Optimization

CISC 7026: Introduction to Deep Learning

University of Macau

Previous Quiz

Quiz 1 grades are on moodle (mean 2.75 / 4)

Previous Quiz

Quiz 1 grades are on moodle (mean 2.75 / 4)

When I have a function called g that maps some inputs $a \in A, b \in B, c \in C$ to outputs $d \in D, e \in E$ I would write

$$g : A \times B \times C \mapsto D \times E$$

or

$$g : A, B, C \mapsto D, E$$

Previous Quiz

Quiz 1 grades are on moodle (mean 2.75 / 4)

When I have a function called g that maps some inputs $a \in A, b \in B, c \in C$ to outputs $d \in D, e \in E$ I would write

$$g : A \times B \times C \mapsto D \times E$$

or

$$g : A, B, C \mapsto D, E$$

In my code, I would write

$$d, e = g(a, b, c)$$

Previous Quiz

$\mathbb{R}$ is the set of real numbers

Previous Quiz

$\mathbb{R}$ is the set of real numbers

$5, \pi, e, -3.2, \frac{3}{2}, 1.3333...$ are all real numbers

Previous Quiz

$\mathbb{R}$ is the set of real numbers

$5, \pi, e, -3.2, \frac{3}{2}, 1.3333...$ are all real numbers

Unless the number is $\sqrt{-1}$, it is probably a real number

Previous Quiz

$\mathbb{R}$ is the set of real numbers

$5, \pi, e, -3.2, \frac{3}{2}, 1.3333...$ are all real numbers

Unless the number is $\sqrt{-1}$, it is probably a real number

When I write $\mathbb{R}^{d_x}$, this corresponds to d_x real numbers

Previous Quiz

$\mathbb{R}$ is the set of real numbers

$5, \pi, e, -3.2, \frac{3}{2}, 1.3333...$ are all real numbers

Unless the number is $\sqrt{-1}$, it is probably a real number

When I write $\mathbb{R}^{d_x}$, this corresponds to d_x real numbers

So $f : \mathbb{R}^{d_x} \mapsto \mathbb{R}^{d_y}$ is a function that maps d_x numbers to d_y numbers

Admin

Assignment one should be graded next week

Admin

Assignment one should be graded next week

Today's lecture will be very theoretical, but I must teach it

Admin

Assignment one should be graded next week

Today's lecture will be very theoretical, but I must teach it

I understand the homework might be difficult for some

Admin

Assignment one should be graded next week

Today’s lecture will be very theoretical, but I must teach it

I understand the homework might be difficult for some

Assignment 2 is **optional** (it is **not** required)

Admin

Assignment one should be graded next week

Today's lecture will be very theoretical, but I must teach it

I understand the homework might be difficult for some

Assignment 2 is **optional** (it is **not** required)

I will give n assignments and you will receive the highest $n - 1$ grades

Admin

Assignment one should be graded next week

Today's lecture will be very theoretical, but I must teach it

I understand the homework might be difficult for some

Assignment 2 is **optional** (it is **not** required)

I will give n assignments and you will receive the highest $n - 1$ grades

E.g., $\text{asg1} = 60$, $\text{asg2} = 90$, $\text{asg3} = 70$, total score $= \frac{160}{200}$

Admin

Assignment one should be graded next week

Today's lecture will be very theoretical, but I must teach it

I understand the homework might be difficult for some

Assignment 2 is **optional** (it is **not** required)

I will give n assignments and you will receive the highest $n - 1$ grades

E.g., $\text{asg1} = 60$, $\text{asg2} = 90$, $\text{asg3} = 70$, total score $= \frac{160}{200}$

E.g., $\text{asg1} = 60$, $\text{asg2} = 0$, $\text{asg3} = 70$, total score $= \frac{130}{200}$

Agenda

1. Review
2. Quiz
3. Optimization
4. Calculus review
5. Deriving linear regression
6. Gradient descent
7. Backpropagation
8. Coding

Agenda

1. **Review**
2. Quiz
3. Optimization
4. Calculus review
5. Deriving linear regression
6. Gradient descent
7. Backpropagation
8. Coding

Review

In lecture 1, we assumed a single-input system

Review

In lecture 1, we assumed a single-input system

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

Review

In lecture 1, we assumed a single-input system

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

But sometimes we want to consider multiple input dimensions

Review

In lecture 1, 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: $X \in \mathbb{R}^3$

Review

In lecture 1, 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

Review

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

Review

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

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

Review

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

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

Review

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

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

Review

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 solution is the same as before

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

Review

We combined **polynomial** and **multivariate** design matrices:

Review

We combined **polynomial** and **multivariate** design matrices:

One-dimensional polynomial functions

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

Review

We combined **polynomial** and **multivariate** design matrices:

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

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

Review

$$\boldsymbol{X}_D = \begin{bmatrix} \boldsymbol{x}_{D,[1]} & \cdots & \boldsymbol{x}_{D,[n]} \end{bmatrix}^\top$$

Review

$$\boldsymbol{X}_D = \begin{bmatrix} \boldsymbol{x}_{D,[1]} & \cdots & \boldsymbol{x}_{D,[n]} \end{bmatrix}^\top$$

$$\boldsymbol{x}_{D,[i]} =$$

$$\begin{bmatrix} \underbrace{x^m_{[i],d_x} x^m_{[i],d_x-1} \dots x^m_{[i],1}}_{(d_x \Rightarrow 1, x^m)} & \underbrace{x^m_{[i],d_x} x^m_{[i],d_x-1} \dots x^m_{[i],2}}_{(d_x \Rightarrow 2, x^m)} & \cdots & \underbrace{x^{m-1}_{[i],d_x} x^{m-1}_{[i],d_x-1} \dots x^m_{[i],1}}_{(d_x \Rightarrow 1, x^{m-1})} & \cdots \end{bmatrix}$$

Review

$$\boldsymbol{X}_D = \begin{bmatrix} \boldsymbol{x}_{D,[1]} & \cdots & \boldsymbol{x}_{D,[n]} \end{bmatrix}^\top$$

$$\boldsymbol{x}_{D,[i]} =$$

$$\begin{bmatrix} \underbrace{x_{[i],d_x}^m x_{[i],d_x-1}^m \dots x_{[i],1}^m}_{(d_x \Rightarrow 1, x^m)} & \underbrace{x_{[i],d_x}^m x_{[i],d_x-1}^m \dots x_{[i],2}^m}_{(d_x \Rightarrow 2, x^m)} & \cdots & \underbrace{x_{[i],d_x}^{m-1} x_{[i],d_x-1}^{m-1} \dots x_{[i],1}^m}_{(d_x \Rightarrow 1, x^{m-1})} & \cdots \end{bmatrix}$$

The resulting design matrix is too large to solve

Review

$$\boldsymbol{X}_D = \begin{bmatrix} \boldsymbol{x}_{D,[1]} & \cdots & \boldsymbol{x}_{D,[n]} \end{bmatrix}^\top$$

$$\boldsymbol{x}_{D,[i]} =$$

$$\begin{bmatrix} \underbrace{x_{[i],d_x}^m x_{[i],d_x-1}^m \dots x_{[i],1}^m}_{(d_x \Rightarrow 1, x^m)} & \underbrace{x_{[i],d_x}^m x_{[i],d_x-1}^m \dots x_{[i],2}^m}_{(d_x \Rightarrow 2, x^m)} & \cdots & \underbrace{x_{[i],d_x}^{m-1} x_{[i],d_x-1}^{m-1} \dots x_{[i],1}^m}_{(d_x \Rightarrow 1, x^{m-1})} & \cdots \end{bmatrix}$$

The resulting design matrix is too large to solve

We introduced neural networks because they scale to larger problems

Review

Brains and neural networks rely on **neurons**

Review

Brains and neural networks rely on **neurons**

Brain: Biological neurons $\rightarrow$ Biological neural network

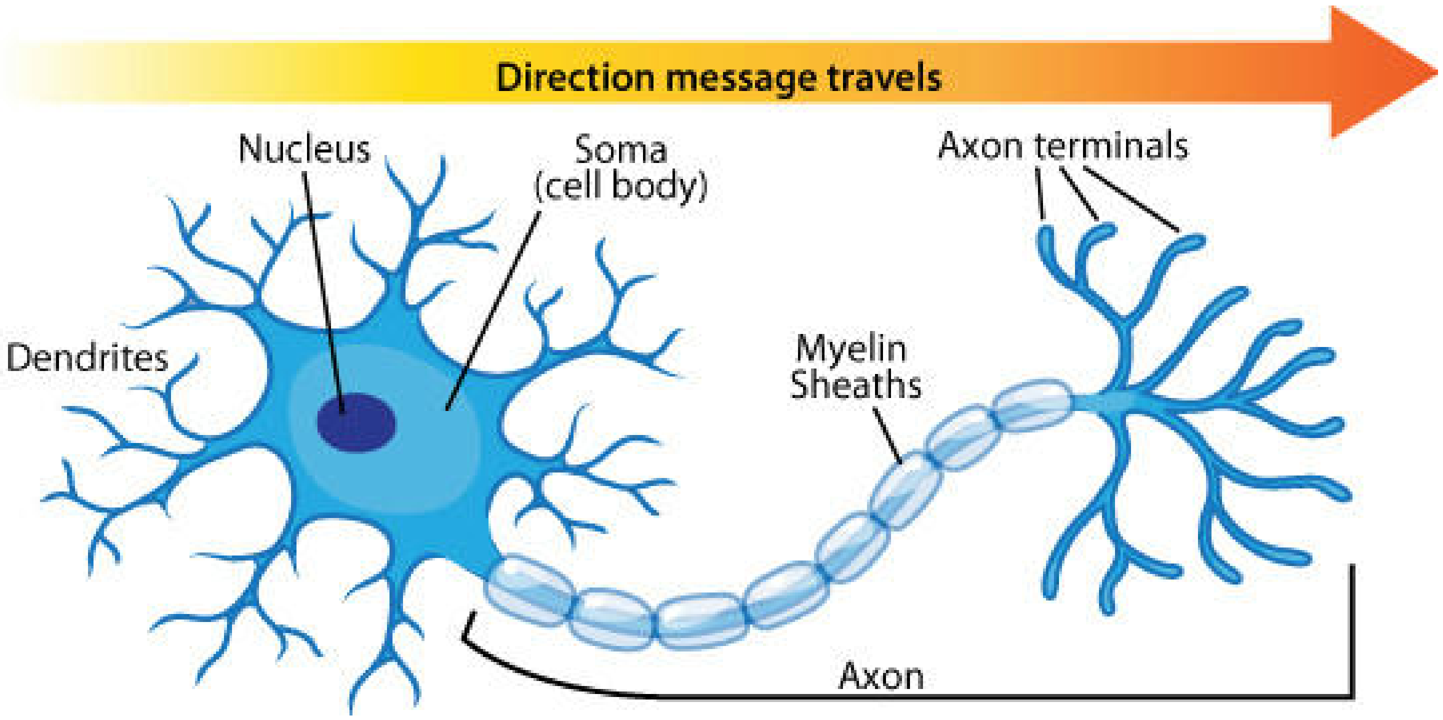
Review

Brains and neural networks rely on **neurons**

Brain: Biological neurons $\rightarrow$ Biological neural network

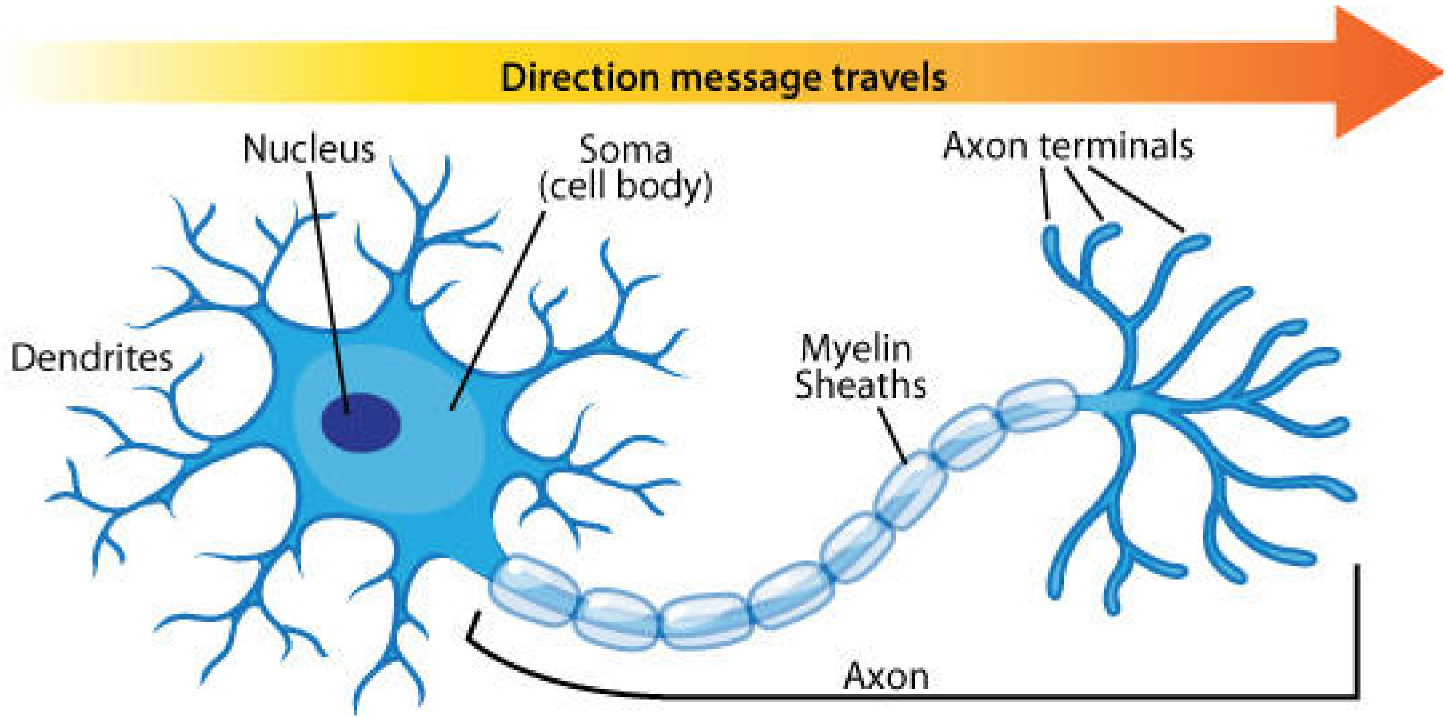
Computer: Artificial neurons $\rightarrow$ Artificial neural network

Review



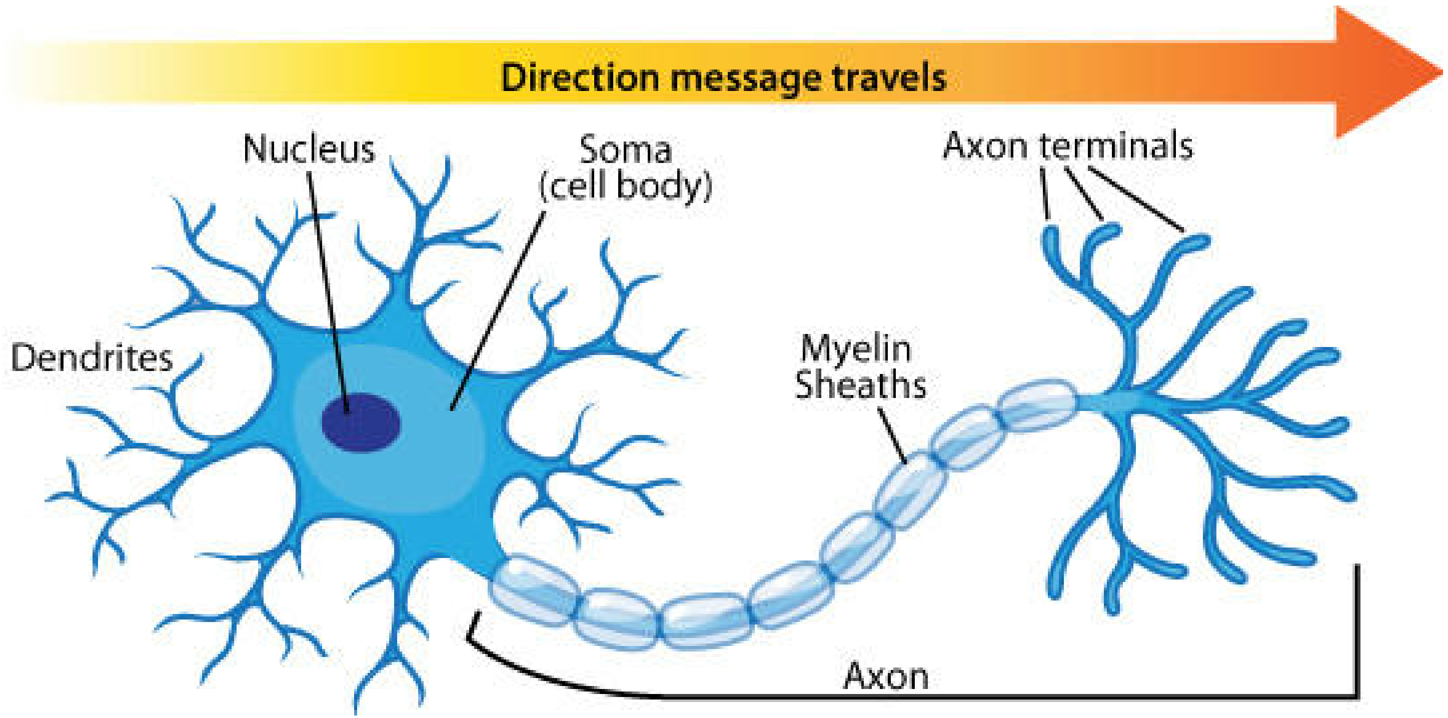
Neurons send messages based on messages received from other neurons

Review



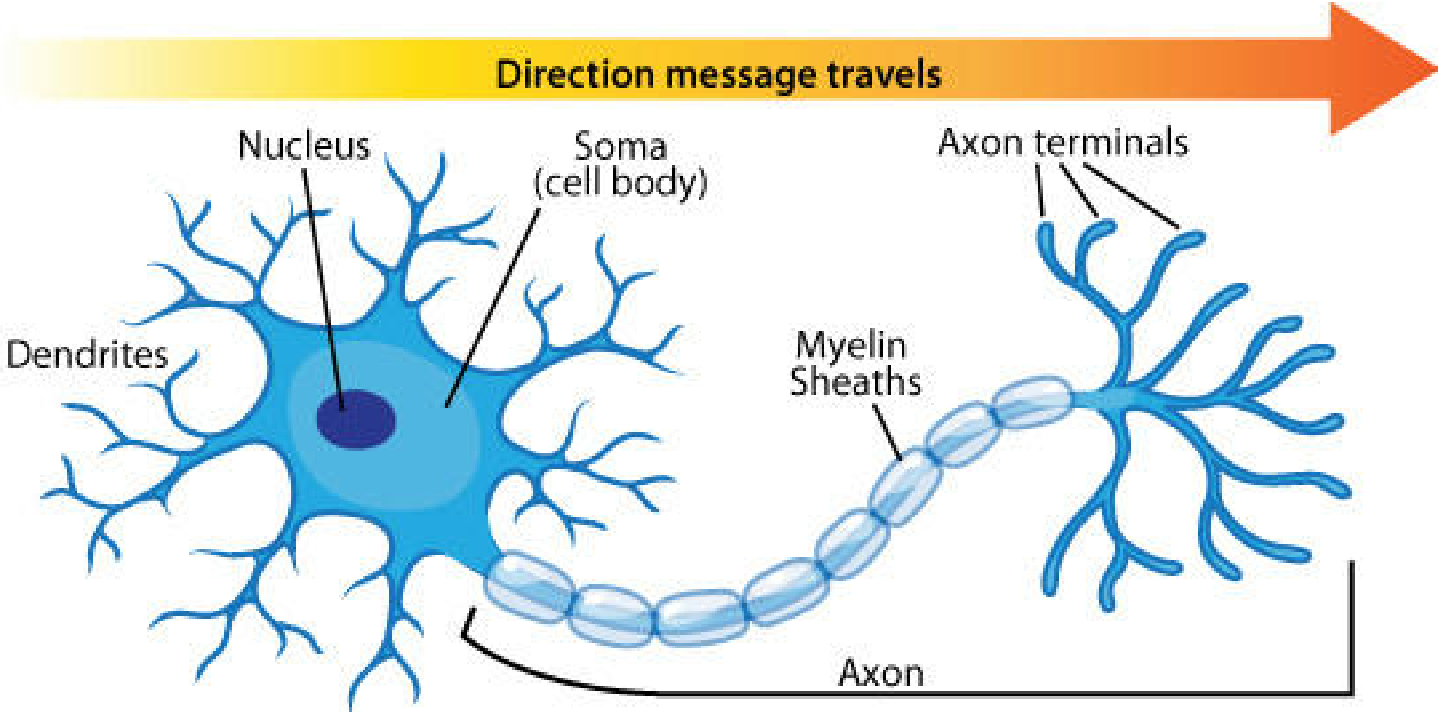
Incoming electrical signals travel along dendrites

Review



Electrical charges collect in the Soma (cell body)

Review



The axon outputs an electrical signal to other neurons

Review

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

Review

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

Dendrites form a parallel circuit

Review

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

Dendrites form a parallel circuit

In a parallel circuit, we can sum voltages together

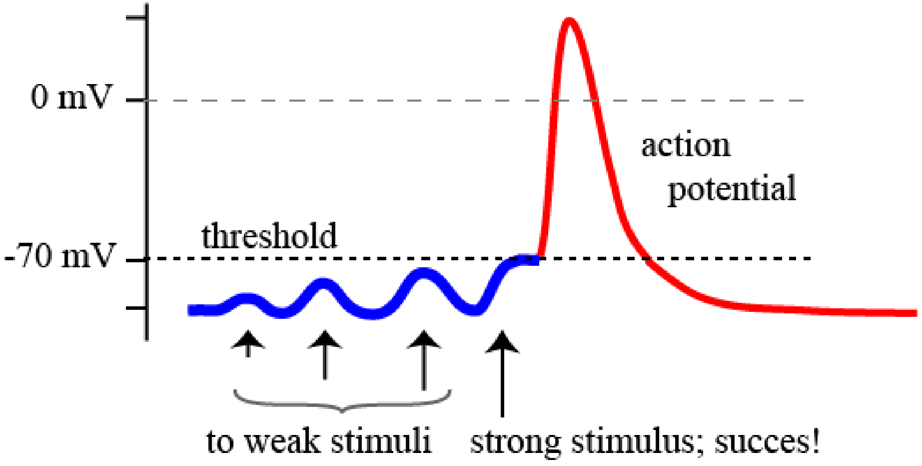
Review

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

Dendrites form a parallel circuit

In a parallel circuit, we can sum voltages together

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



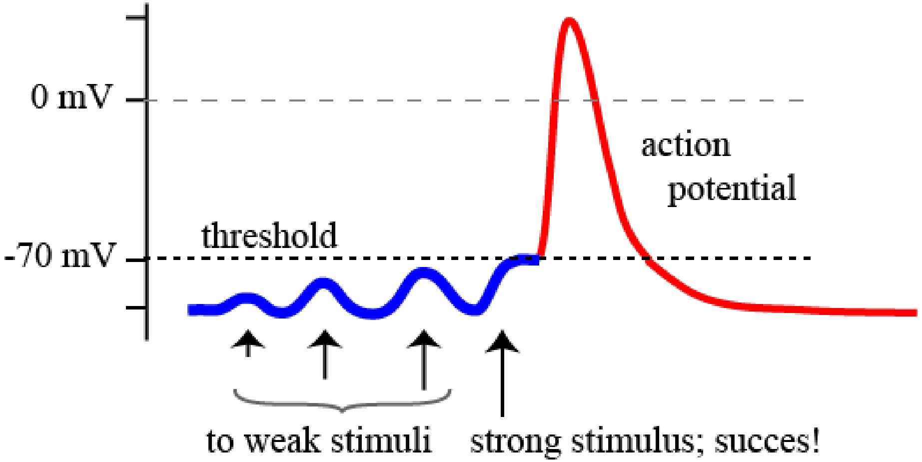
Review

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

Dendrites form a parallel circuit

In a parallel circuit, we can sum voltages together

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



Many active dendrites will add together and trigger an impulse

Review

We model the neuron “firing” using an activation function σ

Review

We model the neuron "firing" using an activation function σ

Last time, we used the heaviside step function as the activation function

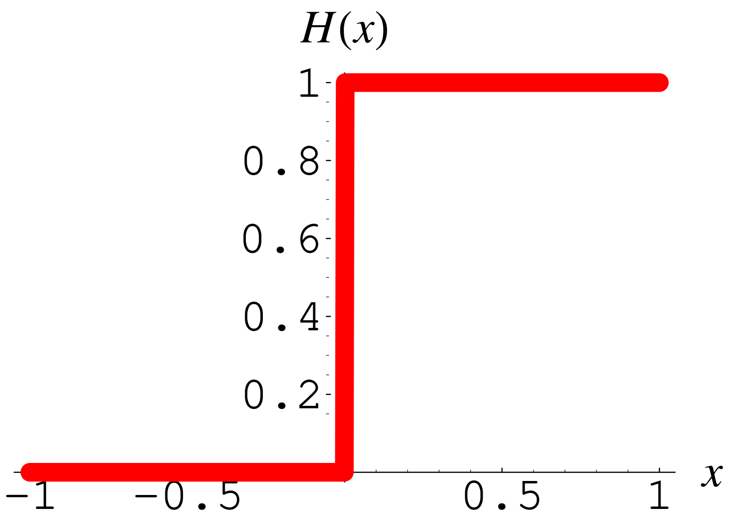
$$\sigma(x) = H(x) = \begin{cases} 0 \text{ if } x \leq 0 \\ 1 \text{ if } x > 0 \end{cases}$$

Review

We model the neuron "firing" using an activation function σ

Last time, we used the heaviside step function as the activation function

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



Review

We modeled a neuron mathematically, creating an artificial neuron

Review

We modeled a neuron mathematically, creating an artificial neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}); \quad \overline{\boldsymbol{x}} = \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}$$

Review

We modeled a neuron mathematically, creating an artificial neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}); \quad \overline{\boldsymbol{x}} = \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}$$

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

Review

We modeled a neuron mathematically, creating an artificial neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}); \quad \overline{\boldsymbol{x}} = \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}$$

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

An artificial neuron is a linear model with an activation function σ

Review

We modeled a neuron mathematically, creating an artificial neuron

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}); \quad \overline{\boldsymbol{x}} = \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}$$

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

An artificial neuron is a linear model with an activation function σ

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\underbrace{\theta_0 1 + \theta_1 x_1 + \ldots + \theta_{d_x} x_{d_x}}_{\text{Linear model}}\right)$$

Review

We can represent AND and OR boolean operators using a neuron

Review

We can represent AND and OR boolean operators using a neuron

But a neuron cannot represent many other functions

Review

We can represent AND and OR boolean operators using a neuron

But a neuron cannot represent many other functions

So we take many neurons and create a neural network

Review

We can represent AND and OR boolean operators using a neuron

But a neuron cannot represent many other functions

So we take many neurons and create a neural network

We discussed **wide** neural networks and **deep** neural networks

Review

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

Review

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

A single neuron:

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

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

Review

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):

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

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

Review

For a wide network (also called a layer):

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

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}); \quad \boldsymbol{\theta} \in \mathbb{R}^{(d_x+1) \times d_y}$$

Review

For a wide network (also called a layer):

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

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}); \quad \boldsymbol{\theta} \in \mathbb{R}^{(d_x+1) \times d_y}$$

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

Review

A **wide** neural network is also called a **layer**

Review

Review

A **wide** neural network is also called a **layer**

A layer is a linear operation and an activation function

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

Review

Many layers makes a deep neural network

$$\boldsymbol{z}_1 = f\left(\boldsymbol{x}, \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{W}_1 \end{bmatrix}\right)$$

$$\boldsymbol{z}_2 = f\left(\boldsymbol{z}_1, \begin{bmatrix} \boldsymbol{b}_2 \\ \boldsymbol{W}_2 \end{bmatrix}\right)$$

$$\boldsymbol{y} = f\left(\boldsymbol{z}_2, \begin{bmatrix} \boldsymbol{b}_2 \\ \boldsymbol{W}_2 \end{bmatrix}\right)$$

Agenda

1. **Review**
2. Quiz
3. Optimization
4. Calculus review
5. Deriving linear regression
6. Gradient descent
7. Backpropagation
8. Coding

Agenda

1. Review
2. **Quiz**
3. Optimization
4. Calculus review
5. Deriving linear regression
6. Gradient descent
7. Backpropagation
8. Coding

Quiz

Put away your phones and laptops

Quiz

Put away your phones and laptops

Take out a paper and pen, write your name and student ID

Quiz

Put away your phones and laptops

Take out a paper and pen, write your name and student ID

I will take away your quiz, give zero points, and refer you to the Dean if:

Quiz

Put away your phones and laptops

Take out a paper and pen, write your name and student ID

I will take away your quiz, give zero points, and refer you to the Dean if:

- Your phone or laptop is open during the quiz

Quiz

Put away your phones and laptops

Take out a paper and pen, write your name and student ID

I will take away your quiz, give zero points, and refer you to the Dean if:

- Your phone or laptop is open during the quiz
- You talk to your neighbor

Quiz

Put away your phones and laptops

Take out a paper and pen, write your name and student ID

I will take away your quiz, give zero points, and refer you to the Dean if:

- Your phone or laptop is open during the quiz
- You talk to your neighbor
- You look at your neighbor's quiz

Quiz

Put away your phones and laptops

Take out a paper and pen, write your name and student ID

I will take away your quiz, give zero points, and refer you to the Dean if:
- Your phone or laptop is open during the quiz
- You talk to your neighbor
- You look at your neighbor's quiz

After I explain the questions, you will have 15 minutes to finish the quiz

Agenda

1. Review
2. **Quiz**
3. Optimization
4. Calculus review
5. Deriving linear regression
6. Gradient descent
7. Backpropagation
8. Coding

Agenda

1. Review
2. Quiz
3. **Optimization**
4. Calculus review
5. Deriving linear regression
6. Gradient descent
7. Backpropagation
8. Coding

Optimization

We previously introduced neural networks as universal function approximators

Optimization

We previously introduced neural networks as universal function approximators

A deep neural network f can approximate **any** continuous function g to infinite precision

Optimization

We previously introduced neural networks as universal function approximators

A deep neural network f can approximate **any** continuous function g to infinite precision

$$f(x, \boldsymbol{\theta}) = g(x) + \varepsilon; \quad \varepsilon \to 0$$

Optimization

We previously introduced neural networks as universal function approximators

A deep neural network f can approximate **any** continuous function g to infinite precision

$$f(x, \boldsymbol{\theta}) = g(x) + \varepsilon; \quad \varepsilon \rightarrow 0$$

g can be a mapping from pictures to text, English to Chinese, etc

Optimization

We previously introduced neural networks as universal function approximators

A deep neural network f can approximate **any** continuous function g to infinite precision

$$f(x, \boldsymbol{\theta}) = g(x) + \varepsilon; \quad \varepsilon \to 0$$

g can be a mapping from pictures to text, English to Chinese, etc

But how do we find the $\boldsymbol{\theta}$ that makes $f(x, \boldsymbol{\theta}) = g(x) + \varepsilon$?

Optimization

We previously introduced neural networks as universal function approximators

A deep neural network f can approximate **any** continuous function g to infinite precision

$$f(x, \boldsymbol{\theta}) = g(x) + \varepsilon; \quad \varepsilon \to 0$$

g can be a mapping from pictures to text, English to Chinese, etc

But how do we find the $\boldsymbol{\theta}$ that makes $f(x, \boldsymbol{\theta}) = g(x) + \varepsilon$?

We said this $\boldsymbol{\theta}$ exists, but never said how to find it

Optimization

We previously introduced neural networks as universal function approximators

A deep neural network f can approximate **any** continuous function g to infinite precision

$$f(x, \boldsymbol{\theta}) = g(x) + \varepsilon; \quad \varepsilon \to 0$$

g can be a mapping from pictures to text, English to Chinese, etc

But how do we find the $\boldsymbol{\theta}$ that makes $f(x, \boldsymbol{\theta}) = g(x) + \varepsilon$?

We said this $\boldsymbol{\theta}$ exists, but never said how to find it

Goal: Find the parameters $\boldsymbol{\theta}$ a neural network

Optimization

When we search for $\boldsymbol{\theta}$, we call it **optimization** or **training**

Optimization

When we search for $\boldsymbol{\theta}$, we call it **optimization** or **training**

We optimize a loss function by computing

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$$

Optimization

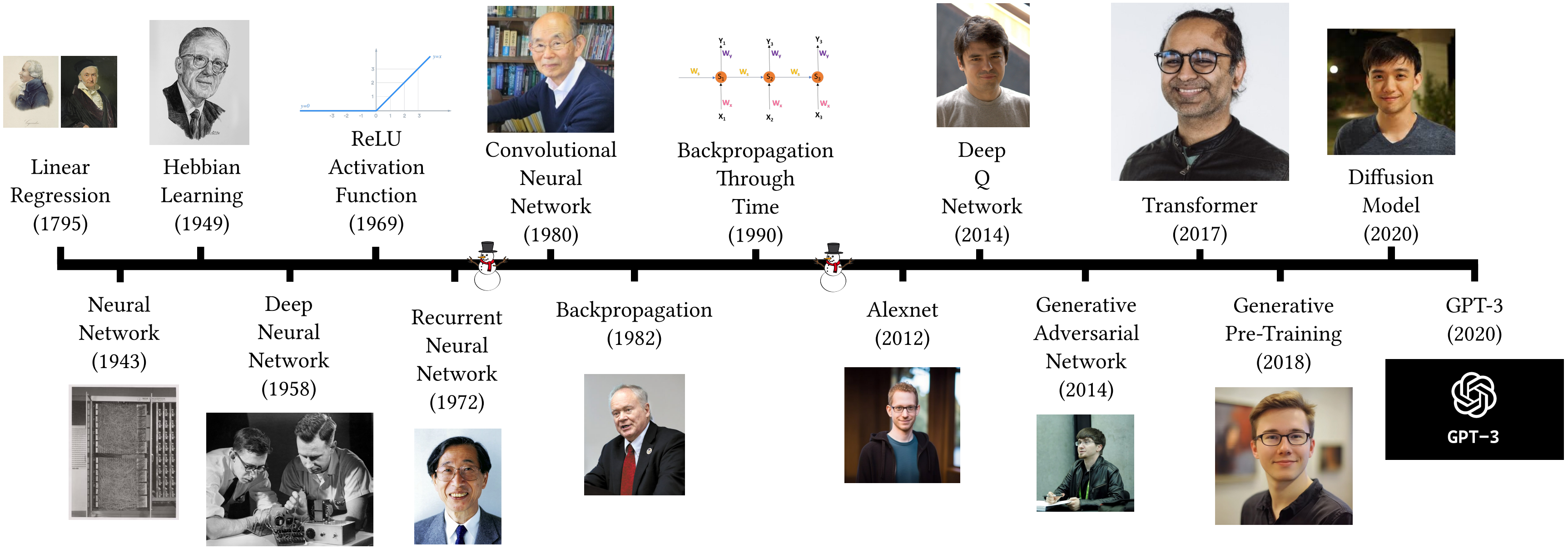
When we search for $\boldsymbol{\theta}$, we call it **optimization** or **training**

We optimize a loss function by computing

$$\arg \min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$$

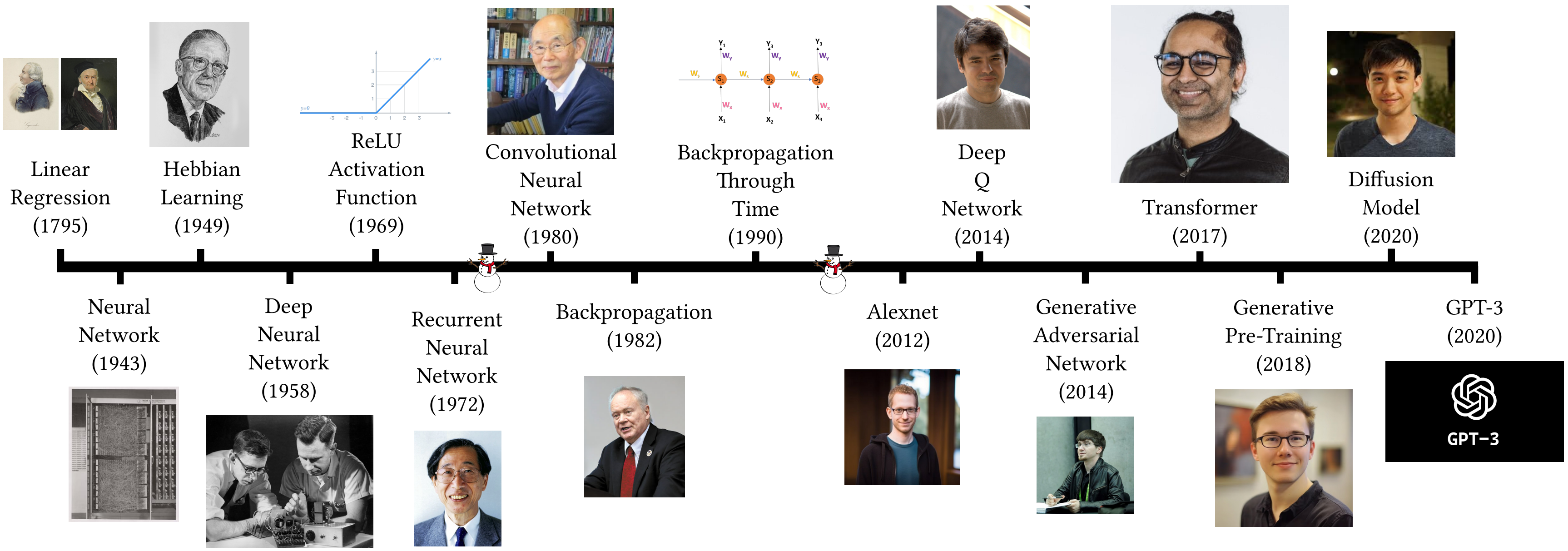
This expression looks very simple, but it can be very hard to evaluate

Optimization



Neural networks were discovered in 1943, but we could not train them until 1982!

Optimization



Neural networks were discovered in 1943, but we could not train them until 1982!

This is why theory is important

Optimization

Recall how we found $\boldsymbol{\theta}$ in the linear regression problem

Optimization

Recall how we found $\boldsymbol{\theta}$ in the linear regression problem

We define the square error loss function

Optimization

Recall how we found $\boldsymbol{\theta}$ in the linear regression problem

We define the square error loss function

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right) - \boldsymbol{y}_{[i]} \right)^2$$

Optimization

Recall how we found $\boldsymbol{\theta}$ in the linear regression problem

We define the square error loss function

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right) - \boldsymbol{y}_{[i]} \right)^2$$

Then, I gave you a magical solution to this optimization problem

Optimization

Recall how we found $\boldsymbol{\theta}$ in the linear regression problem

We define the square error loss function

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right) - \boldsymbol{y}_{[i]} \right)^2$$

Then, I gave you a magical solution to this optimization problem

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

Optimization

Recall how we found $\boldsymbol{\theta}$ in the linear regression problem

We define the square error loss function

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right) - \boldsymbol{y}_{[i]} \right)^2$$

Then, I gave you a magical solution to this optimization problem

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

Where does this solution come from? Can we do the same for neural networks?

Optimization

The solution for linear regression and neural networks comes from **calculus**

Optimization

The solution for linear regression and neural networks comes from **calculus**

The solution for neural networks also comes from calculus

Optimization

The solution for linear regression and neural networks comes from **calculus**

The solution for neural networks also comes from calculus

Let us review basic calculus concepts

Agenda

1. Review
2. Quiz
3. **Optimization**
4. Calculus review
5. Deriving linear regression
6. Gradient descent
7. Backpropagation
8. Coding

Agenda

1. Review
2. Quiz
3. Optimization
4. **Calculus review**
5. Deriving linear regression
6. Gradient descent
7. Backpropagation
8. Coding

Calculus Review

We write the **derivative** of a function f with respect to an input x as

$$f'(x) = \frac{d}{dx} f = \frac{df}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

Calculus Review

We write the **derivative** of a function f with respect to an input x as

$$f'(x) = \frac{d}{dx} f = \frac{df}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

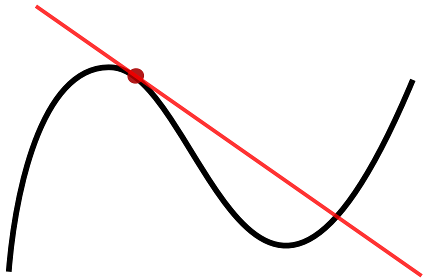
The derivative is the slope of a function

Calculus Review

We write the **derivative** of a function f with respect to an input x as

$$f'(x) = \frac{d}{dx} f = \frac{df}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

The derivative is the slope of a function

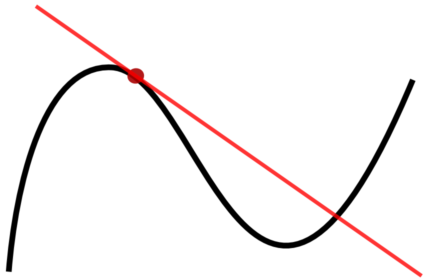


Calculus Review

We write the **derivative** of a function f with respect to an input x as

$$f'(x) = \frac{d}{dx} f = \frac{df}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

The derivative is the slope of a function



$$f(x), {\color{red} f'(x = a)}$$

Calculus Review

It is easiest if you treat the derivative as a **function of functions**

Calculus Review

It is easiest if you treat the derivative as a **function of functions**

$$\text{derivative}(f) = f'$$

Calculus Review

It is easiest if you treat the derivative as a **function of functions**

$$\text{derivative}(f) = f'$$

$$\frac{d}{dx} : f \mapsto f'$$

Calculus Review

It is easiest if you treat the derivative as a **function of functions**

$$\text{derivative}(f) = f'$$

$$\frac{d}{dx} : f \mapsto f'$$

The derivative takes a function f and outputs a new function f'

Calculus Review

It is easiest if you treat the derivative as a **function of functions**

$$\text{derivative}(f) = f'$$

$$\frac{d}{dx} : f \mapsto f'$$

The derivative takes a function f and outputs a new function f'

$$\frac{d}{dx} : [f : X \mapsto Y] \mapsto [f' : X \mapsto Y]$$

Calculus Review

There are formulas for computing the derivative of various operations

Calculus Review

There are formulas for computing the derivative of various operations

Constant $$\frac{d}{dx} c = 0$$

Calculus Review

There are formulas for computing the derivative of various operations

Constant $$\frac{d}{dx} c = 0$$

Power $$\frac{d}{dx} x^n = n x^{n-1}$$

Calculus Review

Sum/Difference

$$\frac{d}{dx}[f(x) + g(x)] = f'(x) + g'(x)$$

Calculus Review

Sum/Difference $$\frac{d}{dx}[f(x) + g(x)] = f'(x) + g'(x)$$

Product $$\frac{d}{dx}[f(x)g(x)] = f'(x)g(x) + g'(x)f(x)$$

Calculus Review

Sum/Difference $$\frac{d}{dx}[f(x) + g(x)] = f'(x) + g'(x)$$

Product $$\frac{d}{dx}[f(x)g(x)] = f'(x)g(x) + g'(x)f(x)$$

Chain $$\frac{d}{dx}[f(g(x))] = f'(g(x))g'(x)$$

Calculus Review

For example, consider the function

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

Calculus Review

For example, consider the function

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

We can write the derivative as

$$\frac{d}{dx}[f(x)] = 2x - 3$$

Calculus Review

For example, consider the function

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

We can write the derivative as

$$\frac{d}{dx}[f(x)] = 2x - 3$$

We can evaluate the derivative at specific points

Calculus Review

For example, consider the function

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

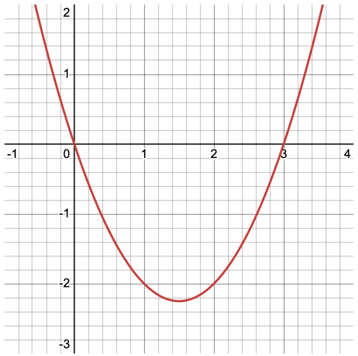
We can write the derivative as

$$\frac{d}{dx}[f(x)] = 2x - 3$$

We can evaluate the derivative at specific points

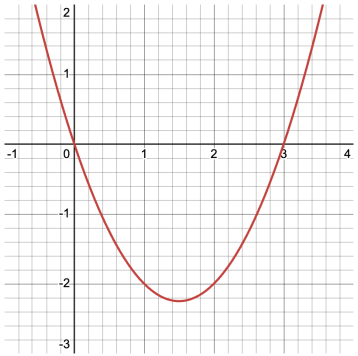
$$\frac{d}{dx}[f](1) = 2 \cdot 1 - 3 = -1$$

Calculus Review

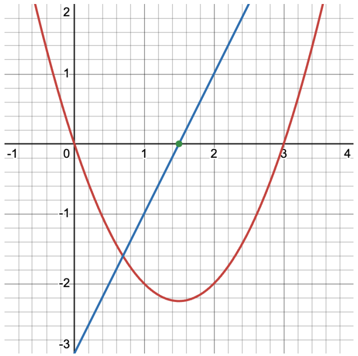


$$f(x) = x^2 - 3x$$

Calculus Review

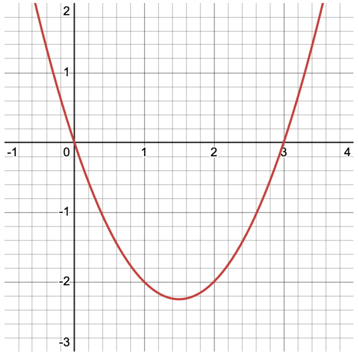


$$f(x) = x^2 - 3x$$

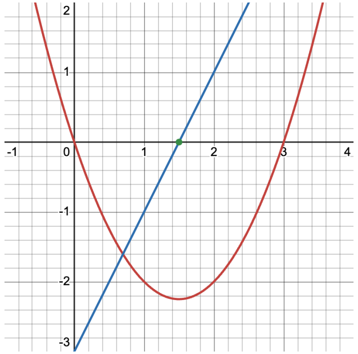


$$\frac{df}{dx} = 2x - 3$$

Calculus Review



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



$$\frac{df}{dx} = 2x - 3$$

$$0 = 2x - 3 \quad \Rightarrow \quad x = \frac{3}{2}$$

Calculus Review

We can expand the definition of derivative to multivariate functions. We call this the **gradient**

$$\nabla_{\boldsymbol{x}} f\left(\begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^\top\right) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \dots & \frac{\partial f}{\partial x_n} \end{bmatrix}^\top$$

Calculus Review

We can expand the definition of derivative to multivariate functions. We call this the **gradient**

$$\nabla_{\boldsymbol{x}} f\left(\begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^\top\right) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \dots & \frac{\partial f}{\partial x_n} \end{bmatrix}^\top$$

Partial derivatives behave similarly to standard derivatives

Calculus Review

We can expand the definition of derivative to multivariate functions. We call this the **gradient**

$$\nabla_{\boldsymbol{x}} f\left(\begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^\top\right) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \dots & \frac{\partial f}{\partial x_n} \end{bmatrix}^\top$$

Partial derivatives behave similarly to standard derivatives

$$\frac{\partial}{\partial x_1} f(x_1, ..., x_n) \approx \frac{d}{d x_1} f(x_1, ..., x_n)$$

Calculus Review

We can expand the definition of derivative to multivariate functions. We call this the **gradient**

$$\nabla_{\boldsymbol{x}} f\left(\begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^\top\right) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \dots & \frac{\partial f}{\partial x_n} \end{bmatrix}^\top$$

Partial derivatives behave similarly to standard derivatives

$$\frac{\partial}{\partial x_1} f(x_1, ..., x_n) \approx \frac{d}{d x_1} f(x_1, ..., x_n)$$

When computing $\frac{\partial}{\partial x_i} f(x_1, ..., x_n)$, we treat $x_1, ..., x_{i-1}, x_{i+1}, ..., x_n$ as constant

Calculus Review

For example, consider the function

$$f(x_1, x_2) = x_1^2 - 3x_1 x_2$$

Calculus Review

For example, consider the function

$$f(x_1, x_2) = x_1^2 - 3x_1 x_2$$

We can write the gradient as

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \nabla_{x_1, x_2} f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x_1, x_2) \\ \frac{\partial}{\partial x_2} f(x_1, x_2) \end{bmatrix} = \begin{bmatrix} 2x_1 - 3x_2 \\ -3x_1 \end{bmatrix}$$

Calculus Review

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \nabla_{x_1, x_2} f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x_1, x_2) \\ \frac{\partial}{\partial x_2} f(x_1, x_2) \end{bmatrix} = \begin{bmatrix} 2x_1 - 3x_2 \\ -3x_1 \end{bmatrix}$$

Calculus Review

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \nabla_{x_1, x_2} f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x_1, x_2) \\ \frac{\partial}{\partial x_2} f(x_1, x_2) \end{bmatrix} = \begin{bmatrix} 2x_1 - 3x_2 \\ -3x_1 \end{bmatrix}$$

We can evaluate the gradient at specific points

Calculus Review

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \nabla_{x_1, x_2} f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x_1, x_2) \\ \frac{\partial}{\partial x_2} f(x_1, x_2) \end{bmatrix} = \begin{bmatrix} 2x_1 - 3x_2 \\ -3x_1 \end{bmatrix}$$

We can evaluate the gradient at specific points

$$\nabla_{\boldsymbol{x}} f\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) = \nabla_{x_1, x_2} f\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(1, 0) \\ \frac{\partial}{\partial x_2} f(1, 0) \end{bmatrix} = \begin{bmatrix} 2 \cdot 1 - 3 \cdot 0 \\ -3 \cdot 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -3 \end{bmatrix}$$

Calculus Review

In calculus, we can find the local extrema of a function $f(x)$ by finding where the derivative is zero

$$f'(x) = \frac{d}{dx} f(x) = 0$$

Calculus Review

In calculus, we can find the local extrema of a function $f(x)$ by finding where the derivative is zero

$$f'(x) = \frac{d}{dx} f(x) = 0$$

With a multivariate function, the extrema lies where the gradient is zero

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \dots & \frac{\partial f}{\partial x_n} \end{bmatrix}^\top = \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix}^\top$$

Agenda

1. Review
2. Quiz
3. Optimization
4. **Calculus review**
5. Deriving linear regression
6. Gradient descent
7. Backpropagation
8. Coding

Agenda

1. Review
2. Quiz
3. Optimization
4. Calculus review
5. **Deriving linear regression**
6. Gradient descent
7. Backpropagation
8. Coding

Deriving Linear Regression

Now that we remember calculus, let us revisit linear regression

Deriving Linear Regression

Now that we remember calculus, let us revisit linear regression

If we can derive the solution for linear regression, maybe we can apply it to deep neural networks

Deriving Linear Regression

In linear regression, our loss function is

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta} \right) - \boldsymbol{y}_{[i]} \right)^2$$

Deriving Linear Regression

In linear regression, our loss function is

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right) - \boldsymbol{y}_{[i]} \right)^2$$

We can write the square error loss in matrix form as

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})^\top (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})$$

Deriving Linear Regression

In linear regression, our loss function is

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right) - \boldsymbol{y}_{[i]} \right)^2$$

We can write the square error loss in matrix form as

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})^\top (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})$$

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \underbrace{\underbrace{(\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})^\top}_{\text{Linear function of } \theta} \underbrace{(\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})}_{\text{Linear function of } \theta}}_{\text{Quadratic function of } \theta}$$

Deriving Linear Regression

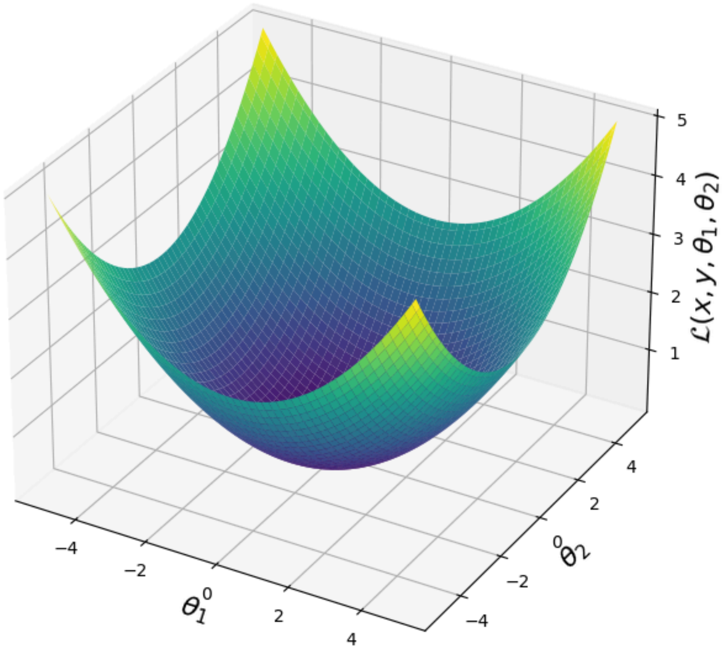
$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \underbrace{\underbrace{(\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})^\top}_{\text{Linear function of } \theta} \underbrace{(\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})}_{\text{Linear function of } \theta}}_{\text{Quadratic function of } \theta}$$

Deriving Linear Regression

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \underbrace{\underbrace{(\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})^\top}_{\text{Linear function of } \theta} \underbrace{(\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})}_{\text{Linear function of } \theta}}_{\text{Quadratic function of } \theta}$$

A quadratic function has a single minima! The minima must be at

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = 0$$



Deriving Linear Regression

Therefore, we know that the $\boldsymbol{\theta}$ that solves

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = 0$$

Deriving Linear Regression

Therefore, we know that the $\boldsymbol{\theta}$ that solves

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = 0$$

Also solves

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$$

Deriving Linear Regression

Using calculus, let us derive the solution to linear regression

Deriving Linear Regression

Using calculus, let us derive the solution to linear regression

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})^\top (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})$$

Deriving Linear Regression

Using calculus, let us derive the solution to linear regression

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})^\top (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \left[(\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})^\top (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta}) \right]$$

Deriving Linear Regression

Using calculus, let us derive the solution to linear regression

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})^\top (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \left[(\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})^\top (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta}) \right]$$

$$= \nabla_{\boldsymbol{\theta}} \left[\boldsymbol{Y}^\top \boldsymbol{Y} - \boldsymbol{Y}^\top \boldsymbol{X}_D \boldsymbol{\theta} - (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{Y} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D \boldsymbol{\theta} \right]$$

Deriving Linear Regression

Using calculus, let us derive the solution to linear regression

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})^\top (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \left[(\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta})^\top (\boldsymbol{Y} - \boldsymbol{X}_D \boldsymbol{\theta}) \right]$$

$$= \nabla_{\boldsymbol{\theta}} \left[\boldsymbol{Y}^\top \boldsymbol{Y} - \boldsymbol{Y}^\top \boldsymbol{X}_D \boldsymbol{\theta} - (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{Y} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D \boldsymbol{\theta} \right]$$

$$= \boldsymbol{0} - \boldsymbol{Y}^\top \boldsymbol{X}_D \boldsymbol{I} - (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{Y} + (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{X}_D \boldsymbol{\theta} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D \boldsymbol{I}$$

Deriving Linear Regression

$$= \mathbf{0} - \boldsymbol{Y}^\top \boldsymbol{X}_D \boldsymbol{I} - (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{Y} + (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{X}_D \boldsymbol{\theta} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D \boldsymbol{I}$$

Deriving Linear Regression

$$= \mathbf{0} - \boldsymbol{Y}^\top \boldsymbol{X}_D \boldsymbol{I} - (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{Y} + (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{X}_D \boldsymbol{\theta} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D \boldsymbol{I}$$

$$= -\boldsymbol{Y}^\top \boldsymbol{X}_D - \boldsymbol{X}_D^\top \boldsymbol{Y} + \boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D$$

Deriving Linear Regression

$$= \mathbf{0} - \boldsymbol{Y}^\top \boldsymbol{X}_D \boldsymbol{I} - (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{Y} + (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{X}_D \boldsymbol{\theta} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D \boldsymbol{I}$$

$$= -\boldsymbol{Y}^\top \boldsymbol{X}_D - \boldsymbol{X}_D^\top \boldsymbol{Y} + \boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D$$

Remember, $(\boldsymbol{A}\boldsymbol{B})^\top = \boldsymbol{B}^\top \boldsymbol{A}^\top$, and so $\boldsymbol{Y}^\top \boldsymbol{X}_D = \boldsymbol{Y}^\top \left(\boldsymbol{X}_D^\top\right)^\top = \boldsymbol{X}_D^\top \boldsymbol{Y}$

Deriving Linear Regression

$$= \mathbf{0} - \boldsymbol{Y}^\top \boldsymbol{X}_D \boldsymbol{I} - (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{Y} + (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{X}_D \boldsymbol{\theta} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D \boldsymbol{I}$$

$$= -\boldsymbol{Y}^\top \boldsymbol{X}_D - \boldsymbol{X}_D^\top \boldsymbol{Y} + \boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D$$

Remember, $(\boldsymbol{A}\boldsymbol{B})^\top = \boldsymbol{B}^\top \boldsymbol{A}^\top$, and so $\boldsymbol{Y}^\top \boldsymbol{X}_D = \boldsymbol{Y}^\top \left(\boldsymbol{X}_D^\top\right)^\top = \boldsymbol{X}_D^\top \boldsymbol{Y}$

$$= -\boldsymbol{Y}^\top \boldsymbol{X}_D - \boldsymbol{Y}^\top \boldsymbol{X}_D + \boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta} + \boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

Deriving Linear Regression

$$= \mathbf{0} - \boldsymbol{Y}^\top \boldsymbol{X}_D \boldsymbol{I} - (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{Y} + (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{X}_D \boldsymbol{\theta} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D \boldsymbol{I}$$

$$= -\boldsymbol{Y}^\top \boldsymbol{X}_D - \boldsymbol{X}_D^\top \boldsymbol{Y} + \boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D$$

Remember, $(\boldsymbol{A}\boldsymbol{B})^\top = \boldsymbol{B}^\top \boldsymbol{A}^\top$, and so $\boldsymbol{Y}^\top \boldsymbol{X}_D = \boldsymbol{Y}^\top (\boldsymbol{X}_D^\top)^\top = \boldsymbol{X}_D^\top \boldsymbol{Y}$

$$= -\boldsymbol{Y}^\top \boldsymbol{X}_D - \boldsymbol{Y}^\top \boldsymbol{X}_D + \boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta} + \boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

$$= -2\boldsymbol{X}_D^\top \boldsymbol{Y} + 2\boldsymbol{X}_D^\top \boldsymbol{X} \boldsymbol{\theta}$$

Deriving Linear Regression

$$= \mathbf{0} - \boldsymbol{Y}^\top \boldsymbol{X}_D \boldsymbol{I} - (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{Y} + (\boldsymbol{X}_D \boldsymbol{I})^\top \boldsymbol{X}_D \boldsymbol{\theta} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D \boldsymbol{I}$$

$$= -\boldsymbol{Y}^\top \boldsymbol{X}_D - \boldsymbol{X}_D^\top \boldsymbol{Y} + \boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta} + (\boldsymbol{X}_D \boldsymbol{\theta})^\top \boldsymbol{X}_D$$

Remember, $(\boldsymbol{A}\boldsymbol{B})^\top = \boldsymbol{B}^\top \boldsymbol{A}^\top$, and so $\boldsymbol{Y}^\top \boldsymbol{X}_D = \boldsymbol{Y}^\top (\boldsymbol{X}_D^\top)^\top = \boldsymbol{X}_D^\top \boldsymbol{Y}$

$$= -\boldsymbol{Y}^\top \boldsymbol{X}_D - \boldsymbol{Y}^\top \boldsymbol{X}_D + \boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta} + \boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

$$= -2\boldsymbol{X}_D^\top \boldsymbol{Y} + 2\boldsymbol{X}_D^\top \boldsymbol{X} \boldsymbol{\theta}$$

And so, the gradient of the loss is

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = -2\boldsymbol{X}_D^\top \boldsymbol{Y} + 2\boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

Deriving Linear Regression

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = -2 \boldsymbol{X}_D^\top \boldsymbol{Y} + 2 \boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

Deriving Linear Regression

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = -2\boldsymbol{X}_D^\top \boldsymbol{Y} + 2\boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

We want to find the $\boldsymbol{\theta}$ that makes the gradient of the loss zero

Deriving Linear Regression

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = -2\boldsymbol{X}_D^\top \boldsymbol{Y} + 2\boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

We want to find the $\boldsymbol{\theta}$ that makes the gradient of the loss zero

$$\boldsymbol{0} = -2\boldsymbol{X}_D^\top \boldsymbol{Y} + 2\boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

Deriving Linear Regression

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = -2\boldsymbol{X}_D^\top \boldsymbol{Y} + 2\boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

We want to find the $\boldsymbol{\theta}$ that makes the gradient of the loss zero

$$\boldsymbol{0} = -2\boldsymbol{X}_D^\top \boldsymbol{Y} + 2\boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

$$2\boldsymbol{X}_D^\top \boldsymbol{Y} = 2\boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

Deriving Linear Regression

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = -2\boldsymbol{X}_D^\top \boldsymbol{Y} + 2\boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

We want to find the $\boldsymbol{\theta}$ that makes the gradient of the loss zero

$$\boldsymbol{0} = -2\boldsymbol{X}_D^\top \boldsymbol{Y} + 2\boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

$$2\boldsymbol{X}_D^\top \boldsymbol{Y} = 2\boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

$$\boldsymbol{X}_D^\top \boldsymbol{Y} = \boldsymbol{X}_D^\top \boldsymbol{X} \boldsymbol{\theta}$$

Deriving Linear Regression

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = -2\boldsymbol{X}_D^\top \boldsymbol{Y} + 2\boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

We want to find the $\boldsymbol{\theta}$ that makes the gradient of the loss zero

$$\boldsymbol{0} = -2\boldsymbol{X}_D^\top \boldsymbol{Y} + 2\boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

$$2\boldsymbol{X}_D^\top \boldsymbol{Y} = 2\boldsymbol{X}_D^\top \boldsymbol{X}_D \boldsymbol{\theta}$$

$$\boldsymbol{X}_D^\top \boldsymbol{Y} = \boldsymbol{X}_D^\top \boldsymbol{X} \boldsymbol{\theta}$$

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

Deriving Linear Regression

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

Deriving Linear Regression

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

This was the "magic" solution I gave you for linear regression

Deriving Linear Regression

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

This was the "magic" solution I gave you for linear regression

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

Deriving Linear Regression

Great! We derived the solution to linear regression

Deriving Linear Regression

Great! We derived the solution to linear regression

Now, we will do the same approach for neural networks

Deriving Linear Regression

Great! We derived the solution to linear regression

Now, we will do the same approach for neural networks

To make it simple, we assume $d_x = 1, d_y = 1, n = 1$

Deriving Linear Regression

Great! We derived the solution to linear regression

Now, we will do the same approach for neural networks

To make it simple, we assume $d_x = 1, d_y = 1, n = 1$

One input dimension, one output dimension, one datapoint

Deriving Linear Regression

Great! We derived the solution to linear regression

Now, we will do the same approach for neural networks

To make it simple, we assume $d_x = 1, d_y = 1, n = 1$

One input dimension, one output dimension, one datapoint

Step 1: Write the loss function for a neural network

Deriving Linear Regression

Like linear regression, we can use square error for a neural network

Deriving Linear Regression

Like linear regression, we can use square error for a neural network

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

Deriving Linear Regression

Like linear regression, we can use square error for a neural network

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

All that changes is the model f

Deriving Linear Regression

Like linear regression, we can use square error for a neural network

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

All that changes is the model f

Linear regression: $$f(x, y, \boldsymbol{\theta}) = \theta_0 + \theta_1 x$$

Deriving Linear Regression

Like linear regression, we can use square error for a neural network

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

All that changes is the model f

Linear regression: $$f(x, y, \boldsymbol{\theta}) = \theta_0 + \theta_1 x$$

Perceptron: $$f(x, y, \boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x)$$

Deriving Linear Regression

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

Loss function

Deriving Linear Regression

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

Loss function

$$f(x, \boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x)$$

Neural network model

Deriving Linear Regression

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

Loss function

$$f(x, \boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x)$$

Neural network model

Now, we plug the model f into the loss function

Deriving Linear Regression

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

Loss function

$$f(x, \boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x)$$

Neural network model

Now, we plug the model f into the loss function

$$\mathcal{L}(x, y, \boldsymbol{\theta}) = (\sigma(\theta_0 + \theta_1 x) - y)^2$$

Deriving Linear Regression

$$\mathcal{L}(x, y, \boldsymbol{\theta}) = (f(x, \boldsymbol{\theta}) - y)^2$$ Loss function

$$f(x, \boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x)$$ Neural network model

Now, we plug the model f into the loss function

$$\mathcal{L}(x, y, \boldsymbol{\theta}) = (\sigma(\theta_0 + \theta_1 x) - y)^2$$

Rewrite

Deriving Linear Regression

$$\mathcal{L}(x, y, \boldsymbol{\theta}) = (f(x, \boldsymbol{\theta}) - y)^2$$ Loss function

$$f(x, \boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x)$$ Neural network model

Now, we plug the model f into the loss function

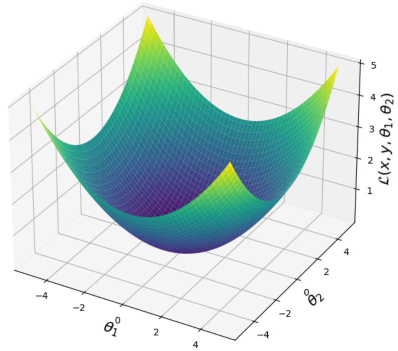
$$\mathcal{L}(x, y, \boldsymbol{\theta}) = (\sigma(\theta_0 + \theta_1 x) - y)^2$$

Rewrite

$$\mathcal{L}(x, y, \boldsymbol{\theta}) = \underbrace{(\sigma(\theta_0 + \theta_1 x) - y)}_{\text{Nonlinear function of } \theta} \underbrace{(\sigma(\theta_0 + \theta_1 x) - y)}_{\text{Nonlinear function of } \theta}$$

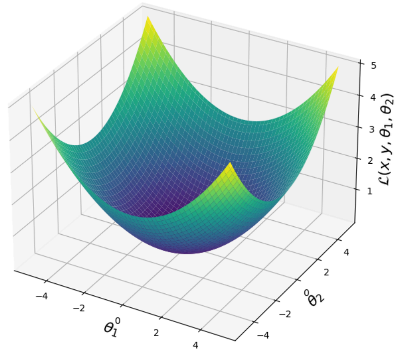
Deriving Linear Regression

Linear regression loss function was quadratic with one minima



Deriving Linear Regression

Linear regression loss function
was quadratic with one minima

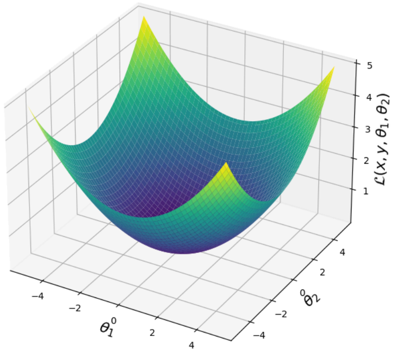


With a neural network, this is our loss function

$$\mathcal{L}(x, y, \boldsymbol{\theta}) = \underbrace{(\sigma(\theta_0 + \theta_1 x) - y)}_{\text{Nonlinear function of } \theta}\underbrace{(\sigma(\theta_0 + \theta_1 x) - y)}_{\text{Nonlinear function of } \theta}$$

Deriving Linear Regression

Linear regression loss function was quadratic with one minima



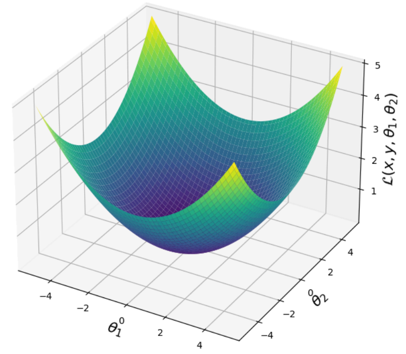
With a neural network, this is our loss function

$$\mathcal{L}(x, y, \boldsymbol{\theta}) = \underbrace{(\sigma(\theta_0 + \theta_1 x) - y)}_{\text{Nonlinear function of } \theta}\underbrace{(\sigma(\theta_0 + \theta_1 x) - y)}_{\text{Nonlinear function of } \theta}$$

Question: How many minima does this function have?

Deriving Linear Regression

Linear regression loss function was quadratic with one minima



With a neural network, this is our loss function

$$\mathcal{L}(x, y, \boldsymbol{\theta}) = \underbrace{(\sigma(\theta_0 + \theta_1 x) - y)}_{\text{Nonlinear function of } \theta} \underbrace{(\sigma(\theta_0 + \theta_1 x) - y)}_{\text{Nonlinear function of } \theta}$$

Question: How many minima does this function have?

Answer: We do not know

Deriving Linear Regression

The nonlinearity/activation function σ in the neural network means we cannot find an analytical solution

Deriving Linear Regression

The nonlinearity/activation function σ in the neural network means we cannot find an analytical solution

$$f(x, \boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x)$$

Deriving Linear Regression

The nonlinearity/activation function σ in the neural network means we cannot find an analytical solution

$$f(x, \boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x)$$

Question: Can we remove the activation function σ?

Deriving Linear Regression

The nonlinearity/activation function σ in the neural network means we cannot find an analytical solution

$$f(x, \boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x)$$

Question: Can we remove the activation function σ?

Answer: Yes, but the result is linear regression

Deriving Linear Regression

The nonlinearity/activation function σ in the neural network means we cannot find an analytical solution

$$f(x, \boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x)$$

Question: Can we remove the activation function σ?

Answer: Yes, but the result is linear regression

$$f(x, \boldsymbol{\theta}) = \theta_0 + \theta_1 x$$

Deriving Linear Regression

The nonlinearity/activation function σ in the neural network means we cannot find an analytical solution

$$f(x, \boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x)$$

Question: Can we remove the activation function σ?

Answer: Yes, but the result is linear regression

$$f(x, \boldsymbol{\theta}) = \theta_0 + \theta_1 x$$

Activation functions make the neural network powerful

Deriving Linear Regression

Linear regression: analytical solution for $\boldsymbol{\theta}$

Deriving Linear Regression

Linear regression: analytical solution for $\boldsymbol{\theta}$

Neural network: no analytical solution for $\boldsymbol{\theta}$

Deriving Linear Regression

Linear regression: analytical solution for $\boldsymbol{\theta}$

Neural network: no analytical solution for $\boldsymbol{\theta}$

So how to find $\boldsymbol{\theta}$ for a neural network?

Agenda

1. Review
2. Quiz
3. Optimization
4. Calculus review
5. **Deriving linear regression**
6. Gradient descent
7. Backpropagation
8. Coding

Agenda

1. Review
2. Quiz
3. Optimization
4. Calculus review
5. Deriving linear regression
6. **Gradient descent**
7. Backpropagation
8. Coding

Gradient Descent

To find $\boldsymbol{\theta}$ for a neural network, we use **gradient descent**

Gradient Descent

To find $\boldsymbol{\theta}$ for a neural network, we use **gradient descent**

Gradient descent optimizes **differentiable** functions

Gradient Descent

To find $\boldsymbol{\theta}$ for a neural network, we use **gradient descent**

Gradient descent optimizes **differentiable** functions

We must be able to take the derivative or gradient of the loss function to use gradient descent

Gradient Descent

To find $\boldsymbol{\theta}$ for a neural network, we use **gradient descent**

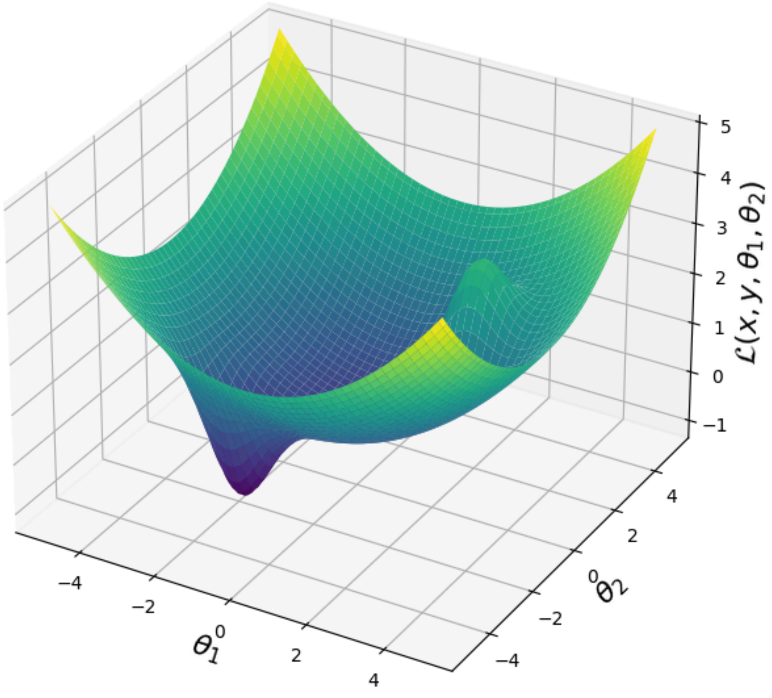
Gradient descent optimizes **differentiable** functions

We must be able to take the derivative or gradient of the loss function to use gradient descent

How does gradient descent work?

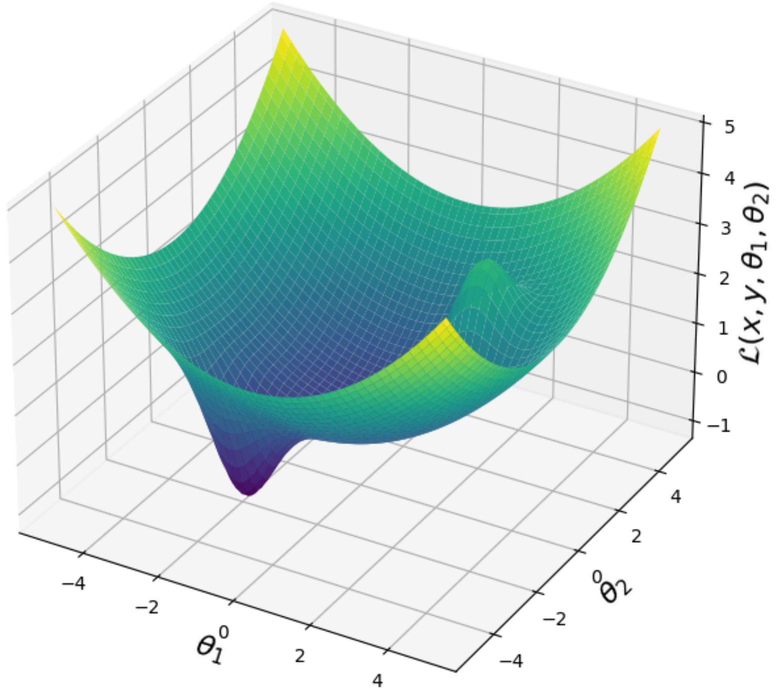
Gradient Descent

A differentiable loss function produces a manifold



Gradient Descent

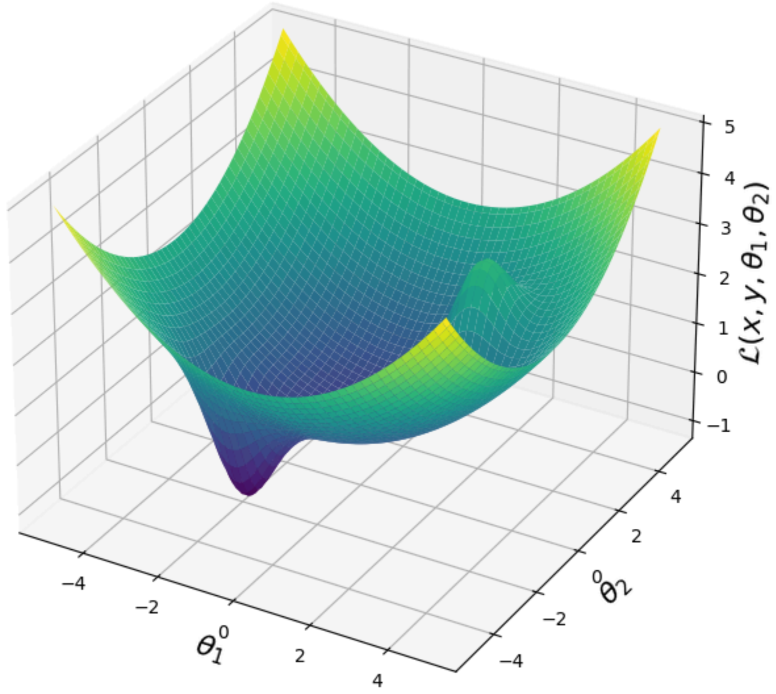
A differentiable loss function produces a manifold



Our goal is to find the lowest point on this manifold

Gradient Descent

A differentiable loss function produces a manifold



Our goal is to find the lowest point on this manifold

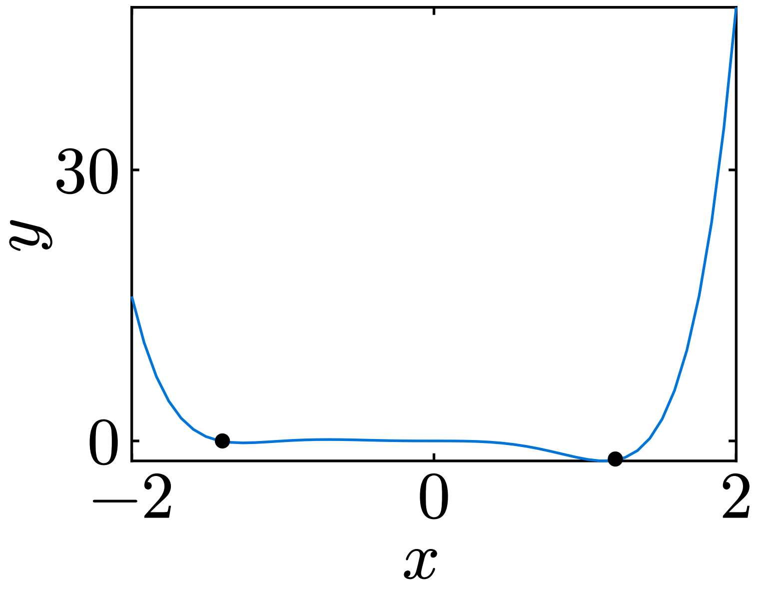
The lowest point solves $\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$

Gradient Descent

Note: Gradient descent provides a **local** optima, not necessarily a **global** optima

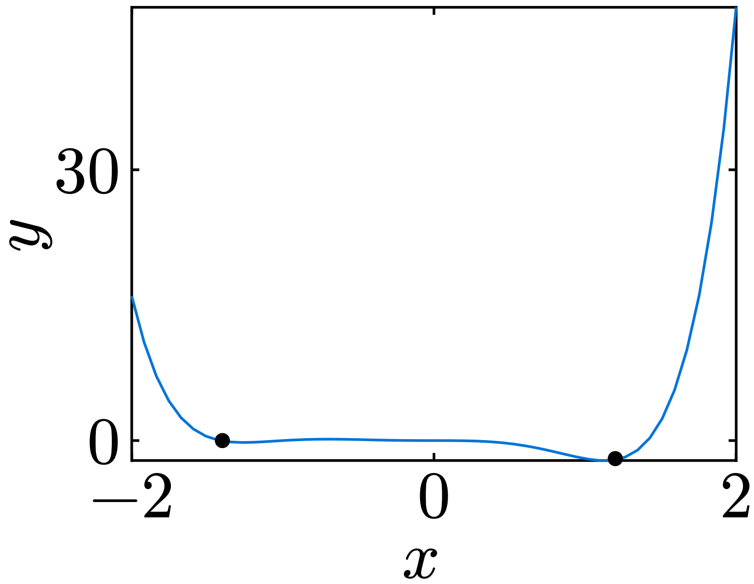
Gradient Descent

Note: Gradient descent provides a **local** optima, not necessarily a **global** optima



Gradient Descent

Note: Gradient descent provides a **local** optima, not necessarily a **global** optima



In practice, a local optima provides a good enough model

Relax

Gradient Descent

Let us define gradient descent without math

Gradient Descent

Let us define gradient descent without math

You are on the top of a mountain and there is lightning storm

Gradient Descent

Let us define gradient descent without math

You are on the top of a mountain and there is lightning storm

Gradient Descent

Let us define gradient descent without math

You are on the top of a mountain and there is lightning storm

For safety, you should walk down the mountain to escape the lightning

Gradient Descent

But you do not know the path down!

Gradient Descent

But you do not know the path down!

You see this, which way do you walk next?

Gradient Descent

This is gradient descent

Gradient Descent

In gradient descent, we look at the **slope** of the loss function

Gradient Descent

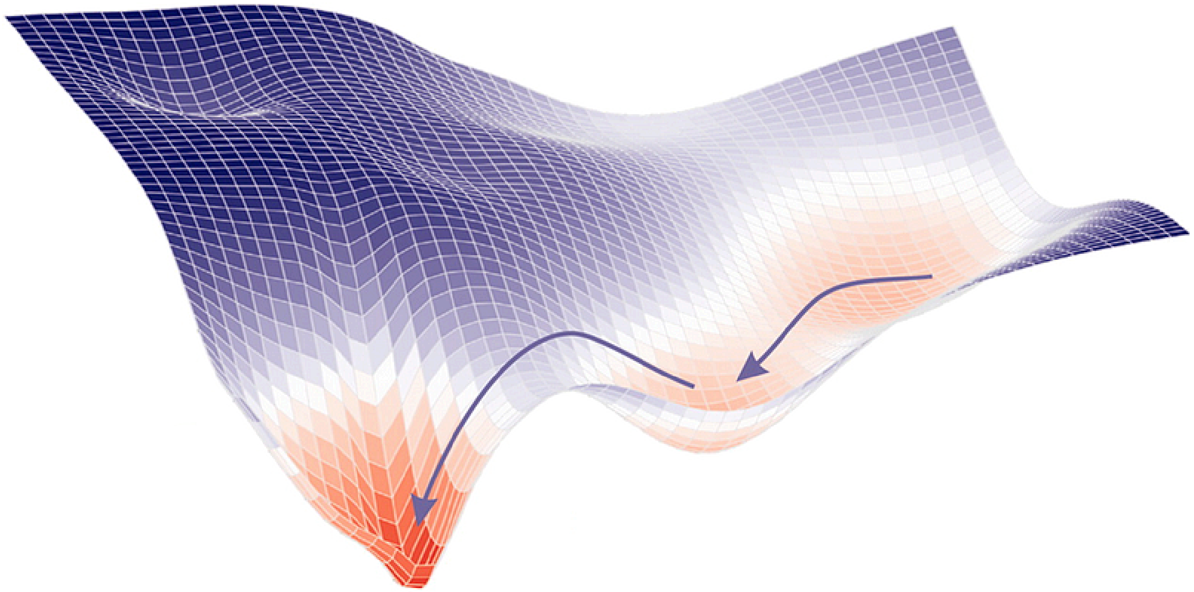
In gradient descent, we look at the **slope** of the loss function

And we walk in the steepest direction

Gradient Descent

In gradient descent, we look at the **slope** of the loss function

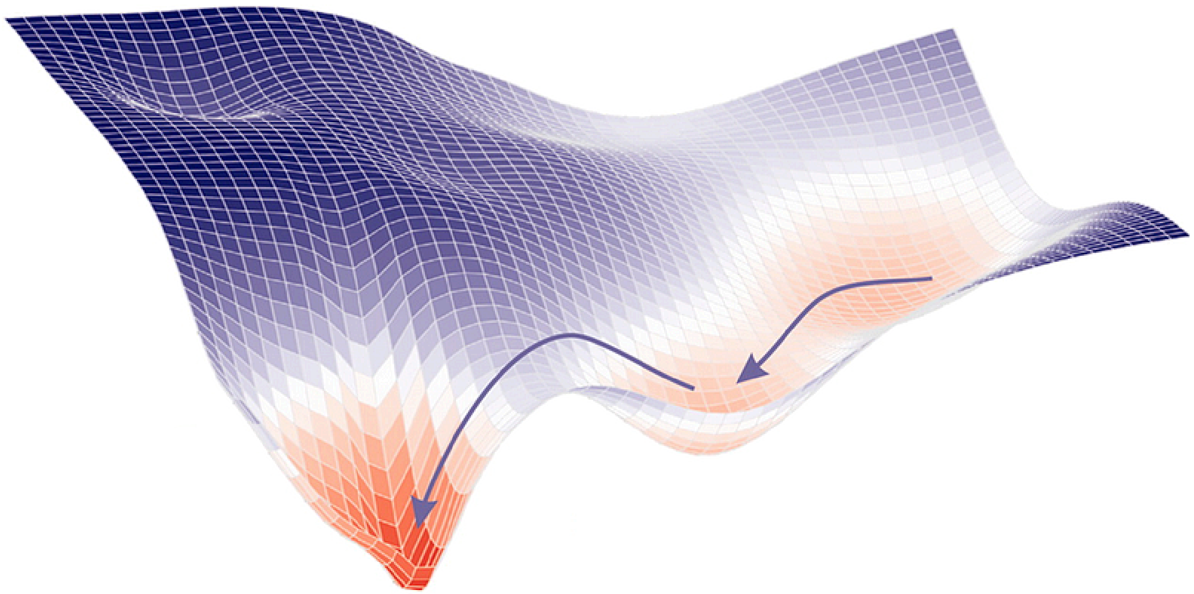
And we walk in the steepest direction



Gradient Descent

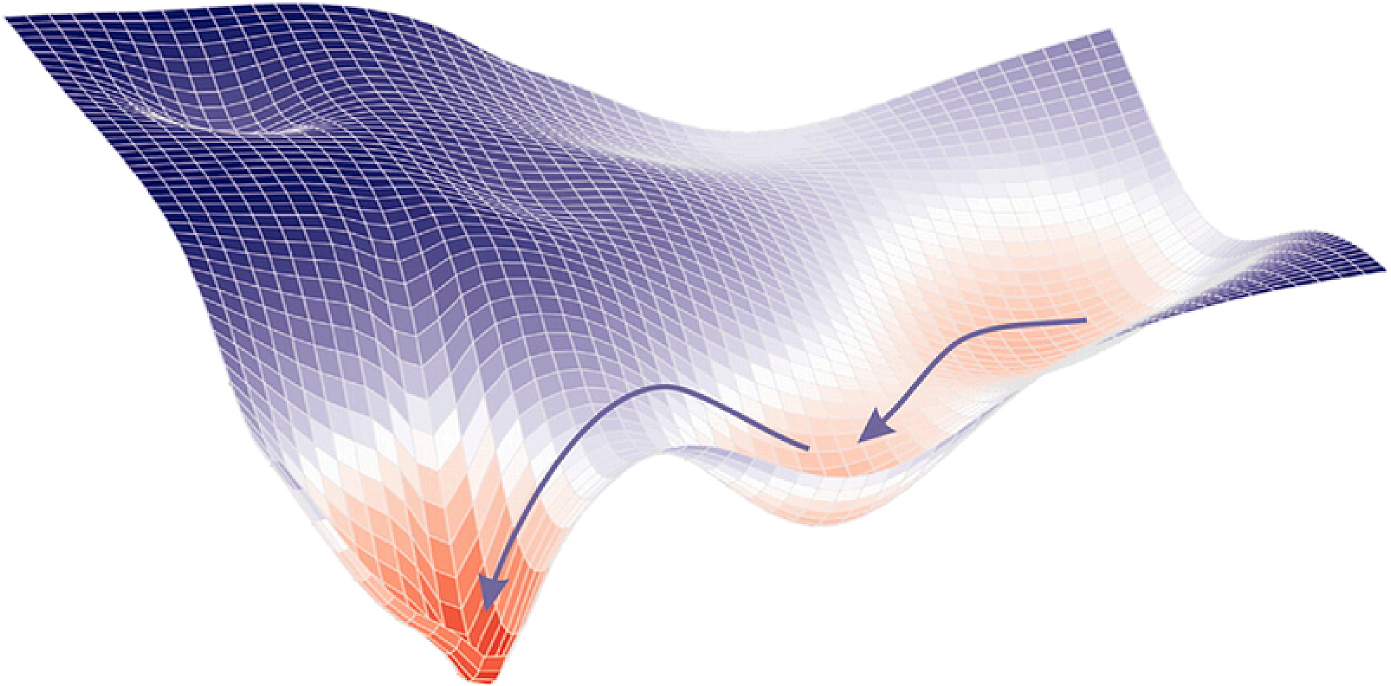
In gradient descent, we look at the **slope** of the loss function

And we walk in the steepest direction

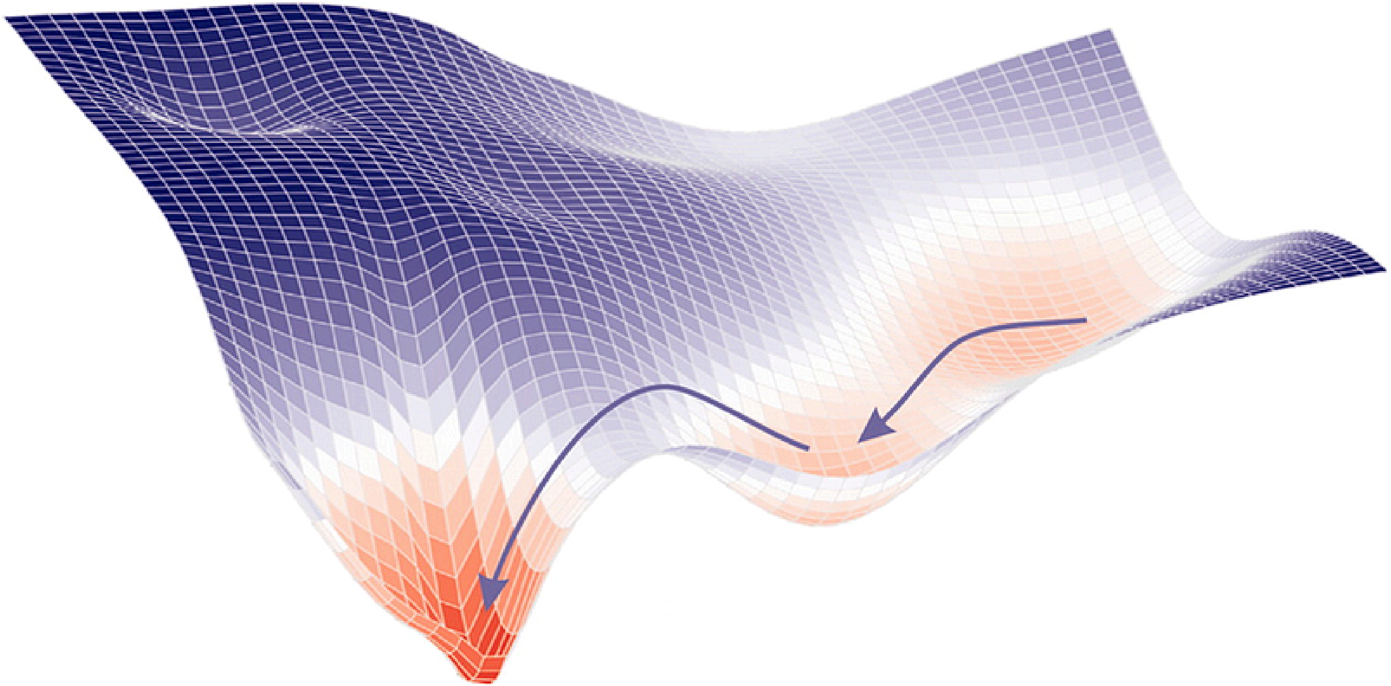


And then we repeat

Gradient Descent

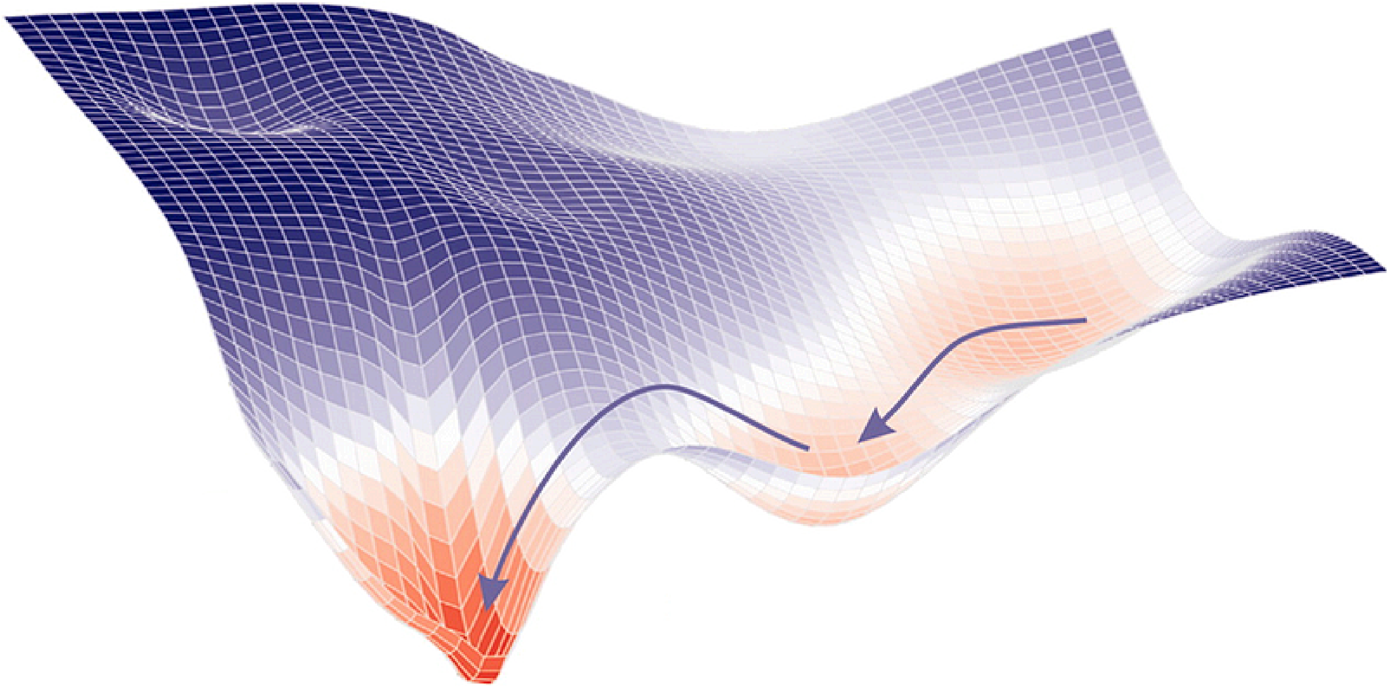


Gradient Descent



We find the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$

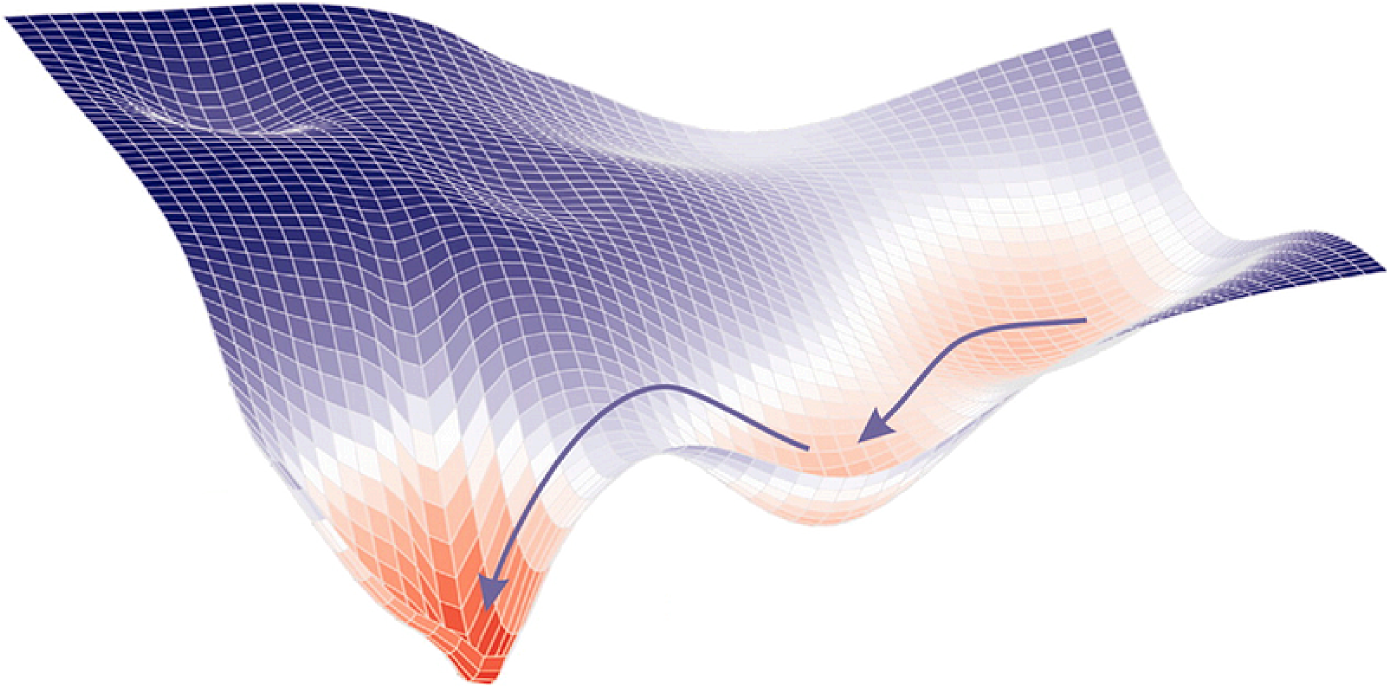
Gradient Descent



We find the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$

And update $\boldsymbol{\theta}$ in the steepest direction

Gradient Descent



Eventually, we arrive at the bottom

Gradient Descent

With gradient descent, the loss function must be differentiable

Gradient Descent

With gradient descent, the loss function must be differentiable

If we cannot compute the derivative/gradient, then we do not know which way to walk!

Gradient Descent

The gradient descent algorithm:

function Gradient Descent($\boldsymbol{X}$, $\boldsymbol{Y}$, $\mathcal{L}$, t, α)

  ▷ Randomly initialize parameters

  $\boldsymbol{\theta} \leftarrow \mathcal{N}(0, 1)$

  for $i \in 1...t$ **do**

    ▷ Compute the gradient of the loss

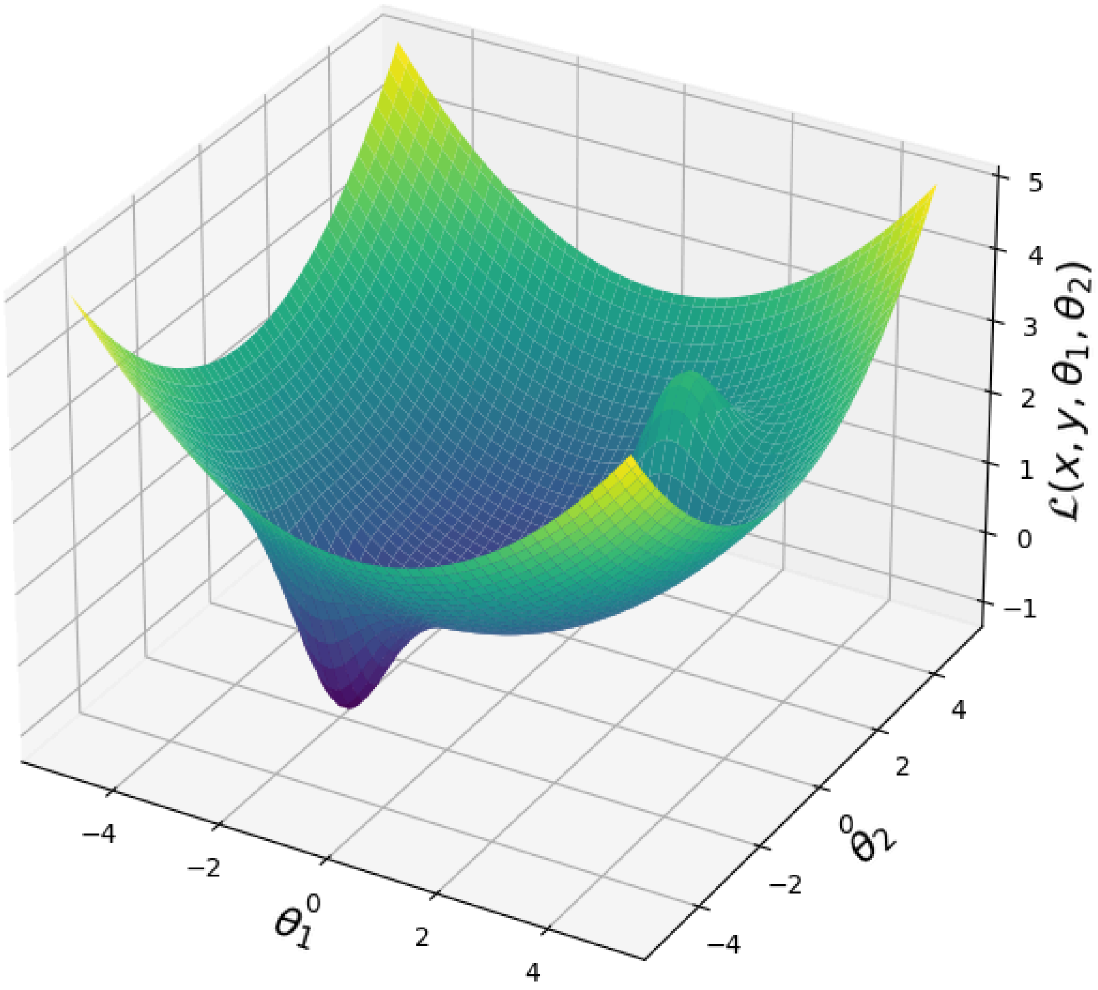
    $\boldsymbol{J} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$

    ▷ Update the parameters using the negative gradient

    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{J}$

  return $\boldsymbol{\theta}$

Gradient Descent



Gradient Descent

Two main steps in gradient descent:

Gradient Descent

Two main steps in gradient descent:

Step 1: Compute the gradient of the loss

Gradient Descent

Two main steps in gradient descent:

Step 1: Compute the gradient of the loss

Step 2: Update the parameters using the gradient

Gradient Descent

Two main steps in gradient descent:

Step 1: Compute the gradient of the loss

Step 2: Update the parameters using the gradient

Let us start with step 1

Agenda

1. Review
2. Quiz
3. Optimization
4. Calculus review
5. Deriving linear regression
6. **Gradient descent**
7. Backpropagation
8. Coding

Agenda

1. Review
2. Quiz
3. Optimization
4. Calculus review
5. Deriving linear regression
6. Gradient descent
7. **Backpropagation**
8. Coding

Backpropagation

Goal: Compute the gradient of the loss $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$

Backpropagation

Goal: Compute the gradient of the loss $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$

We call this process **backpropagation**

Backpropagation

Goal: Compute the gradient of the loss $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$

We call this process **backpropagation**

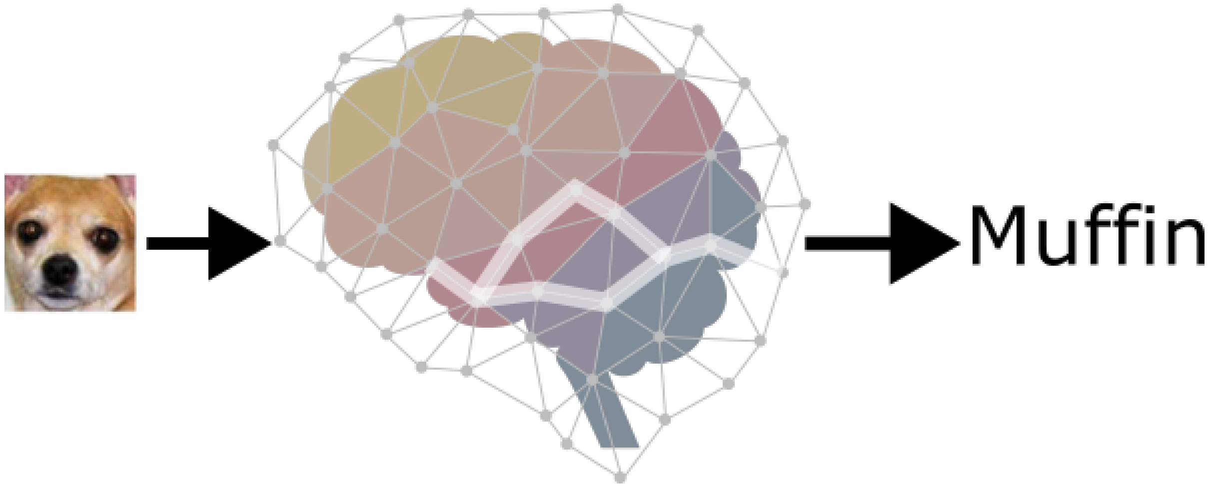
We propagate errors from the loss function **backward** through each layer of the neural network

Backpropagation

Goal: Compute the gradient of the loss $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$

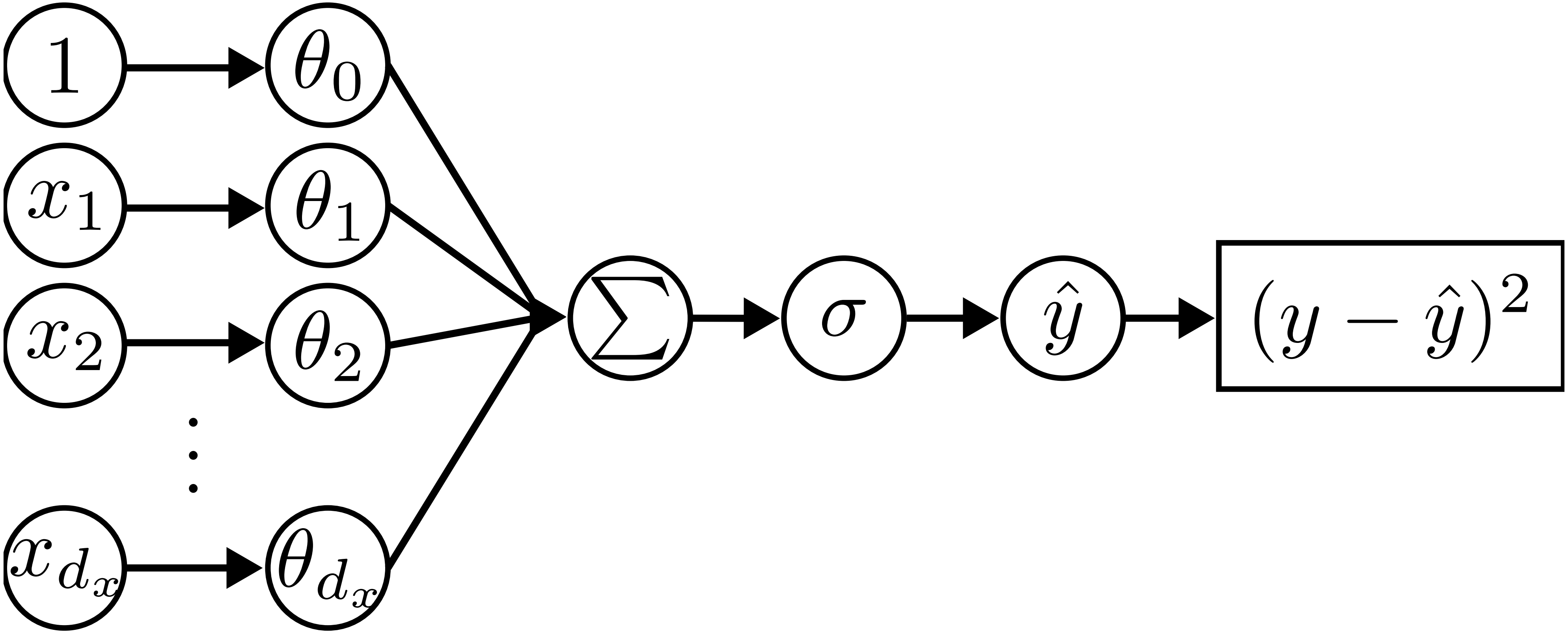
We call this process **backpropagation**

We propagate errors from the loss function **backward** through each layer of the neural network



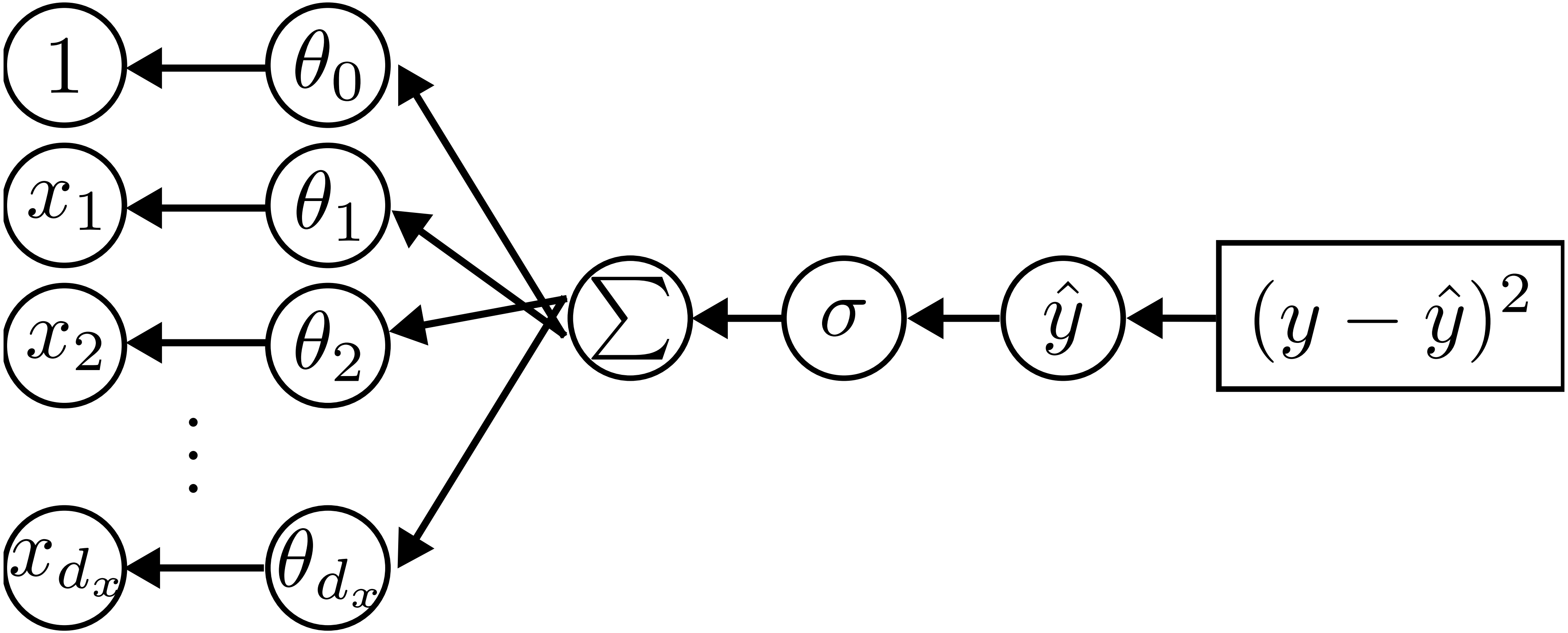
Backpropagation

Forward propagation



Backpropagation

Backward propagation



Backpropagation

Finding the gradient is necessary to use gradient descent!

Backpropagation

Finding the gradient is necessary to use gradient descent!

First, we will find the gradient of a neural network layer

Backpropagation

Finding the gradient is necessary to use gradient descent!

First, we will find the gradient of a neural network layer

Then, we will find the gradient of a deep neural network

Backpropagation

Finding the gradient is necessary to use gradient descent!

First, we will find the gradient of a neural network layer

Then, we will find the gradient of a deep neural network

Finally, we will find the gradient of the loss function

Backpropagation

Start with the equation of a neural network layer

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

Backpropagation

Start with the equation of a neural network layer

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

Take the gradient of both sides

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

Backpropagation

Start with the equation of a neural network layer

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

Take the gradient of both sides

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

$$\text{Chain: } \frac{d}{dx} f(g(x)) = f'(g(x)) \cdot g'(x)$$

Backpropagation

Start with the equation of a neural network layer

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

Take the gradient of both sides

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

$$\text{Chain: } \frac{d}{dx} f(g(x)) = f'(g(x)) \cdot g'(x)$$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} [\sigma](\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \cdot \nabla_{\boldsymbol{\theta}} (\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

Backpropagation

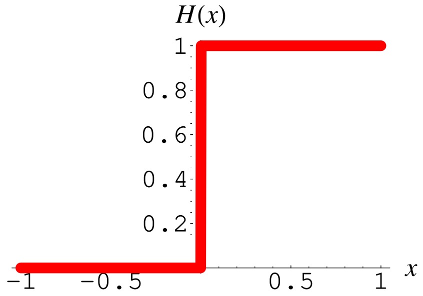
$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}[\sigma](\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \cdot \nabla_{\boldsymbol{\theta}}(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

What is $\nabla_{\boldsymbol{\theta}} \sigma$?

Backpropagation

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}[\sigma](\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \cdot \nabla_{\boldsymbol{\theta}}(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

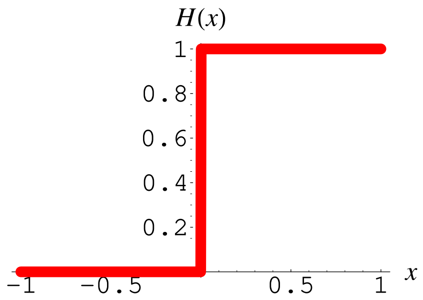
What is $\nabla_{\boldsymbol{\theta}} \sigma$?



Backpropagation

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}[\sigma](\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \cdot \nabla_{\boldsymbol{\theta}}(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

What is $\nabla_{\boldsymbol{\theta}} \sigma$?



Derivative is zero everywhere and infinity at $x = 0$, so the derivative for a layer is either infinity or zero

Backpropagation

We use a differentiable approximation of the heaviside step function

Backpropagation

We use a differentiable approximation of the heaviside step function

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

Backpropagation

We use a differentiable approximation of the heaviside step function

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

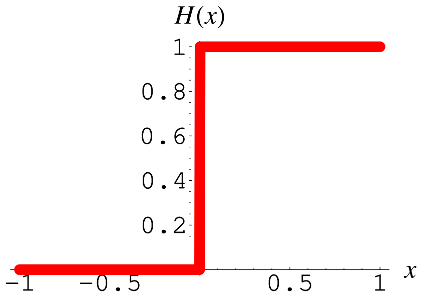
We call this approximation the **sigmoid function**

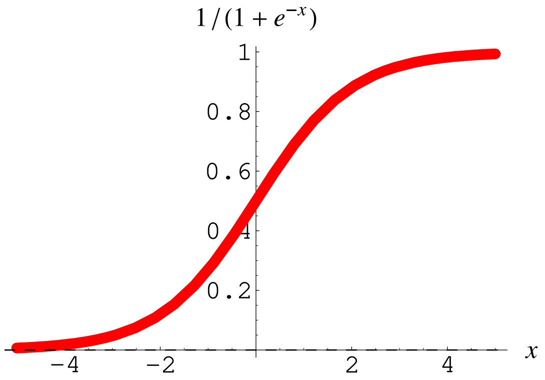
Backpropagation

We use a differentiable approximation of the heaviside step function

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

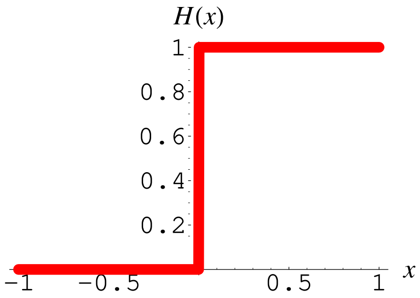
We call this approximation the **sigmoid function**

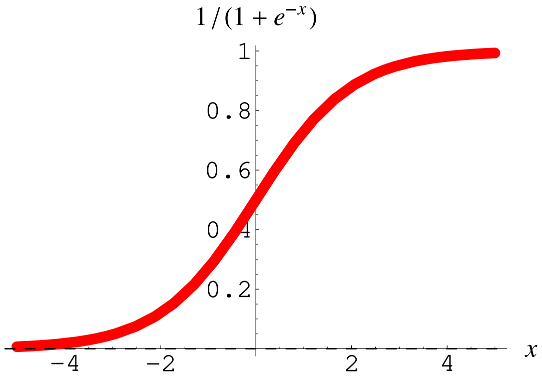




The sigmoid function has finite and nonzero derivative everywhere

Backpropagation



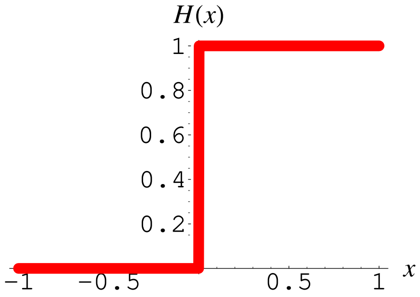


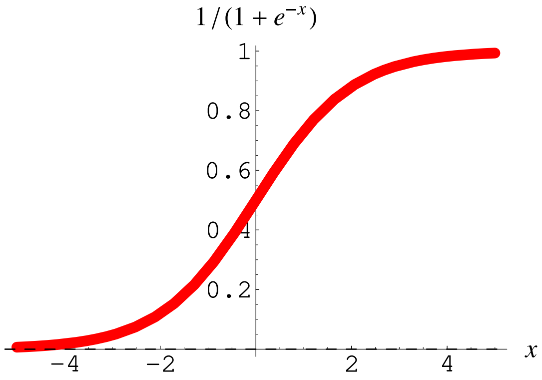
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The derivative of the sigmoid function is

$$\frac{d}{dz}\sigma(z) = \sigma(z) \cdot (1 - \sigma(z))$$

Backpropagation





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

The derivative of the sigmoid function is

$$\frac{d}{dz}\sigma(z) = \sigma(z) \cdot (1 - \sigma(z))$$

$$\nabla_{\boldsymbol{z}}\sigma(\boldsymbol{z}) = \sigma(\boldsymbol{z}) \odot (1 - \sigma(\boldsymbol{z}))$$

Backpropagation

Back to our layer

Backpropagation

Back to our layer

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}[\sigma](\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \cdot \nabla_{\boldsymbol{\theta}}(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

Backpropagation

Back to our layer

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}[\sigma](\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \cdot \nabla_{\boldsymbol{\theta}}(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

Plug in the gradient of our new activation function

$$\nabla_{\boldsymbol{z}} \sigma(\boldsymbol{z}) = \sigma(\boldsymbol{z}) \odot (1 - \sigma(\boldsymbol{z}))$$

Backpropagation

Back to our layer

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}[\sigma](\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \cdot \nabla_{\boldsymbol{\theta}}(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

Plug in the gradient of our new activation function

$$\nabla_{\boldsymbol{z}} \sigma(\boldsymbol{z}) = \sigma(\boldsymbol{z}) \odot (1 - \sigma(\boldsymbol{z}))$$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = (\sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \odot (1 - \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}))) \nabla_{\boldsymbol{\theta}}(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

Backpropagation

Back to our layer

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}[\sigma](\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \cdot \nabla_{\boldsymbol{\theta}}(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

Plug in the gradient of our new activation function

$$\nabla_{\boldsymbol{z}} \sigma(\boldsymbol{z}) = \sigma(\boldsymbol{z}) \odot (1 - \sigma(\boldsymbol{z}))$$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = (\sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \odot (1 - \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}))) \nabla_{\boldsymbol{\theta}}(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

Evalute the final term

Backpropagation

Back to our layer

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}[\sigma](\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \cdot \nabla_{\boldsymbol{\theta}}(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

Plug in the gradient of our new activation function

$$\nabla_{\boldsymbol{z}} \sigma(\boldsymbol{z}) = \sigma(\boldsymbol{z}) \odot (1 - \sigma(\boldsymbol{z}))$$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = (\sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \odot (1 - \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}))) \nabla_{\boldsymbol{\theta}}(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

Evalute the final term

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = (\sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \odot (1 - \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}))) \overline{\boldsymbol{x}}^\top$$

Backpropagation

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \left(\sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \odot (1 - \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}))\right)\overline{\boldsymbol{x}}^\top$$

Backpropagation

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = (\sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \odot (1 - \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})))\overline{\boldsymbol{x}}^\top$$

This is the gradient for the layer of a neural network!

Backpropagation

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \left(\sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}) \odot (1 - \sigma(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}}))\right)\overline{\boldsymbol{x}}^\top$$

This is the gradient for the layer of a neural network!

We will use this to compute the gradient of a deep neural network

Backpropagation

Recall the deep neural network has many layers

$$f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}}) \quad f_2(\boldsymbol{x}, \boldsymbol{\psi}) = \sigma(\boldsymbol{\psi}^\top \overline{\boldsymbol{x}}) \quad \dots \quad f_\ell(\boldsymbol{x}, \boldsymbol{\xi}) = \sigma(\boldsymbol{\xi}^\top \overline{\boldsymbol{x}})$$

Backpropagation

Recall the deep neural network has many layers

$$f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}}) \quad f_2(\boldsymbol{x}, \boldsymbol{\psi}) = \sigma(\boldsymbol{\psi}^\top \overline{\boldsymbol{x}}) \quad \ldots \quad f_\ell(\boldsymbol{x}, \boldsymbol{\xi}) = \sigma(\boldsymbol{\xi}^\top \overline{\boldsymbol{x}})$$

And that we call them in series

$$\begin{aligned} \boldsymbol{z}_1 &= f_1(\boldsymbol{x}, \boldsymbol{\varphi}) \\ \boldsymbol{z}_2 &= f_2(\boldsymbol{z}_1, \boldsymbol{\psi}) \\ &\vdots \\ \boldsymbol{z}_\ell &= f_\ell(\boldsymbol{z}_{\ell-1}, \boldsymbol{\xi}) \end{aligned}$$

Backpropagation

Recall the deep neural network has many layers

$$f_1(\boldsymbol{x}, \boldsymbol{\varphi}) = \sigma(\boldsymbol{\varphi}^\top \overline{\boldsymbol{x}}) \quad f_2(\boldsymbol{x}, \boldsymbol{\psi}) = \sigma(\boldsymbol{\psi}^\top \overline{\boldsymbol{x}}) \quad \dots \quad f_\ell(\boldsymbol{x}, \boldsymbol{\xi}) = \sigma(\boldsymbol{\xi}^\top \overline{\boldsymbol{x}})$$

And that we call them in series

$$\begin{aligned} \boldsymbol{z}_1 &= f_1(\boldsymbol{x}, \boldsymbol{\varphi}) \\ \boldsymbol{z}_2 &= f_2(\boldsymbol{z}_1, \boldsymbol{\psi}) \\ &\vdots \\ \boldsymbol{z}_\ell &= f_\ell(\boldsymbol{z}_{\ell-1}, \boldsymbol{\xi}) \end{aligned}$$

Backpropagation

Take the gradient of both sides

$$\nabla_{\varphi, \psi, \dots, \xi} z_1 = \nabla_{\varphi, \psi, \xi} f_1(x, \varphi)$$

$$\nabla_{\varphi, \psi, \dots, \xi} z_2 = \nabla_{\varphi, \psi, \xi} f_2(z_1, \psi)$$

$$\vdots$$

$$\nabla_{\varphi, \psi, \dots, \xi} z_\ell = \nabla_{\varphi, \psi, \xi} f_\ell(z_{\ell-1}, \xi)$$

Backpropagation

Take the gradient of both sides

$$\begin{aligned}
\nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \ldots, \boldsymbol{\xi}} \boldsymbol{z}_1 &= \nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \boldsymbol{\xi}} f_1(\boldsymbol{x}, \boldsymbol{\varphi}) \\
\nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \ldots, \boldsymbol{\xi}} \boldsymbol{z}_2 &= \nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \boldsymbol{\xi}} f_2(\boldsymbol{z}_1, \boldsymbol{\psi}) \\
&\vdots \\
\nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \ldots, \boldsymbol{\xi}} \boldsymbol{z}_\ell &= \nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \boldsymbol{\xi}} f_\ell(\boldsymbol{z}_{\ell-1}, \boldsymbol{\xi})
\end{aligned}$$

Each layer only uses one set of parameters

$$\begin{aligned}
\nabla_{\boldsymbol{\varphi}} \boldsymbol{z}_1 &= \nabla_{\boldsymbol{\varphi}} f_1(\boldsymbol{x}, \boldsymbol{\varphi}) \\
\nabla_{\boldsymbol{\psi}} \boldsymbol{z}_2 &= \nabla_{\boldsymbol{\psi}} f_2(\boldsymbol{z}_1, \boldsymbol{\psi}) \\
&\vdots \\
\nabla_{\boldsymbol{\xi}} \boldsymbol{z}_\ell &= \nabla_{\boldsymbol{\xi}} f_\ell(\boldsymbol{z}_{\ell-1}, \boldsymbol{\xi})
\end{aligned}$$

Backpropagation

Take the gradient of both sides

$$\nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \ldots, \boldsymbol{\xi}} \boldsymbol{z}_1 = \nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \boldsymbol{\xi}} f_1(\boldsymbol{x}, \boldsymbol{\varphi})$$

$$\nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \ldots, \boldsymbol{\xi}} \boldsymbol{z}_2 = \nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \boldsymbol{\xi}} f_2(\boldsymbol{z}_1, \boldsymbol{\psi})$$

$$\vdots$$

$$\nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \ldots, \boldsymbol{\xi}} \boldsymbol{z}_\ell = \nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \boldsymbol{\xi}} f_\ell(\boldsymbol{z}_{\ell-1}, \boldsymbol{\xi})$$

Each layer only uses one set of parameters

$$\nabla_{\boldsymbol{\varphi}} \boldsymbol{z}_1 = \nabla_{\boldsymbol{\varphi}} f_1(\boldsymbol{x}, \boldsymbol{\varphi})$$

$$\nabla_{\boldsymbol{\psi}} \boldsymbol{z}_2 = \nabla_{\boldsymbol{\psi}} f_2(\boldsymbol{z}_1, \boldsymbol{\psi})$$

$$\vdots$$

$$\nabla_{\boldsymbol{\xi}} \boldsymbol{z}_\ell = \nabla_{\boldsymbol{\xi}} f_\ell(\boldsymbol{z}_{\ell-1}, \boldsymbol{\xi})$$

Backpropagation

The gradient of a deep neural network is

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \ldots, \boldsymbol{\xi}} f\left(\boldsymbol{x}, [\boldsymbol{\varphi} \ \ \boldsymbol{\psi} \ \ \ldots \ \ \boldsymbol{\xi}]^\top\right) = \begin{bmatrix} \nabla_{\boldsymbol{\varphi}} f_1(\boldsymbol{x}, \boldsymbol{\varphi}) \\ \nabla_{\boldsymbol{\psi}} f_2(\boldsymbol{z}_1, \boldsymbol{\psi}) \\ \vdots \\ \nabla_{\boldsymbol{\xi}} f_\ell(\boldsymbol{z}_{\ell-1}, \boldsymbol{\xi}) \end{bmatrix}$$

Backpropagation

The gradient of a deep neural network is

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \ldots, \boldsymbol{\xi}} f\left(\boldsymbol{x}, [\boldsymbol{\varphi} \ \ \boldsymbol{\psi} \ \ \ldots \ \ \boldsymbol{\xi}]^\top\right) = \begin{bmatrix} \nabla_{\boldsymbol{\varphi}} f_1(\boldsymbol{x}, \boldsymbol{\varphi}) \\ \nabla_{\boldsymbol{\psi}} f_2(\boldsymbol{z}_1, \boldsymbol{\psi}) \\ \vdots \\ \nabla_{\boldsymbol{\xi}} f_\ell(\boldsymbol{z}_{\ell-1}, \boldsymbol{\xi}) \end{bmatrix}$$

Where each layer gradient is

$$\nabla_{\boldsymbol{\xi}} f_\ell(\boldsymbol{z}_{\ell-1}, \boldsymbol{\xi}) = (\sigma(\boldsymbol{\xi}^\top \overline{\boldsymbol{z}}_{\ell-1}) \odot (1 - \sigma(\boldsymbol{\xi}^\top \overline{\boldsymbol{z}}_{\ell-1}))) \overline{\boldsymbol{z}}_{\ell-1}^\top$$

Backpropagation

We computed the gradient of a neural network layer

Backpropagation

We computed the gradient of a neural network layer

We computed the gradient of the neural network

Backpropagation

We computed the gradient of a neural network layer

We computed the gradient of the neural network

Now, we must compute gradient of the loss function

Backpropagation

We computed the gradient of a neural network layer

We computed the gradient of the neural network

Now, we must compute gradient of the loss function

Backpropagation

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta} \right) - \boldsymbol{y}_{[i]} \right)^2$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta} \right) - \boldsymbol{y}_{[i]} \right)^2$$

Backpropagation

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta} \right) - \boldsymbol{y}_{[i]} \right)^2$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta} \right) - \boldsymbol{y}_{[i]} \right)^2$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta} \right) - \boldsymbol{y}_{[i]} \right)^2$$

Backpropagation

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta} \right) - \boldsymbol{y}_{[i]} \right)^2$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta} \right) - \boldsymbol{y}_{[i]} \right)^2$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta} \right) - \boldsymbol{y}_{[i]} \right)^2$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} 2 \left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta} \right) - \boldsymbol{y}_{[i]} \right) \nabla_{\boldsymbol{\theta}} f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta} \right)$$

Backpropagation

To summarize:

Backpropagation

To summarize:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} 2\left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right) - \boldsymbol{y}_{[i]} \right) \nabla_{\boldsymbol{\theta}} f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)$$

Backpropagation

To summarize:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} 2\left(f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right) - \boldsymbol{y}_{[i]}\right) \nabla_{\boldsymbol{\theta}} f\left(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\right)$$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \ldots, \boldsymbol{\xi}} f\left(\boldsymbol{x}, [\boldsymbol{\varphi} \ \ \boldsymbol{\psi} \ \ \ldots \ \ \boldsymbol{\xi}]^\top\right) = \begin{bmatrix} \nabla_{\boldsymbol{\varphi}} f_1(\boldsymbol{x}, \boldsymbol{\varphi}) \\ \nabla_{\boldsymbol{\psi}} f_2(\boldsymbol{z}_1, \boldsymbol{\psi}) \\ \vdots \\ \nabla_{\boldsymbol{\xi}} f_\ell(\boldsymbol{z}_{\ell-1}, \boldsymbol{\xi}) \end{bmatrix}$$

Backpropagation

To summarize:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} 2\Big(f\Big(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\Big) - \boldsymbol{y}_{[i]}\Big) \nabla_{\boldsymbol{\theta}} f\Big(\boldsymbol{x}_{[i]}, \boldsymbol{\theta}\Big)$$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\varphi}, \boldsymbol{\psi}, \ldots, \boldsymbol{\xi}} f\Big(\boldsymbol{x}, \begin{bmatrix} \boldsymbol{\varphi} & \boldsymbol{\psi} & \ldots & \boldsymbol{\xi} \end{bmatrix}^\top\Big) = \begin{bmatrix} \nabla_{\boldsymbol{\varphi}} f_1(\boldsymbol{x}, \boldsymbol{\varphi}) \\ \nabla_{\boldsymbol{\psi}} f_2(\boldsymbol{z}_1, \boldsymbol{\psi}) \\ \vdots \\ \nabla_{\boldsymbol{\xi}} f_\ell(\boldsymbol{z}_{\ell-1}, \boldsymbol{\xi}) \end{bmatrix}$$

$$\nabla_{\boldsymbol{\xi}} f_\ell(\boldsymbol{z}_{\ell-1}, \boldsymbol{\xi}) = \left(\sigma(\boldsymbol{\xi}^\top \overline{\boldsymbol{z}}_{\ell-1}) \odot (1 - \sigma(\boldsymbol{\xi}^\top \overline{\boldsymbol{z}}_{\ell-1}))\right) \overline{\boldsymbol{z}}_{\ell-1}^\top$$

Backpropagation

Question: Why did we spend all this time deriving gradients?

Backpropagation

Question: Why did we spend all this time deriving gradients?

Answer: The gradient is necessary for gradient descent

Backpropagation

Question: Why did we spend all this time deriving gradients?

Answer: The gradient is necessary for gradient descent

for $i \in 1 ... t$ **do**

$\qquad \triangleright$ Compute the gradient of the loss

$\qquad \boldsymbol{J} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$

$\qquad \triangleright$ Update the parameters using the negative gradient

$\qquad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{J}$

Backpropagation

Question: Why did we spend all this time deriving gradients?

Answer: The gradient is necessary for gradient descent

for $i \in 1 \ldots t$ **do**
  $\triangleright$ Compute the gradient of the loss
  $\boldsymbol{J} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$
  $\triangleright$ Update the parameters using the negative gradient
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{J}$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}_t)$$

Agenda

1. Review
2. Quiz
3. Optimization
4. Calculus review
5. Deriving linear regression
6. Gradient descent
7. **Backpropagation**
8. Coding

Agenda

1. Review
2. Quiz
3. Optimization
4. Calculus review
5. Deriving linear regression
6. Gradient descent
7. Backpropagation
8. **Coding**

How do gradients work in `jax` or `torch`?

How do gradients work in `jax` or `torch`?

The libraries compute the gradients using **autograd**

How do gradients work in `jax` or `torch`?

The libraries compute the gradients using **autograd**

Autograd differentiates nested functions using the chain rule

How do gradients work in `jax` or `torch`?

The libraries compute the gradients using **autograd**

Autograd differentiates nested functions using the chain rule

$$\nabla_{\boldsymbol{\theta}} f(g(h(x, \boldsymbol{\theta}))) = f'(g(h(x, \boldsymbol{\theta}))) \cdot g'(h(x, \boldsymbol{\theta})) \cdot h'(x, \boldsymbol{\theta})$$

How do gradients work in `jax` or `torch`?

The libraries compute the gradients using **autograd**

Autograd differentiates nested functions using the chain rule

$$\nabla_{\boldsymbol{\theta}} f(g(h(x, \boldsymbol{\theta}))) = f'(g(h(x, \boldsymbol{\theta}))) \cdot g'(h(x, \boldsymbol{\theta})) \cdot h'(x, \boldsymbol{\theta})$$

Engineers derived gradients for hundreds of functions $f, g, h, ...$

How do gradients work in `jax` or `torch`?

The libraries compute the gradients using **autograd**

Autograd differentiates nested functions using the chain rule

$$\nabla_{\boldsymbol{\theta}} f(g(h(x, \boldsymbol{\theta}))) = f'(g(h(x, \boldsymbol{\theta}))) \cdot g'(h(x, \boldsymbol{\theta})) \cdot h'(x, \boldsymbol{\theta})$$

Engineers derived gradients for hundreds of functions $f, g, h, ...$

Researchers derive their own analytical gradients like we did today

How do gradients work in `jax` or `torch`?

The libraries compute the gradients using **autograd**

Autograd differentiates nested functions using the chain rule

$$\nabla_{\boldsymbol{\theta}} f(g(h(x, \boldsymbol{\theta}))) = f'(g(h(x, \boldsymbol{\theta}))) \cdot g'(h(x, \boldsymbol{\theta})) \cdot h'(x, \boldsymbol{\theta})$$

Engineers derived gradients for hundreds of functions $f, g, h, ...$

Researchers derive their own analytical gradients like we did today

Now, let us look at `jax` and `torch` optimization code

```python
import jax

def L(theta, X, Y):
  ...

# Create a new function that is the gradient of L
# Then compute gradient of L for given inputs
J = jax.grad(L)(X, Y, theta)
# Update parameters
alpha = 0.0001
theta = theta - alpha * J
```

```python
import torch
optimizer = torch.optim.SGD(lr=0.0001)

def L(model, X, Y):
  ...
# Pytorch will record a graph of all operations
# Everytime you do theta @ x, it stores inputs and outputs
loss = L(X, Y, model) # compute gradient
# Traverse the graph backward and compute the gradient
loss.backward() # Sets .grad attribute on each parameter
optimizer.step() # Update the parameters using .grad
optimizer.zero_grad() # Set .grad to zero, DO NOT FORGET!!
```

Time for some interactive coding

https://colab.research.google.com/drive/1W8WVZ8n_9yJCcOqkPVURp_wJUx3EQc5w