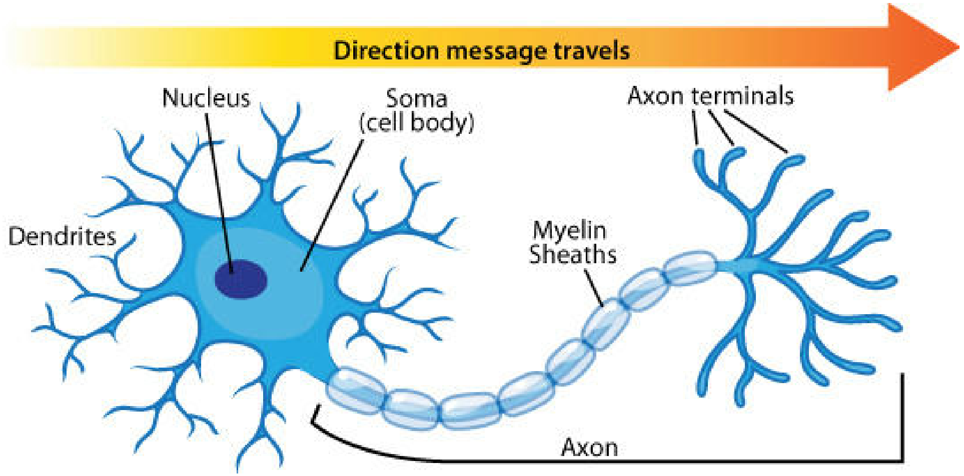


Voltage potentials sum together to give us the voltage in the cell body

$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_{d_x} \end{bmatrix}, \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_{d_x} \end{bmatrix}\right) = \sum_{j=1}^{d_x} \theta_j x_j$$

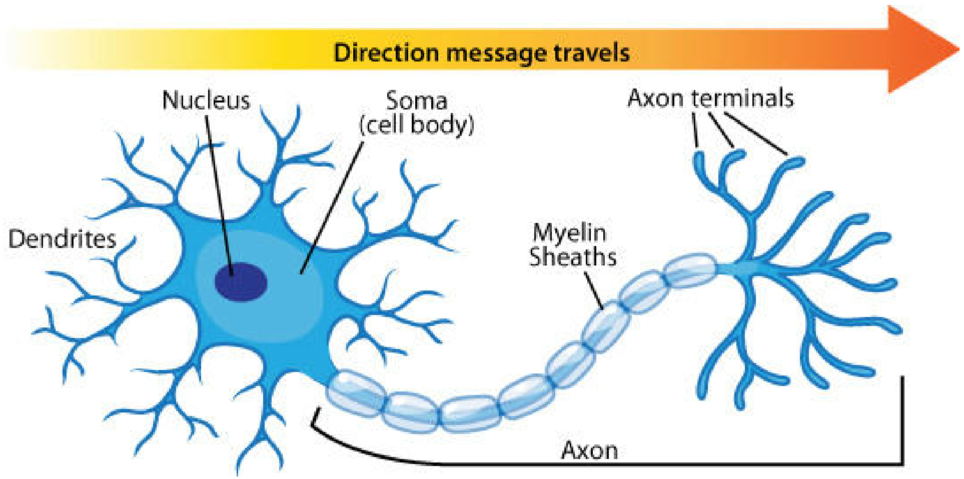
$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{x}$$

Let us implement an artifical neuron as a function



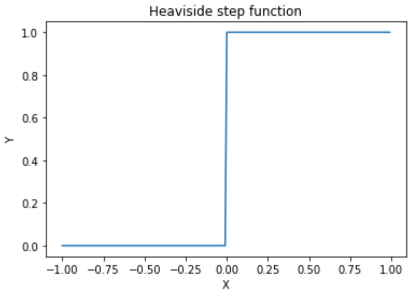
The axon fires only if the voltage is over a threshold

Let us implement an artifical neuron as a function

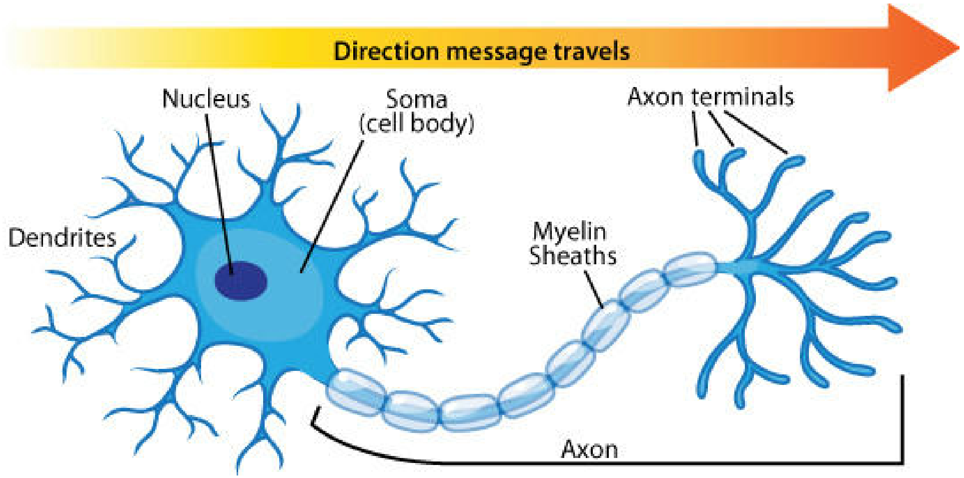


The axon fires only if the voltage is over a threshold

$$\sigma(x) =$$

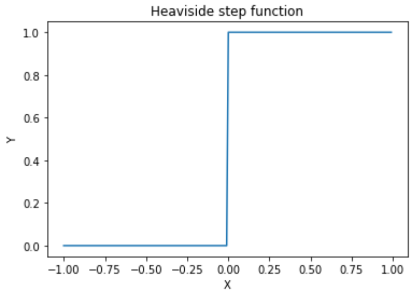


Let us implement an artifical neuron as a function



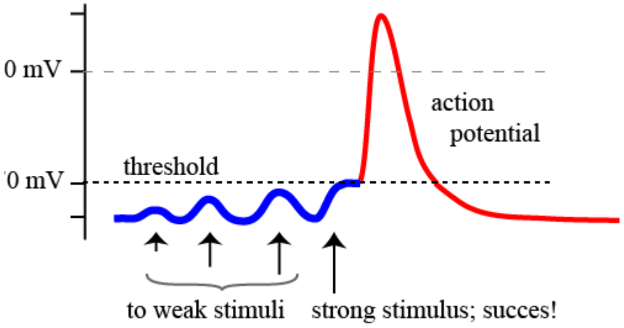
The axon fires only if the voltage is over a threshold

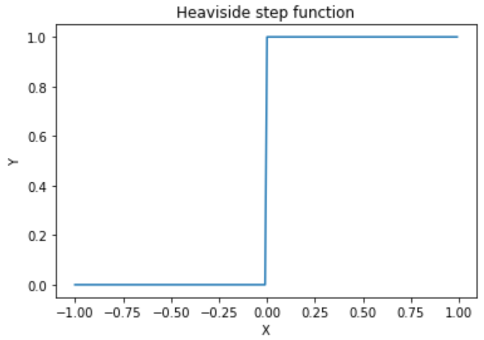
$$\sigma(x) =$$



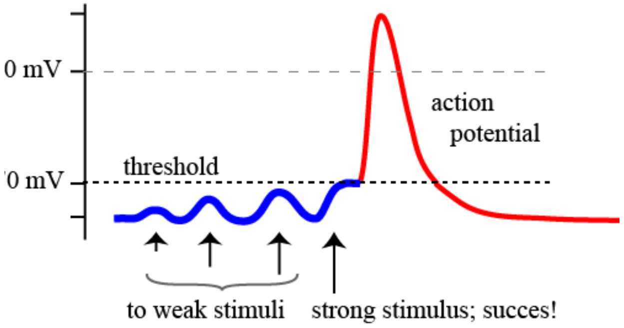
$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}\right) = \sigma\left(\sum_{j=1}^{d_x} x_j \theta_j\right)$$

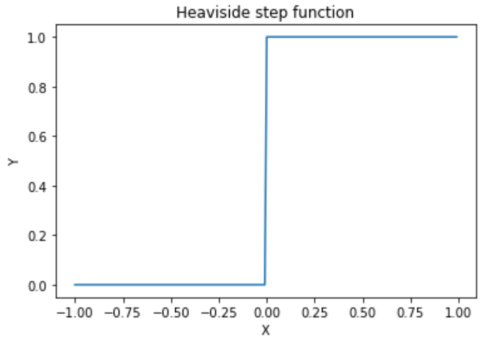
Maybe we want to vary the activation threshold





Maybe we want to vary the activation threshold





$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_{d_x} \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{d_x} \end{bmatrix}\right) = \sigma\left(\theta_0 + \sum_{j=1}^{d_x} \theta_j x_j\right)$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta}_{1:d_x}^\top \boldsymbol{x}$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta}_{1:d_x}^\top \boldsymbol{x}$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta}_{1:d_x}^\top \boldsymbol{x}$$

This is the artificial neuron!

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta}_{1:d_x}^\top \boldsymbol{x}$$

This is the artificial neuron!

Let us write out the full equation for a neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta}_{1:d_x}^\top \boldsymbol{x}$$

This is the artificial neuron!

Let us write out the full equation for a neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\theta_{d_x} x_{d_x} + \theta_{d_x - 1} x_{d_x - 1} + \ldots + \theta_0\right)$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta}_{1:d_x}^\top \boldsymbol{x}$$

This is the artificial neuron!

Let us write out the full equation for a neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\theta_{d_x} x_{d_x} + \theta_{d_x - 1} x_{d_x - 1} + \ldots + \theta_0\right)$$

Question: Does this look familiar to anyone?

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta}_{1:d_x}^\top \boldsymbol{x}$$

This is the artificial neuron!

Let us write out the full equation for a neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\theta_{d_x} x_{d_x} + \theta_{d_x - 1} x_{d_x - 1} + \ldots + \theta_0\right)$$

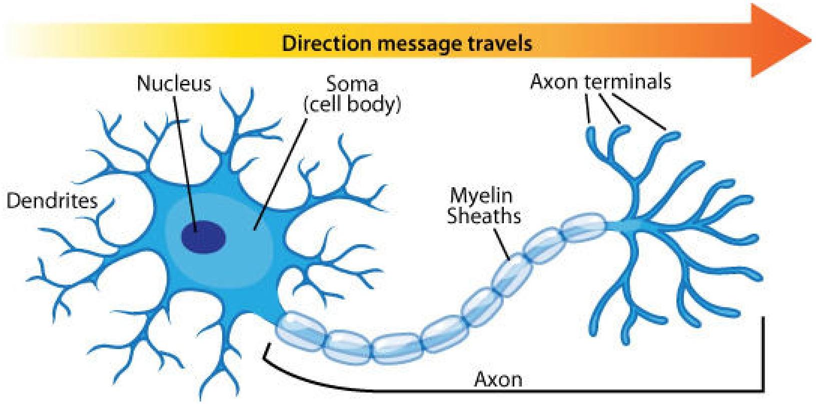
Question: Does this look familiar to anyone?

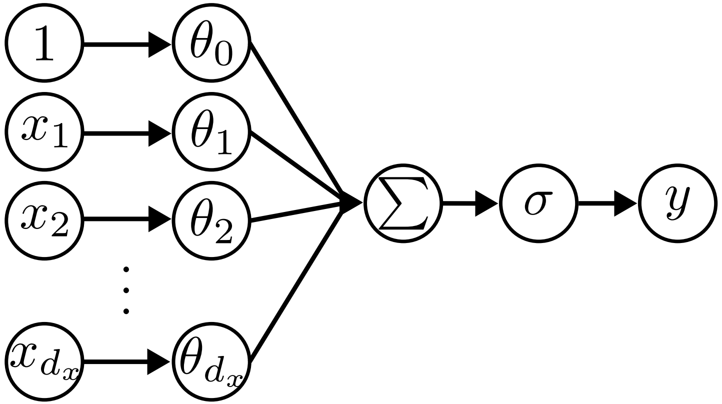
Answer: Inside σ is the multivariate linear model!

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_{d_x} x_{d_x} + \theta_{d_x - 1} x_{d_x - 1} + \ldots + \theta_0$$

We model a neuron using a linear model and activation function

We model a neuron using a linear model and activation function





$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta}_{1:d_x}^\top \boldsymbol{x}$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta}_{1:d_x}^\top \boldsymbol{x}$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta}_{1:d_x}^\top \boldsymbol{x}$$

Sometimes, we will write $\boldsymbol{\theta}$ as a bias and weight $b, \boldsymbol{w}$

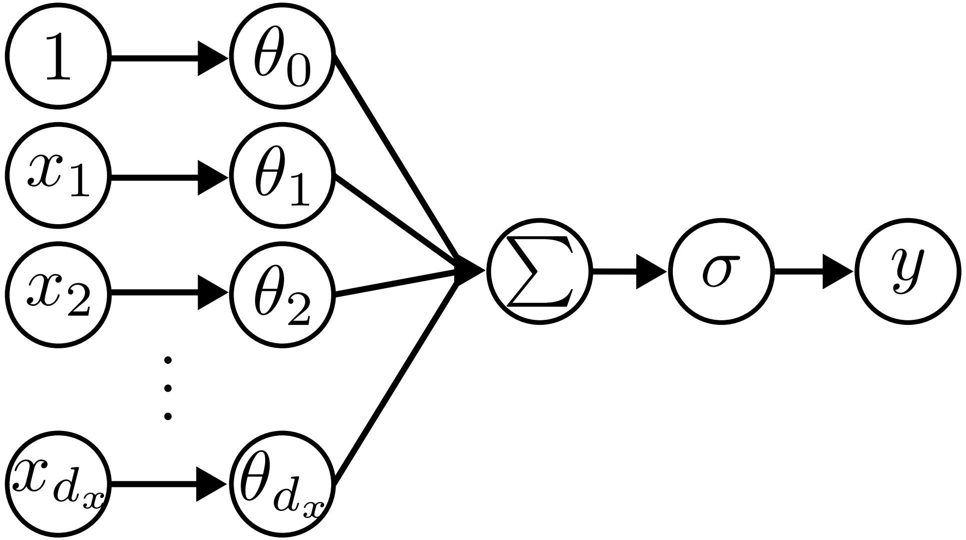
$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta}_{1:d_x}^\top \boldsymbol{x}$$

Sometimes, we will write $\boldsymbol{\theta}$ as a bias and weight $b, \boldsymbol{w}$

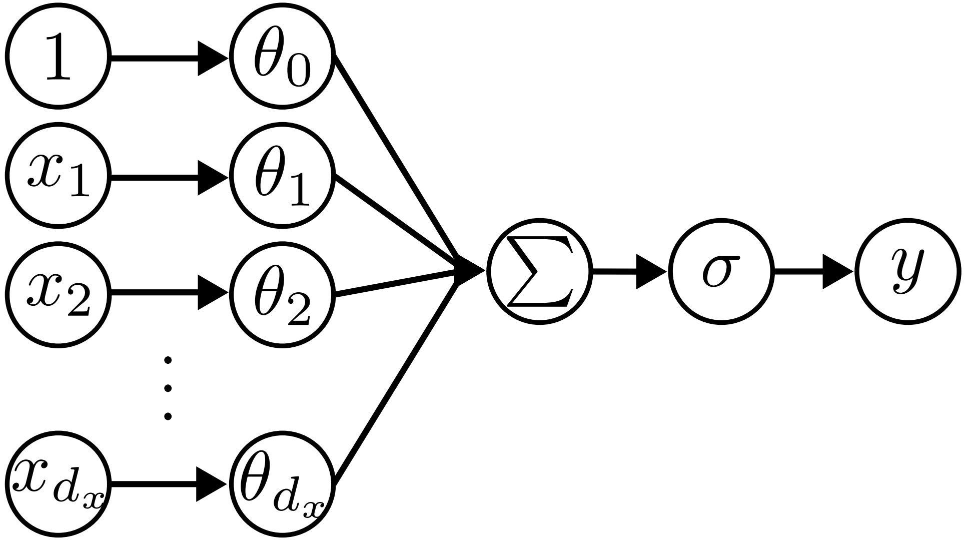
$$\boldsymbol{\theta} = \begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix}; \quad \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{d_x} \end{bmatrix} = \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_{d_x} \end{bmatrix}$$

$$f\left(\boldsymbol{x}, \begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix}\right) = b + \boldsymbol{w}^\top \boldsymbol{x}$$

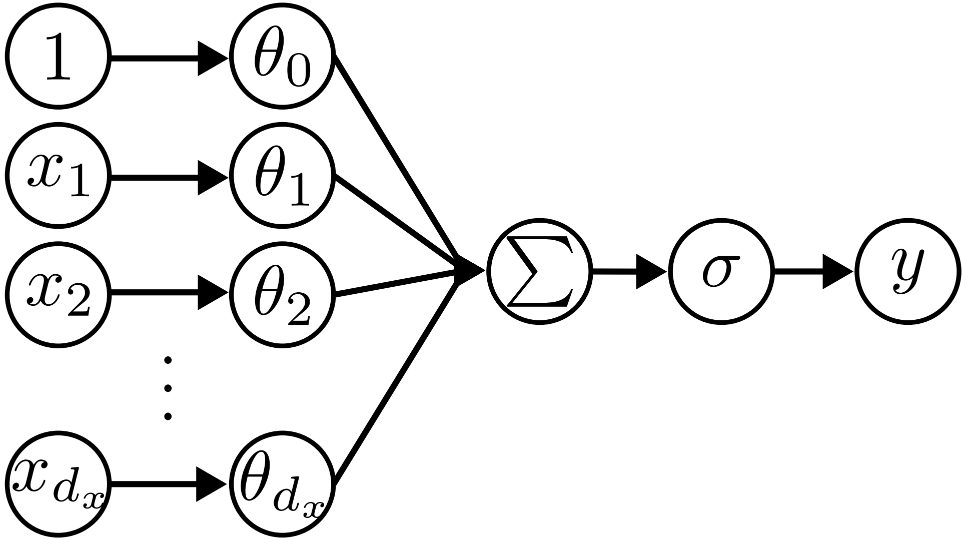
Relax



1
x_1
x_2
x_{d_x}
θ_0
θ_1
θ_2
θ_{d_x}
$\sum$
σ
y

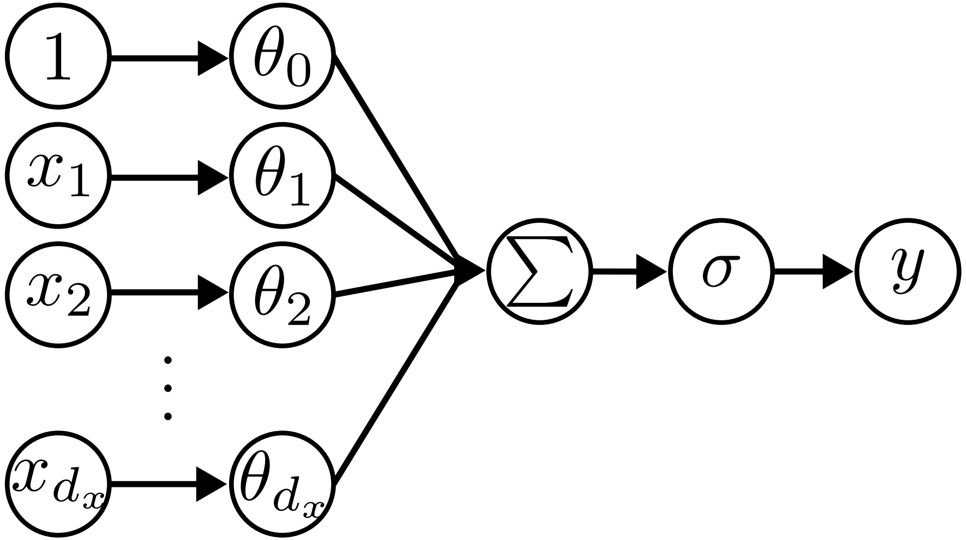


Recall that in machine learning we deal with functions



Recall that in machine learning we deal with functions

What kinds of functions can our neuron represent?

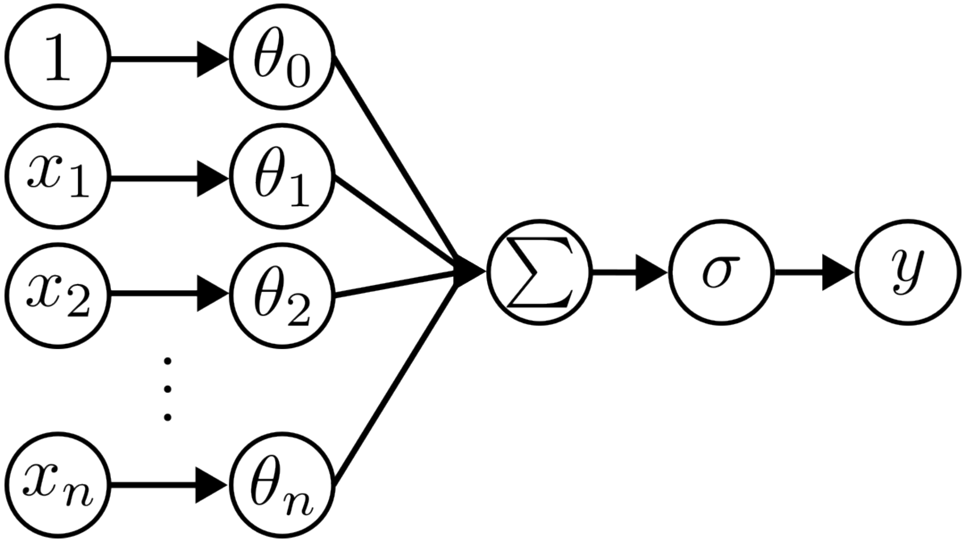


Recall that in machine learning we deal with functions

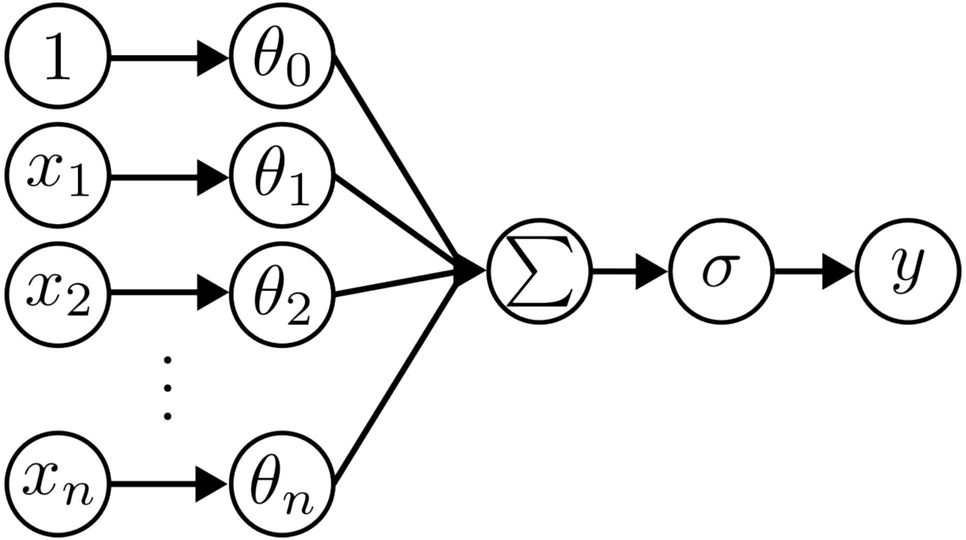
What kinds of functions can our neuron represent?

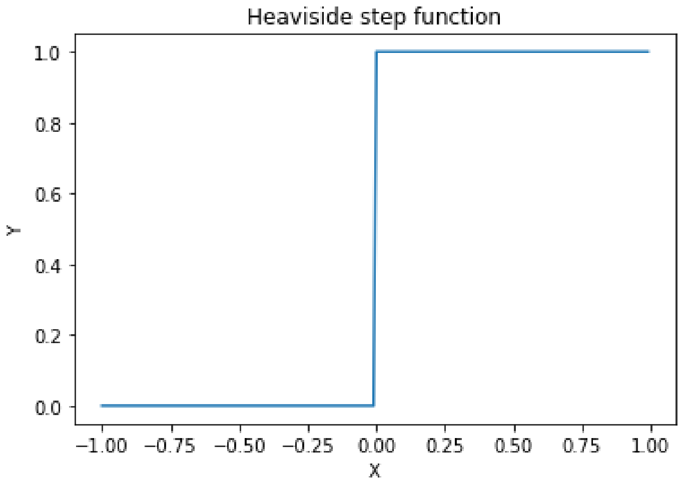
Let us start with a logical AND function

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

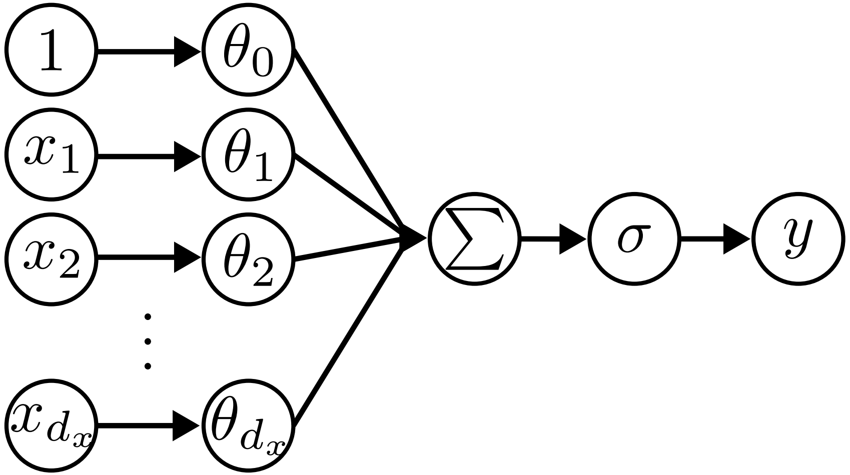
Let us think back to biology, maybe it has an answer

Brain: Biological neurons $\rightarrow$ Biological neural network

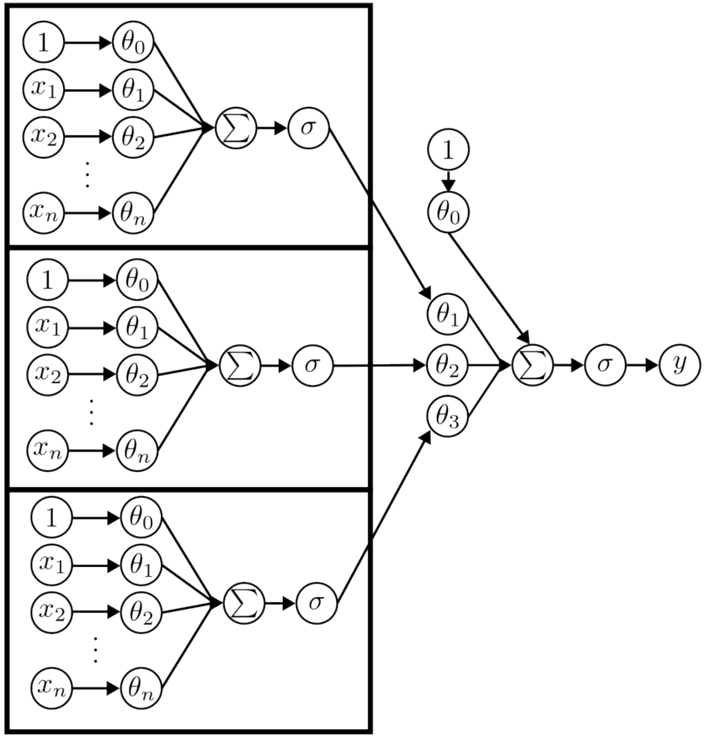
Brain: Biological neurons $\to$ Biological neural network

Computer: Artificial neurons $\to$ Artificial neural network

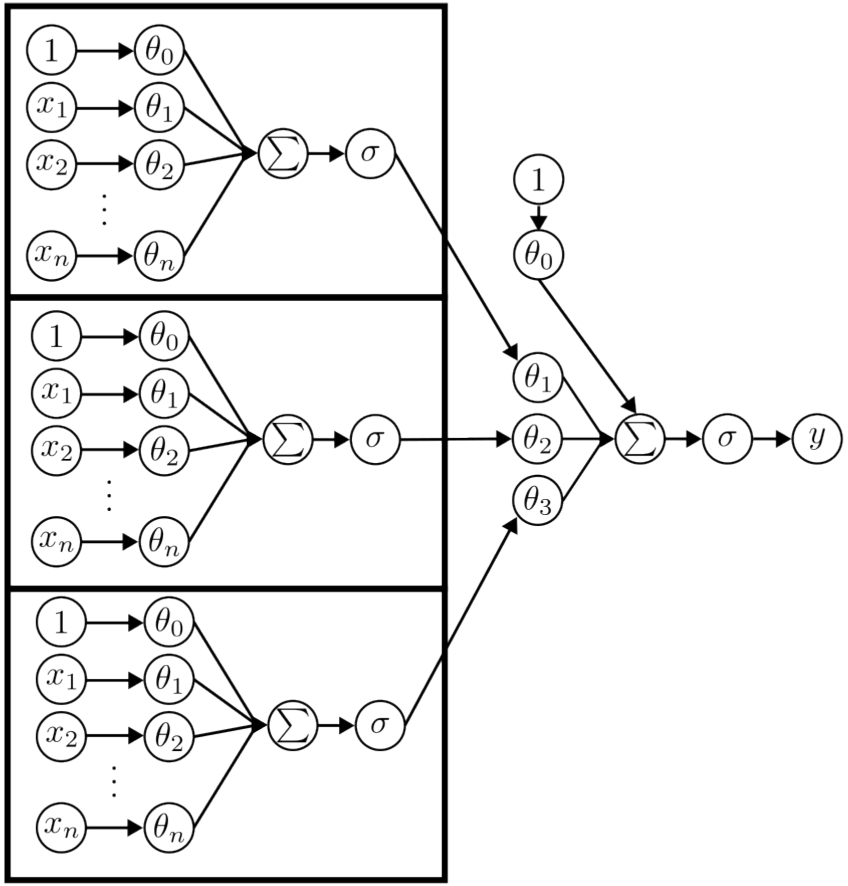
Connect artificial neurons into a network



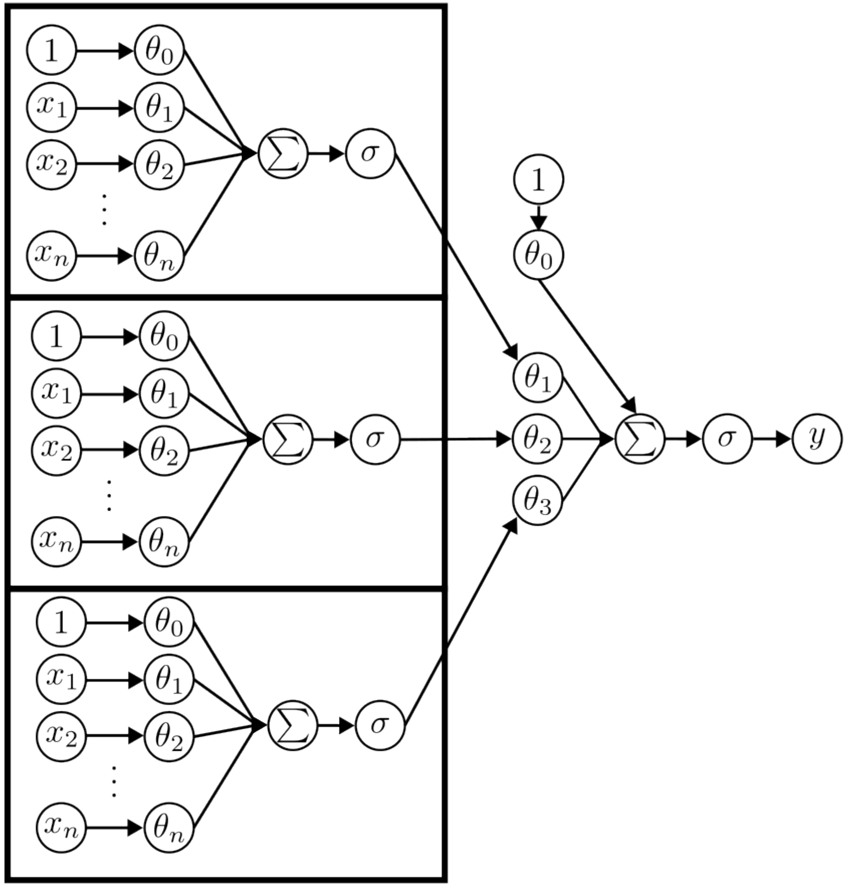
Neuron



Neural Network

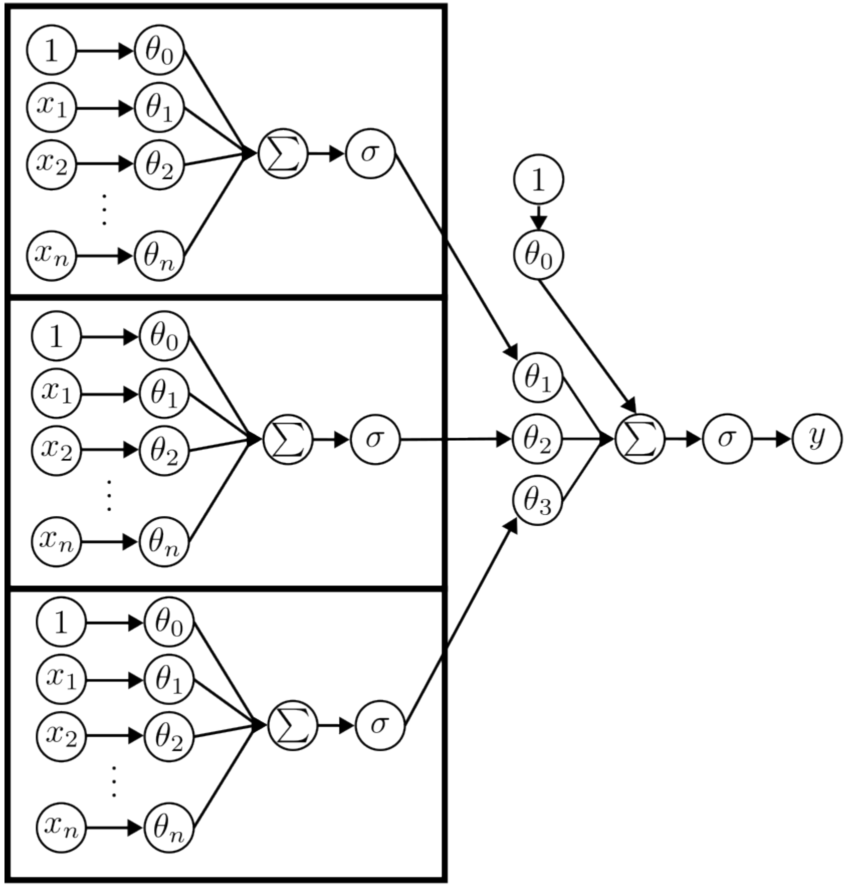


Adding neurons in **parallel** creates a **wide** neural network



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

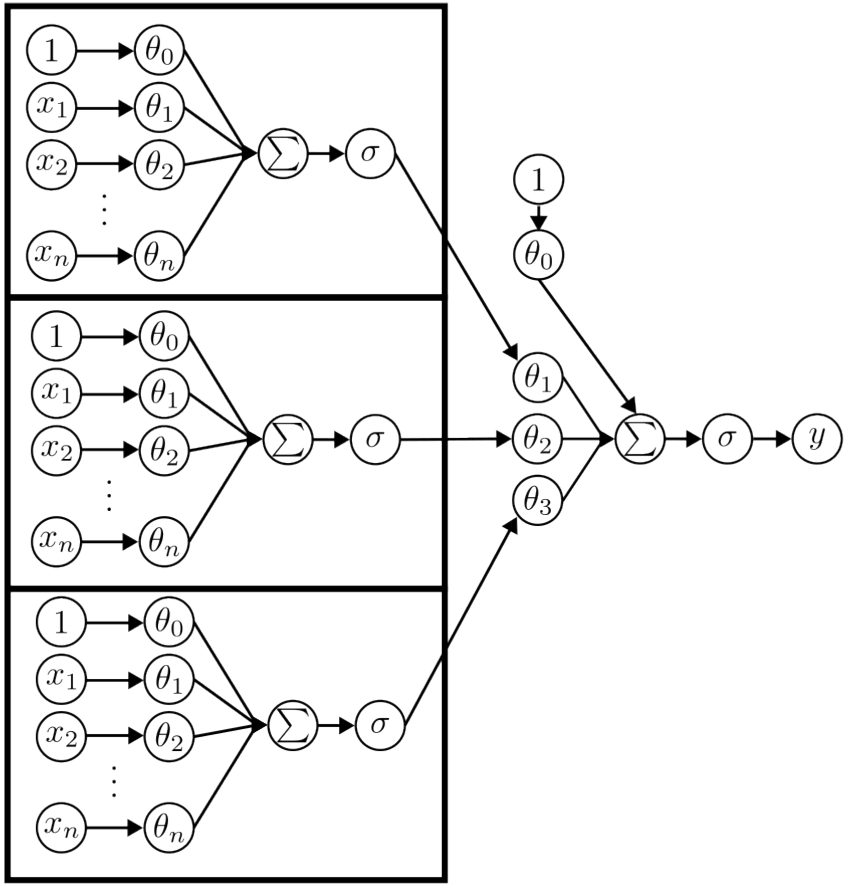
Adding neurons in **series** creates a **deep** neural network



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

1. Review
2. Multivariate linear regression
3. Limitations of linear regression
4. History of neural networks
5. Biological neurons
6. **Artificial neurons**
7. Wide neural networks
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron

1. Review
2. Multivariate linear regression
3. Limitations of linear regression
4. History of neural networks
5. Biological neurons
6. Artificial neurons
7. **Wide neural networks**
8. Deep neural networks
9. Perceptron
10. Multilayer Perceptron

How do we express a **wide** neural network mathematically?

How do we express a **wide** neural network mathematically?

A single neuron:

$$f : \mathbb{R}^{d_x} \times \Theta \mapsto \mathbb{R}$$

$$\Theta \in \mathbb{R}^{d_x + 1}$$

How do we express a **wide** neural network mathematically?

A single neuron:

$$f : \mathbb{R}^{d_x} \times \Theta \mapsto \mathbb{R}$$

$$\Theta \in \mathbb{R}^{d_x + 1}$$

d_y neurons (wide):

$$f : \mathbb{R}^{d_x} \times \Theta \mapsto \mathbb{R}^{d_y}$$

$$\Theta \in \mathbb{R}^{(d_x + 1) \times d_y}$$

How do we express a **wide** neural network mathematically?

A single neuron:

$$f : \mathbb{R}^{d_x} \times \Theta \mapsto \mathbb{R}$$

$$\Theta \in \mathbb{R}^{d_x+1}$$

d_y neurons (wide):

$$f : \mathbb{R}^{d_x} \times \Theta \mapsto \mathbb{R}^{d_y}$$

$$\Theta \in \mathbb{R}^{(d_x+1) \times d_y}$$

For a single neuron:

$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_{d_x} \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_{d_x} \end{bmatrix}\right) = \sigma\left(\theta_0 + \sum_{i=1}^{d_x} \theta_i x_i\right)$$

For a single neuron:

$$f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_{d_x} \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_{d_x} \end{bmatrix}\right) = \sigma\left(\theta_0 + \sum_{i=1}^{d_x} \theta_i x_i\right)$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\theta_0 + \boldsymbol{\theta}_{1:n}^\top \boldsymbol{x})$$

$$f\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} \theta_{1,0} & \theta_{2,0} & \cdots & \theta_{d_x,0} \\ \theta_{1,1} & \theta_{2,1} & \cdots & \theta_{d_x,1} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{1,d_y} & \theta_{2,d_y} & \cdots & \theta_{d_y,d_x} \end{bmatrix}\right) = \begin{bmatrix} \sigma\left(\theta_{1,0} + \sum_{i=1}^{d_x} x_i \theta_{1,i}\right) \\ \sigma\left(\theta_{2,0} + \sum_{i=1}^{d_x} x_i \theta_{2,i}\right) \\ \vdots \\ \sigma\left(\theta_{d_y,0} + \sum_{i=1}^{d_x} x_i \theta_{d_y,i}\right) \end{bmatrix}$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:d_x} \boldsymbol{x}\right); \quad \boldsymbol{\theta} \in \mathbb{R}^{(d_y+1) \times d_x}$$

$$f\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} \theta_{1,0} & \theta_{2,0} & \cdots & \theta_{d_x,0} \\ \theta_{1,1} & \theta_{2,1} & \cdots & \theta_{d_x,1} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{1,d_y} & \theta_{2,d_y} & \cdots & \theta_{d_y,d_x} \end{bmatrix}\right) = \begin{bmatrix} \sigma\left(\theta_{1,0} + \sum_{i=1}^{d_x} x_i \theta_{1,i}\right) \\ \sigma\left(\theta_{2,0} + \sum_{i=1}^{d_x} x_i \theta_{2,i}\right) \\ \vdots \\ \sigma\left(\theta_{d_y,0} + \sum_{i=1}^{d_x} x_i \theta_{d_y,i}\right) \end{bmatrix}$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:d_x} \boldsymbol{x}\right); \quad \boldsymbol{\theta} \in \mathbb{R}^{(d_y+1) \times d_x}$$

$$f\left(\boldsymbol{x}, \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{W} \end{bmatrix}\right) = \sigma(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{x}); \quad \boldsymbol{b} \in \mathbb{R}^{d_y}, \boldsymbol{W} \in \mathbb{R}^{d_y \times d_x}$$

How do we express a **deep** neural network mathematically?

How do we express a **deep** neural network mathematically?

A single neuron:

$$f : \mathbb{R}^n, \boldsymbol{\theta} \mapsto \mathbb{R}$$

How do we express a **deep** neural network mathematically?

A single neuron:

$$f : \mathbb{R}^n, \boldsymbol{\theta} \mapsto \mathbb{R}$$

Multiple neurons (deep):

$$f : \mathbb{R}^n, \boldsymbol{\theta}, \boldsymbol{\psi}, ..., \boldsymbol{\rho} \mapsto \mathbb{R}^m$$

A single neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:n} \boldsymbol{x}$$

A single neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:n} \boldsymbol{x}$$

A composition of neurons with parameters $\boldsymbol{\theta}, \boldsymbol{\psi}, \boldsymbol{\rho}$

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:n} \boldsymbol{x} \quad f_2(\boldsymbol{x}, \boldsymbol{\psi}) = \boldsymbol{\psi}_{\cdot,0} + \boldsymbol{\psi}_{\cdot,1:n} \boldsymbol{x} \quad \dots \quad f_\ell(\boldsymbol{x}, \boldsymbol{\rho}) = \boldsymbol{\rho}_{\cdot,0} + \boldsymbol{\rho}_{\cdot,1:n} \boldsymbol{x}$$

A single neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:n} \boldsymbol{x}$$

A composition of neurons with parameters $\boldsymbol{\theta}, \boldsymbol{\psi}, \boldsymbol{\rho}$

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:n} \boldsymbol{x} \quad f_2(\boldsymbol{x}, \boldsymbol{\psi}) = \boldsymbol{\psi}_{\cdot,0} + \boldsymbol{\psi}_{\cdot,1:n} \boldsymbol{x} \quad \dots \quad f_\ell(\boldsymbol{x}, \boldsymbol{\rho}) = \boldsymbol{\rho}_{\cdot,0} + \boldsymbol{\rho}_{\cdot,1:n} \boldsymbol{x}$$

A single neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:n} \boldsymbol{x}$$

A composition of neurons with parameters $\boldsymbol{\theta}, \boldsymbol{\psi}, \boldsymbol{\rho}$

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:n} \boldsymbol{x} \quad f_2(\boldsymbol{x}, \boldsymbol{\psi}) = \boldsymbol{\psi}_{\cdot,0} + \boldsymbol{\psi}_{\cdot,1:n} \boldsymbol{x} \quad \dots \quad f_\ell(\boldsymbol{x}, \boldsymbol{\rho}) = \boldsymbol{\rho}_{\cdot,0} + \boldsymbol{\rho}_{\cdot,1:n} \boldsymbol{x}$$

$$f_\ell(\dots f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\psi}) \dots)$$

Written more plainly as

$$\boldsymbol{z}_1 = f_1(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:n}\boldsymbol{x}$$

Written more plainly as

$$\boldsymbol{z}_1 = f_1(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}_{\cdot, 0} + \boldsymbol{\theta}_{\cdot, 1:n} \boldsymbol{x}$$

$$\boldsymbol{z}_2 = f_2(\boldsymbol{z}_{\mathbf{1}}, \psi) = \boldsymbol{\psi}_{\cdot, 0} + \boldsymbol{\psi}_{\cdot, 1:n} \boldsymbol{z}_1$$

Written more plainly as

$$\boldsymbol{z}_1 = f_1(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:n} \boldsymbol{x}$$

$$\boldsymbol{z}_2 = f_2(\boldsymbol{z_1}, \psi) = \boldsymbol{\psi}_{\cdot,0} + \boldsymbol{\psi}_{\cdot,1:n} \boldsymbol{z}_1$$

$$\vdots$$

Written more plainly as

$$\boldsymbol{z}_1 = f_1(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:n} \boldsymbol{x}$$

$$\boldsymbol{z}_2 = f_2(\boldsymbol{z}_{\mathbf{1}}, \psi) = \boldsymbol{\psi}_{\cdot,0} + \boldsymbol{\psi}_{\cdot,1:n} \boldsymbol{z}_1$$

$$\vdots$$

$$\boldsymbol{y} = f_\ell(\boldsymbol{x}, \boldsymbol{\rho}) = \boldsymbol{\rho}_{\cdot,0} + \boldsymbol{\rho}_{\cdot,1:n} \boldsymbol{z}_{(d_x)-1}$$

Written more plainly as

$$\boldsymbol{z}_1 = f_1(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:n}\boldsymbol{x}$$

$$\boldsymbol{z}_2 = f_2(\boldsymbol{z}_{\mathbf{1}}, \psi) = \boldsymbol{\psi}_{\cdot,0} + \boldsymbol{\psi}_{\cdot,1:n}\boldsymbol{z}_1$$

$$\vdots$$

$$\boldsymbol{y} = f_\ell(\boldsymbol{x}, \boldsymbol{\rho}) = \boldsymbol{\rho}_{\cdot,0} + \boldsymbol{\rho}_{\cdot,1:n}\boldsymbol{z}_{(d_x)-1}$$

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

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

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

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?

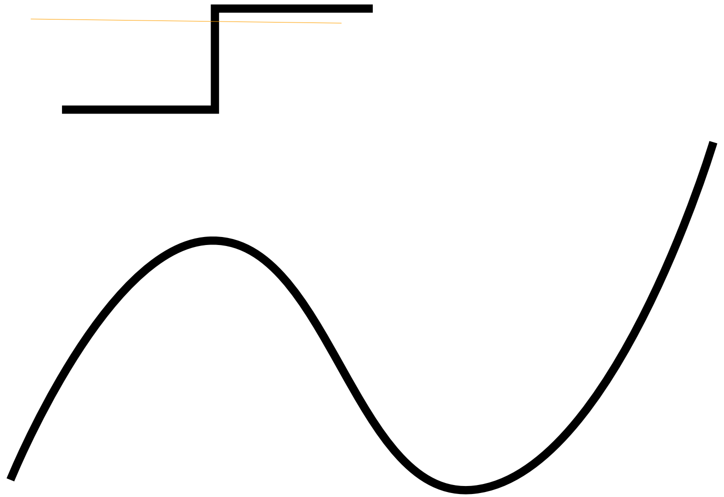
Consider a one-dimensional arbitrary function $g(x) = y$

We can approximate g using our neural network f

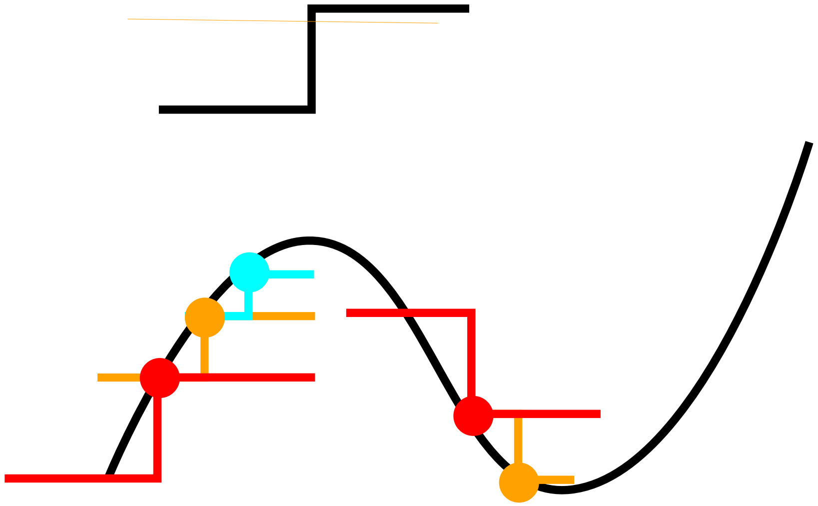
$$
\begin{aligned}
f(x_1, x_2, \boldsymbol{\theta}) = H\big(\theta_{3,0} & \\
+\theta_{3,1} & \quad \cdot \quad H\big(\theta_{1,0} + x_1 \theta_{1,1} + x_2 \theta_{1,2}\big) \\
+\theta_{3,2} & \quad \cdot \quad H\big(\theta_{2,0} + x_1 \theta_{2,1} + x_2 \theta_{2,2}\big)\big)
\end{aligned}
$$

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

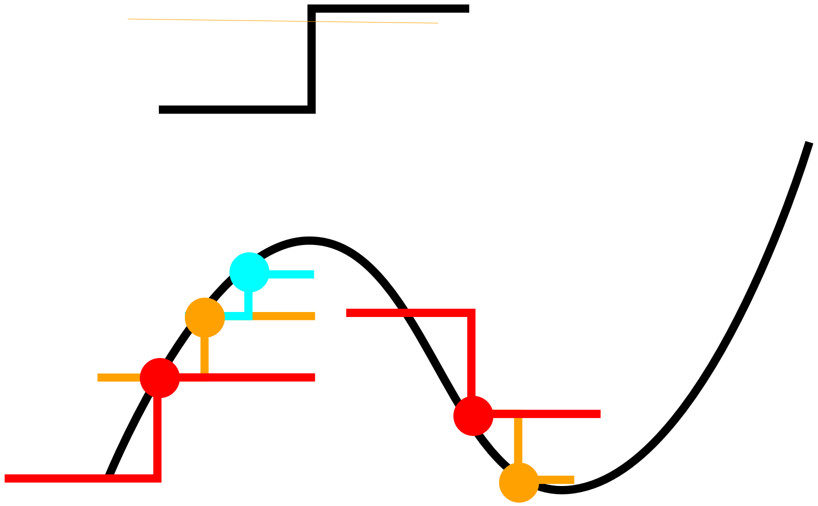
Proof Sketch: Approximate a function $g(x)$ using a linear combination of Heaviside functions



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



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

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

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

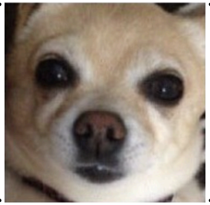
As we increase the width and depth of the network, ε shrinks

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.

All the models we examine in this course will use this neural network structure internally

All the models we examine in this course will use this neural network structure internally

- Transformers

All the models we examine in this course will use this neural network structure internally

- Transformers
- Graph neural networks

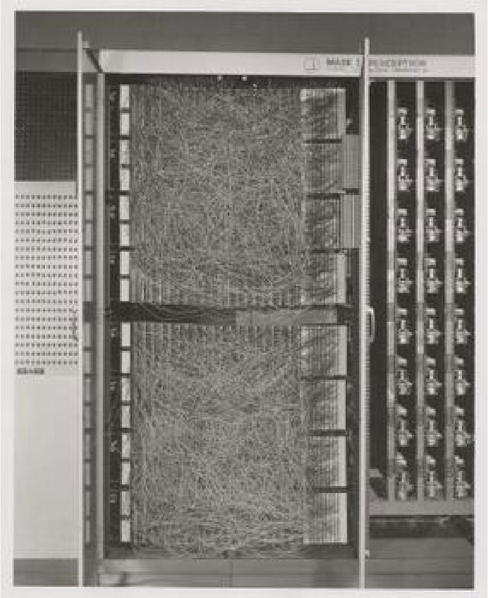
All the models we examine in this course will use this neural network structure internally

- Transformers
- Graph neural networks

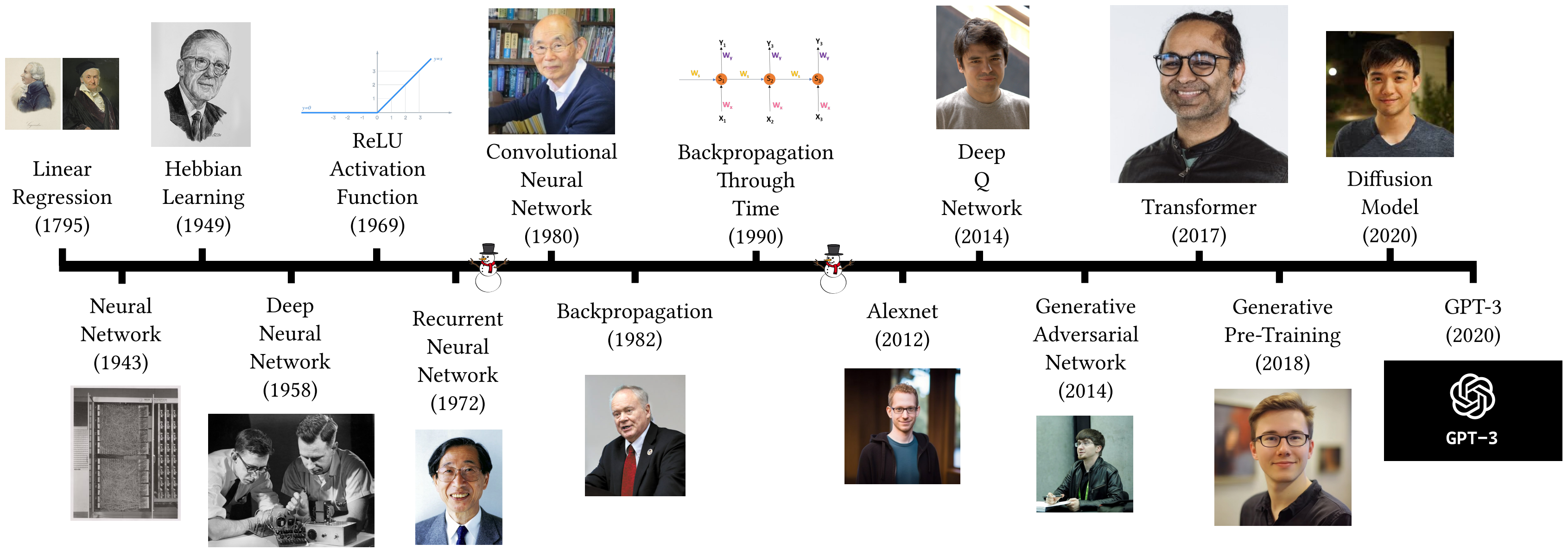
Relax

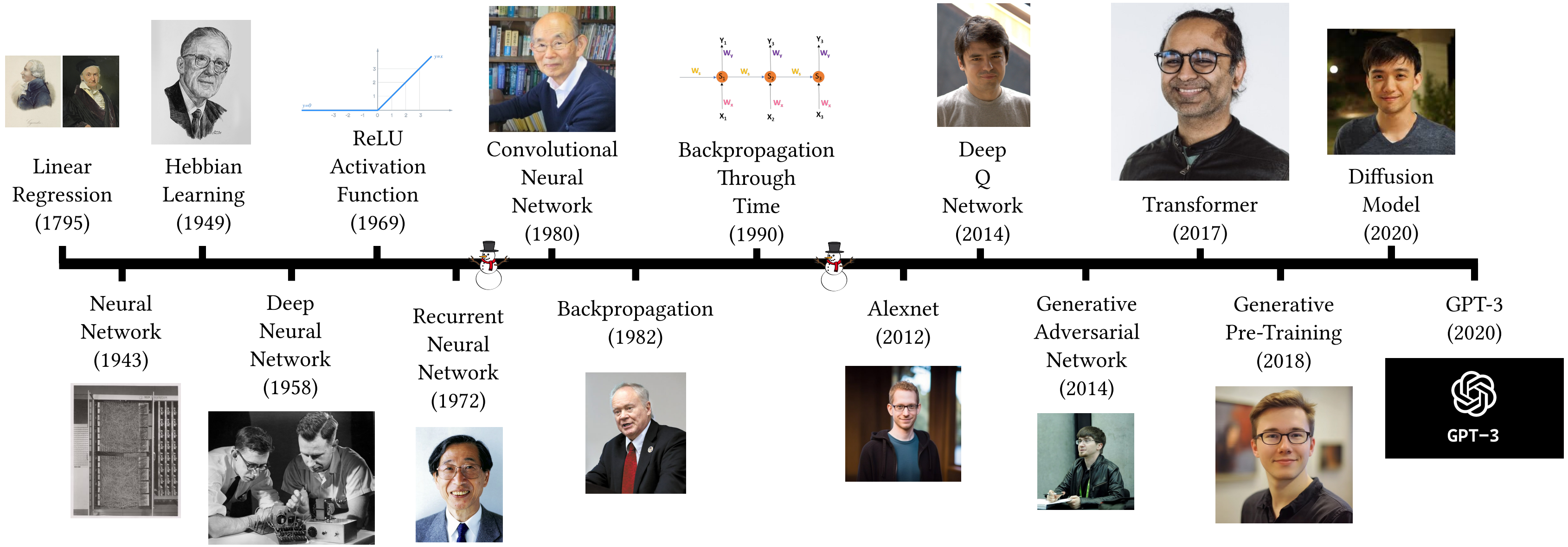
We call this form of a neural network a **feedforward network** or **perceptron** (invented in 1943)

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



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

Answer: Deep learning requires very deep and wide networks

1. Hardware advances enabled very deep and wide networks

Answer: Deep learning requires very deep and wide networks

1. Hardware advances enabled very deep and wide networks
2. Many theoretical improvements allow us to successfully train deeper and wider networks

Answer: Deep learning requires very deep and wide networks

1. Hardware advances enabled very deep and wide networks

2. Many theoretical improvements allow us to successfully train deeper and wider networks

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
- Corresponds to four parameter matrices in $\boldsymbol{\theta}$

Let us construct deep and wide neural networks in `torch` and `jax`

Here are the equations for one neural network layer

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\boldsymbol{\theta}_{\cdot, 0} + \boldsymbol{\theta}_{\cdot, 1:n} \boldsymbol{x}\right)$$

or

$$f(\boldsymbol{x}, (\boldsymbol{b}, \boldsymbol{W})) = \sigma(\boldsymbol{b} + \boldsymbol{W} \boldsymbol{x})$$

Here are the equations for one neural network layer

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:n}\boldsymbol{x}\right) \quad \text{or} \quad f(\boldsymbol{x}, (\boldsymbol{b}, \boldsymbol{W})) = \sigma(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{x})$$

We must implement the linear function $\boldsymbol{b} + \boldsymbol{W}\boldsymbol{x}$ and the activation function σ to create a neural network layer

Here are the equations for one neural network layer

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\boldsymbol{\theta}_{\cdot,0} + \boldsymbol{\theta}_{\cdot,1:n} \boldsymbol{x}\right) \quad \text{or} \quad f(\boldsymbol{x}, (\boldsymbol{b}, \boldsymbol{W})) = \sigma(\boldsymbol{b} + \boldsymbol{W} \boldsymbol{x})$$

We must implement the linear function $\boldsymbol{b} + \boldsymbol{W} \boldsymbol{x}$ and the activation function σ to create a neural network layer

Let us do this in colab! https://colab.research.google.com/drive/1bLtf3QY-yROIif_EoQSU1WS7svd0q8j7?usp=sharing