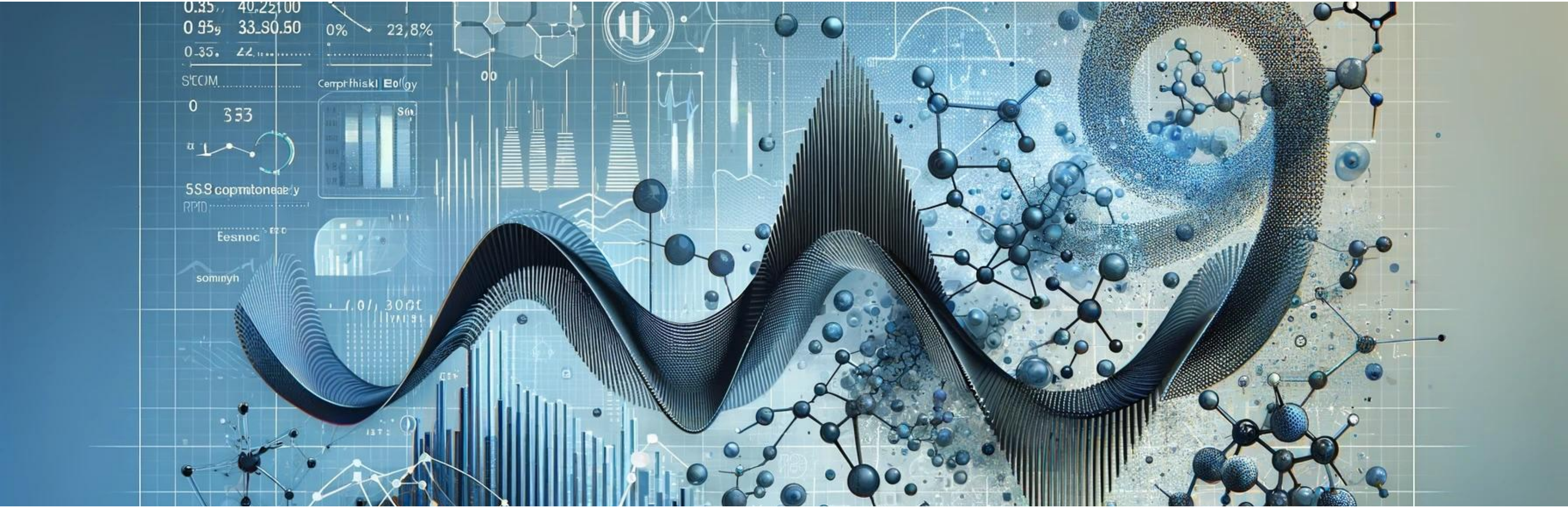
CSE7850/CX4803 Machine Learning in Computational Biology



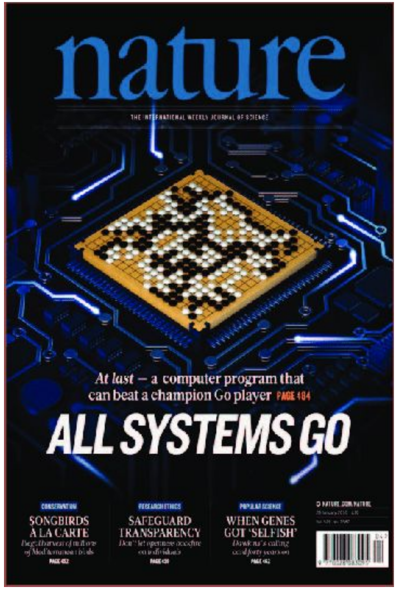
Lecture 5: Regression

Yunan Luo

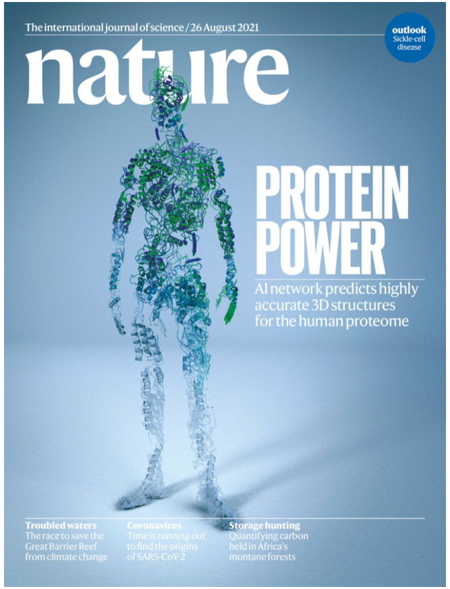
Acknowledgements: Some of the slides were adapted from lectures of Jian Peng and Volodymyr Kuleshov

What is machine learning?

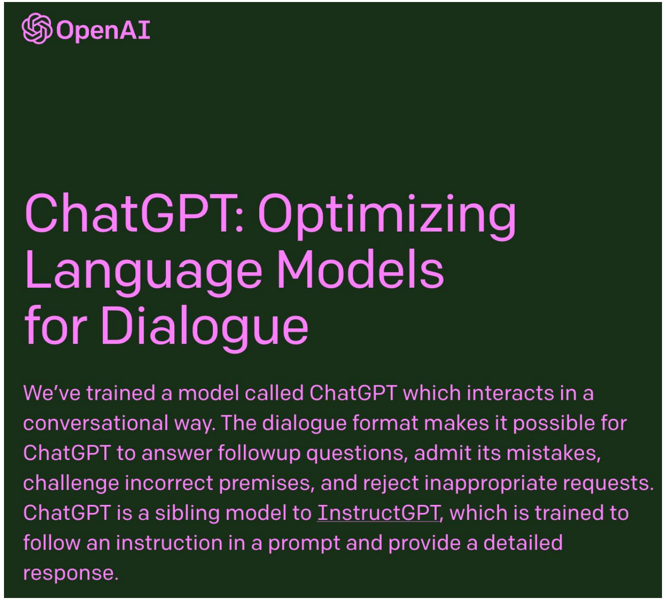
We hear a lot about machine learning (or ML for short) in the news.



Alpha Go (2016)



Alpha Fold (2018-2021)

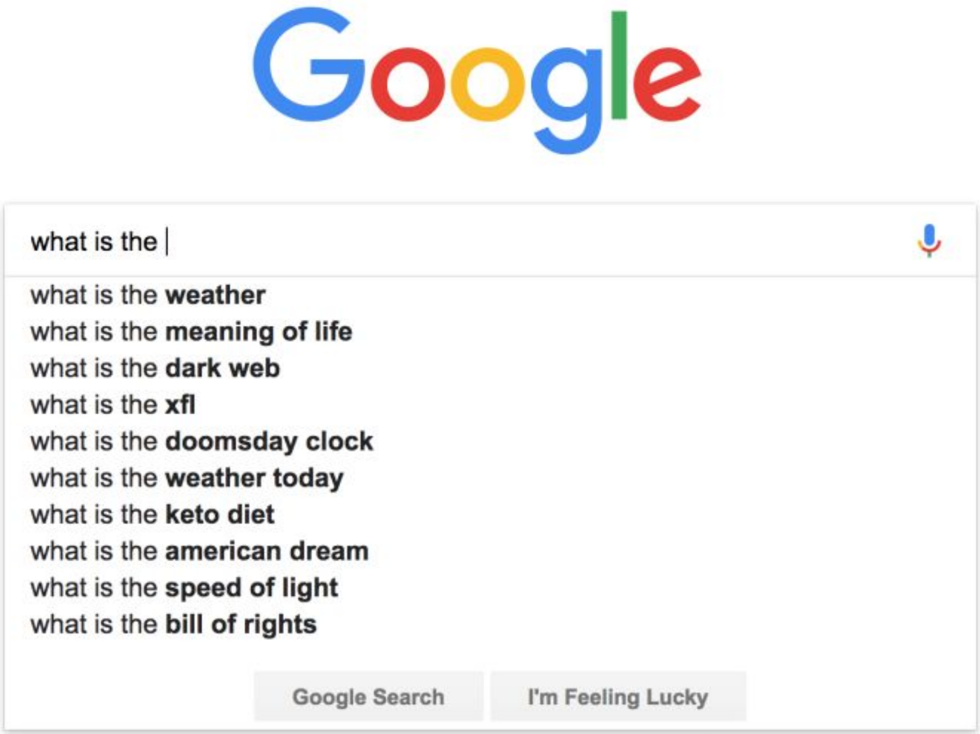


ChatGPT (2022)

But what is it, really?

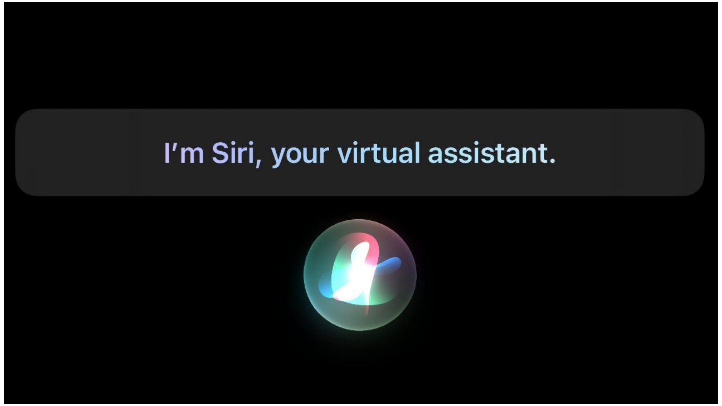
You use ML everyday!

Search engines



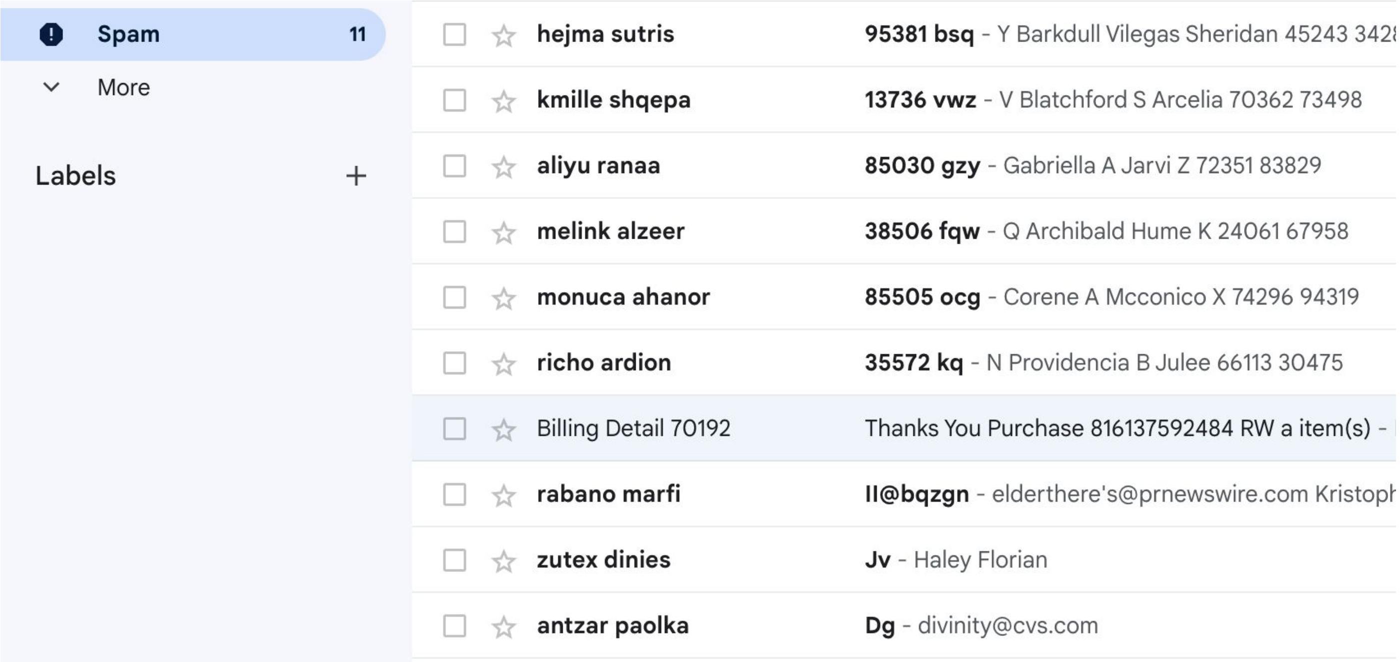
You use ML everyday!

Personal assistants



You use ML everyday!

Email spam filter



A definition of machine learning

(1901 – 1990)

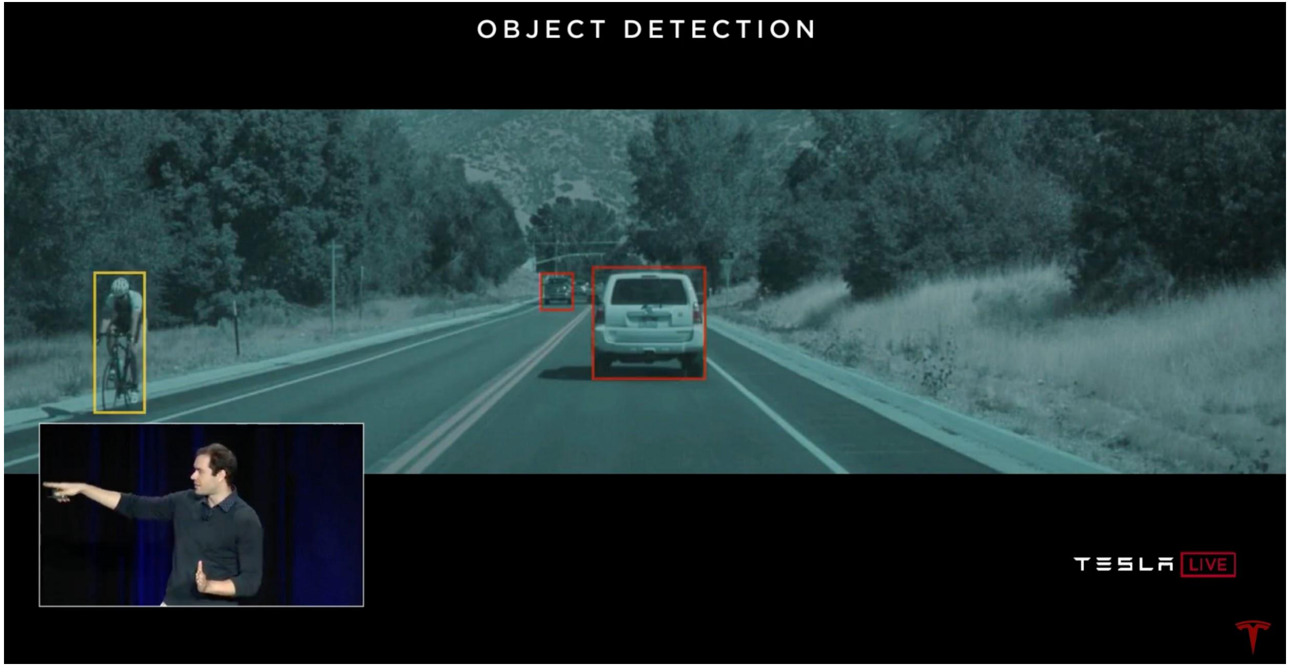
- In 1959, Arthur Samuel defined machine learning as follows.

> "Machine learning is a field of study that gives computers the ability to learn without being explicitly programmed."

- What does "learn" and "explicitly programmed" mean here?
  Let's look at an example.

An example: Self-driving Cars

- A self-driving car system uses dozens of components that include detection of cars, pedestrians, and other objects.



Self Driving Cars: A Rule-Based Algorithm

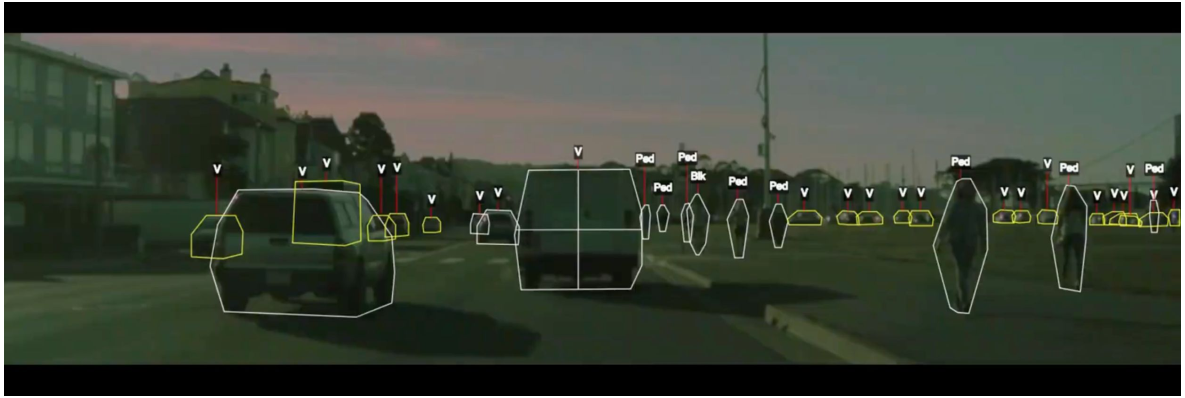
- One way to build a detection system is to write down rules.

```
# pseudocode example for a rule-based classification system
object = camera.get_object()
if object.has_wheels(): # does the object have wheels?
    if len(object.wheels) == 4: return "Car" # four wheels => car
    elif len(object.wheels) == 2:,
        if object.seen_from_back():
            return "Car" # viewed from back, car has 2 wheels
        else:
            return "Bicycle" # normally, 2 wheels => bicycle
return "Unknown" # no wheels? we don't know what it is
```

- In practice, it's almost impossible for a human to specify all the edge cases.

Self Driving Cars: An ML Approach

- The machine learning approach is to teach a computer how to do detection by showing it many examples of different objects.



- No manual programming is needed: the computer learns what defines a pedestrian or a car on its own!

Revisiting Our Definition of ML

> "Machine learning is a field of study that gives computers the ability to learn without being explicitly programmed."

- This principle can be applied to countless domains: medical diagnosis, factory automation, machine translation, and many more!

Why Machine Learning?

Why is this approach to building software interesting?

- It lets us build practical systems for real-world applications for which other engineering approaches don't work.
- Learning is widely regarded as a key approach towards building general-purpose artificial intelligence systems.
- The science and engineering of machine learning offers insights into human intelligence.

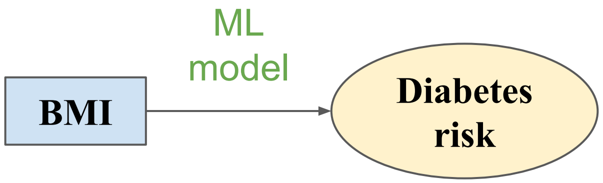
An ML example: linear regression

Predicting diabetes risk

Let’s start with a simple example of machine learning problem: **predicting diabetes risk**.

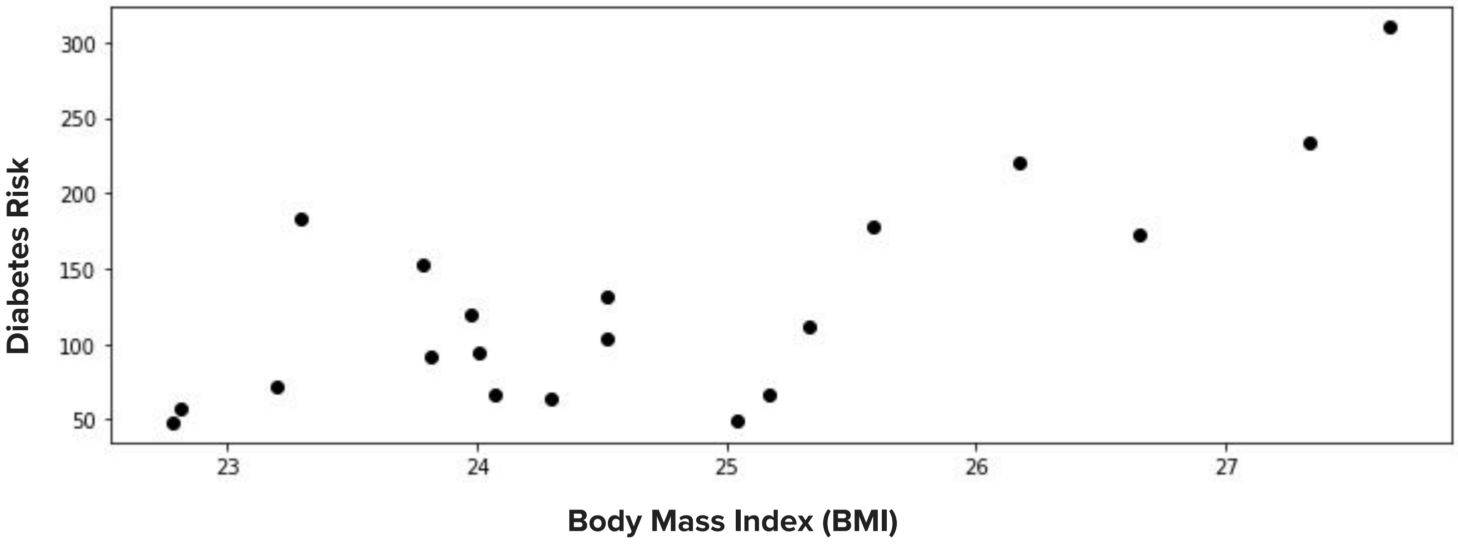
We start with a dataset of diabetes patients.

- For each patient we have an access to their **BMI** (body mass index) and an estimate of **diabetes risk** (from 0-400).
- We are interested if we can ***predict*** an individual's diabetes risk based on the BMI



Individual	BMI	Risk
1	27	233
2	24	91
3	25	111
…	…	…

Visualization of the dataset



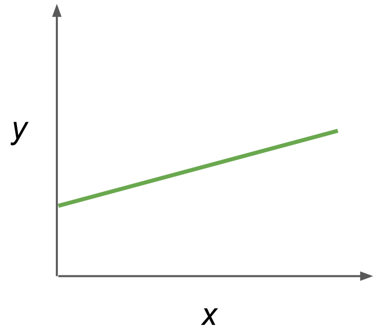
Predictive model

- We assume that risk is a linear function of BMI.
- In other words, for some unknown $\theta_0, \theta_1 \in \mathbb{R}$, we have

$$y = \theta_1 \cdot x + \theta_0,$$

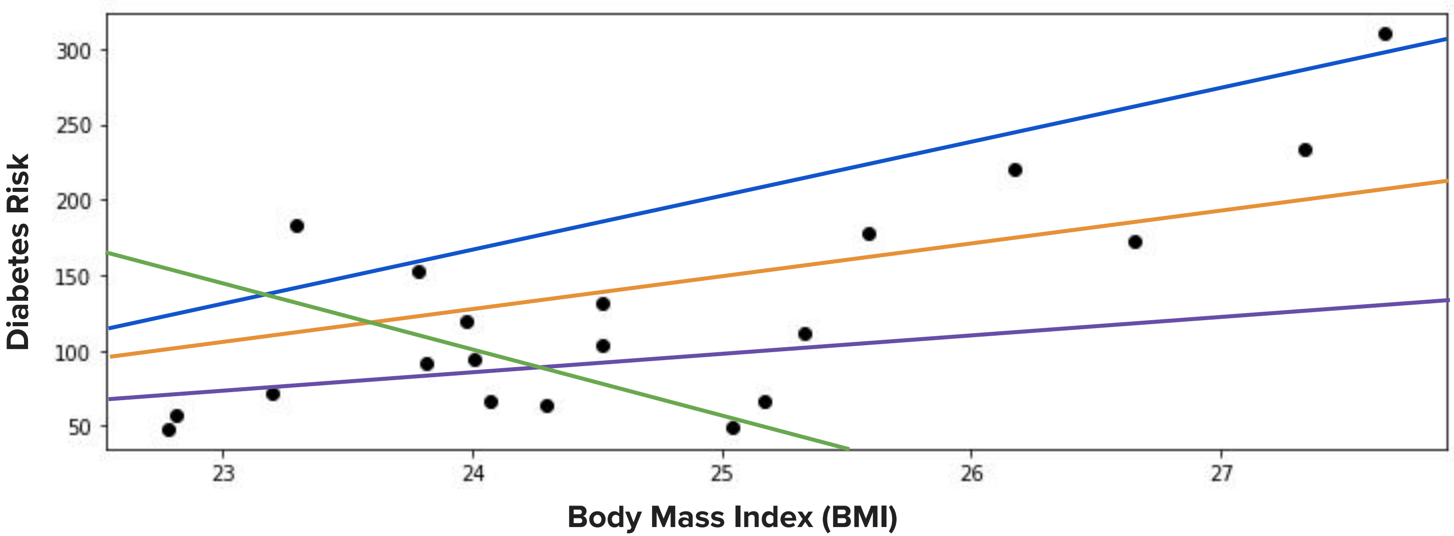
where x is the **BMI** and y is the **diabetes risk score**

- What does the curve of $y = \theta_1 \cdot x + \theta_0,$ look like?
  - A linear line!
  - θ_1 is the slope, and θ_0 is the intercept (or bias) of the line, respectively



Visualization of the dataset

- A specific value of parameter pair (θ_0, θ_1) defines a unique linear line



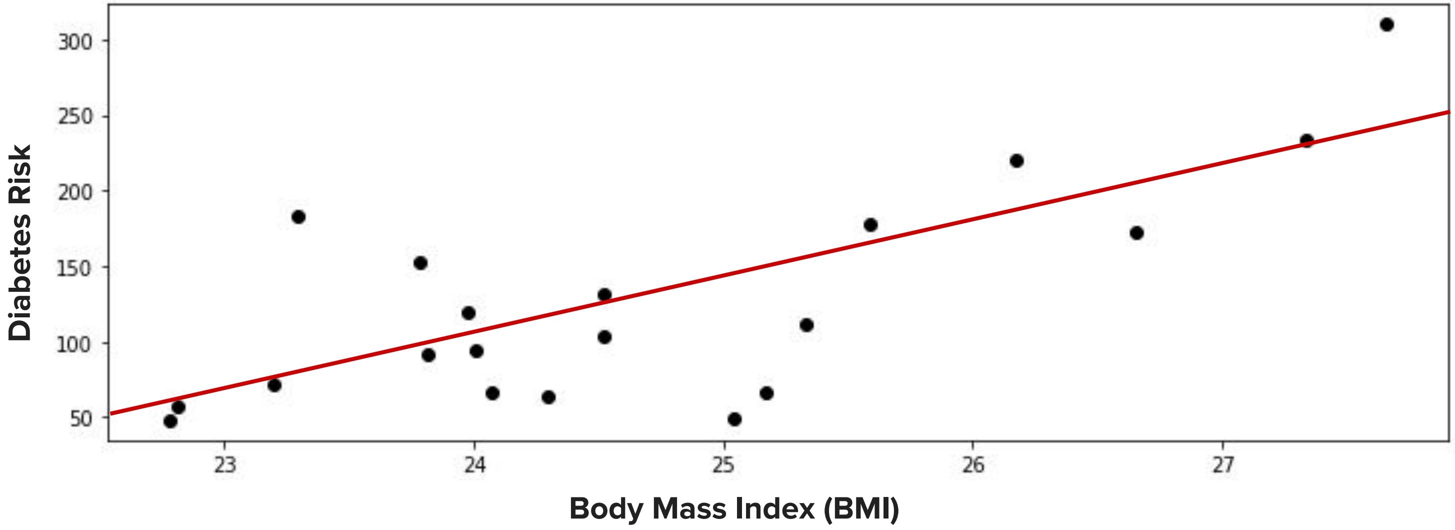
Which line is more consistent with data?

Optimal predictive model

- Suppose the best parameter is (θ^*_0, θ^*_1). This defines the predictive model f

$$f(x) = \theta_1^* \cdot x + \theta_0^*,$$

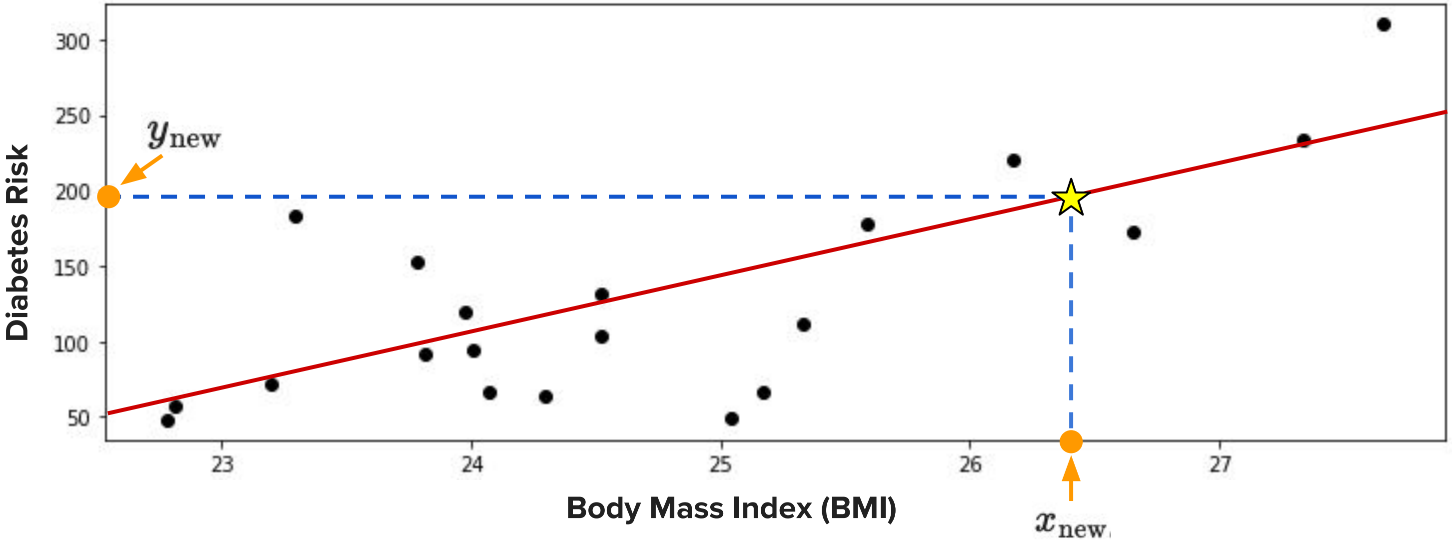
- We will see how to compute the best parameters in a few slides



Making new predictions

- Given a new dataset of patients with a known BMI, we can use this model to estimate their diabetes risk.
- Given an x_{new}, we can output prediction y_{new} as

$$y_{\text{new}} = f(x_{\text{new}}) = \theta_1^* \cdot x_{\text{new}} + \theta_0$$



Recipe: linear regression for diabetes risk prediction

- **Dataset**

$$\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid i = 1, 2, \ldots, n\}$$

Each $x^{(i)}$ denotes an input (e.g., the measurements for patient i), and each $y^{(i)} \in \mathcal{Y}$ is a target (e.g., the diabetes risk).

Together, $(x^{(i)}, y^{(i)})$ form a *training example*.

- **Model**

$$y = \theta_1 \cdot x + \theta_0,$$

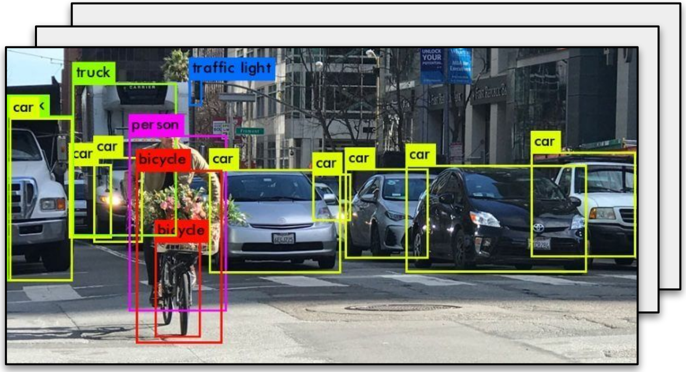
- **Optimization**
  - Finding the "best" parameters θ^*_0 and θ^*_1: $f(x) = \theta_1^* \cdot x + \theta_0^*,$

- **Predictions**

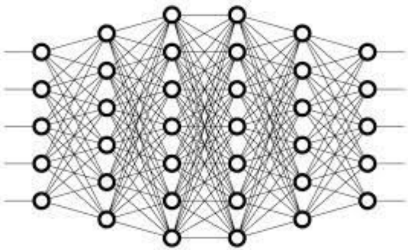
$$y_{\text{new}} = f(x_{\text{new}}) = \theta_1^* \cdot x_{\text{new}} + \theta_0$$

Recipe: Object detection

1. Dataset



2. Model



More complex model

3. Optimization

- Find the “best” value for every parameter in the model

4. Predictions

Remarks: Supervised learning

- The ML problems we have seen so far (diabetes prediction, object detection) are called **supervised learning**.

- Typical workflow:

  1. First, we collect a dataset of ***labeled*** training examples.
  2. We train a model to output predicted **labels** (*y*) based on input **features** (*x*).
  3. When the model sees new, similar data, it will also be accurate.

- We will learn **unsupervised learning** later in this course
  - Learn patterns from ***unlabeled*** data

features (*x*) **labels** (*y*)

BMI	Risk
27	233
24	91
25	111
...	...

We can have more than one feature!

- **Dataset**

$$\mathcal{D} = \{(\underbrace{x^{(i)}}_{\text{features}}, \underbrace{y^{(i)}}_{\text{labels}}) \mid i = 1, 2, \ldots, n\}$$

- In the diabetes prediction example, we can have more features besides **BMI**, including age, sex, and blood pressure.

	age	sex	bmi	bp
0	0.038076	0.050680	0.061696	0.021872
1	-0.001882	-0.044642	-0.051474	-0.026328
2	0.085299	0.050680	0.044451	-0.005671
3	-0.089063	-0.044642	-0.011595	-0.036656
4	0.005383	-0.044642	-0.036385	0.021872

target
233.0
91.0
111.0
152.0
120.0

Features of individual #4

Label of individual #4

We can have more than one feature!

- **Dataset**

$$\mathcal{D} = \{(\underbrace{x^{(i)}}_{\text{features}}, \underbrace{y^{(i)}}_{\text{labels}}) \mid i = 1, 2, \ldots, n\}$$

- When we have more than one feature,

$x^{(i)} \in \mathcal{X}$ is a d-dimensional vector of the form

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix}$$

How about the model?

- When we have only a *single* feature for an individual

$$y = \theta_1 \cdot x + \theta_0,$$

- When we have *multiple* features for an individual

$$y = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \ldots + \theta_d \cdot x_d$$

How to find the “best” parameters?

- How to decide predictive model M is good or not?

$$f(x) = \theta_1 \cdot x + \theta_0,$$

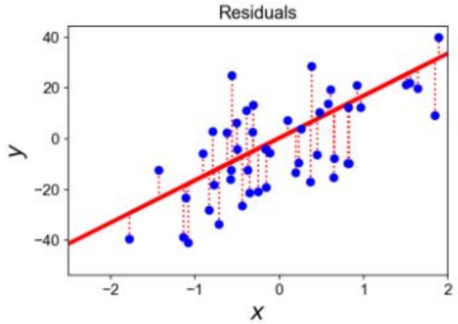
- Objective function (or loss function)

$$J(f) : \mathcal{M} \to [0, \infty),$$

which describes the extent to which f "fits" the data $\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid i = 1, 2, \ldots, n\}.$

- Prediction error!
    - Mean squared error (MSE):

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(f_\theta(x^{(i)}) - y^{(i)} \right)^2$$



How to find the “best” parameters?

- Objective function (or loss function)

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(f_\theta(x^{(i)}) - y^{(i)} \right)^2$$

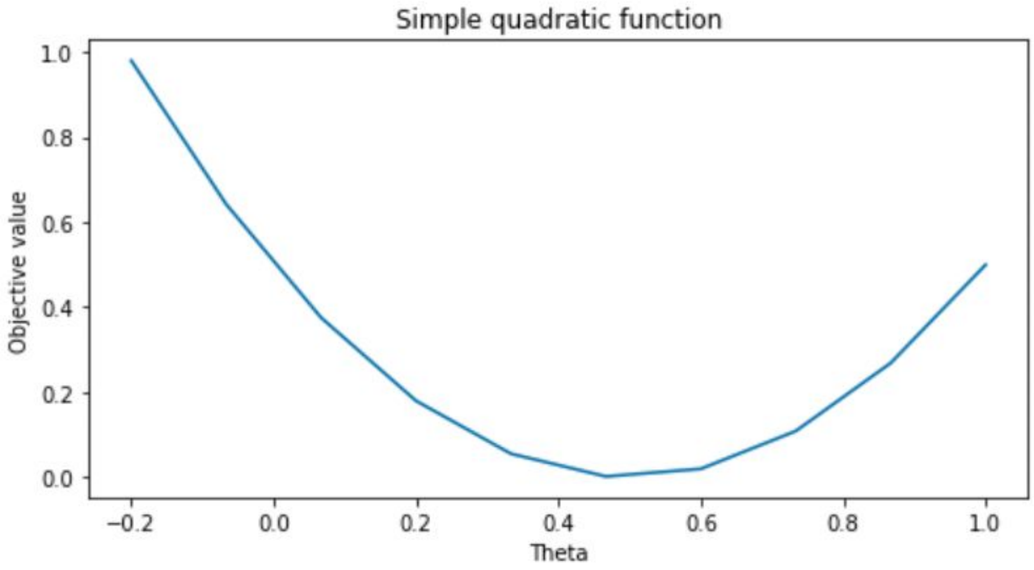
- The best parameters can be obtained by solving the following optimization problem

$$\min_{\theta \in \mathbb{R}} \frac{1}{2n} \sum_{i=1}^{n} \left(f_\theta(x^{(i)}) - y^{(i)} \right)^2$$

How to solve this optimization problem?

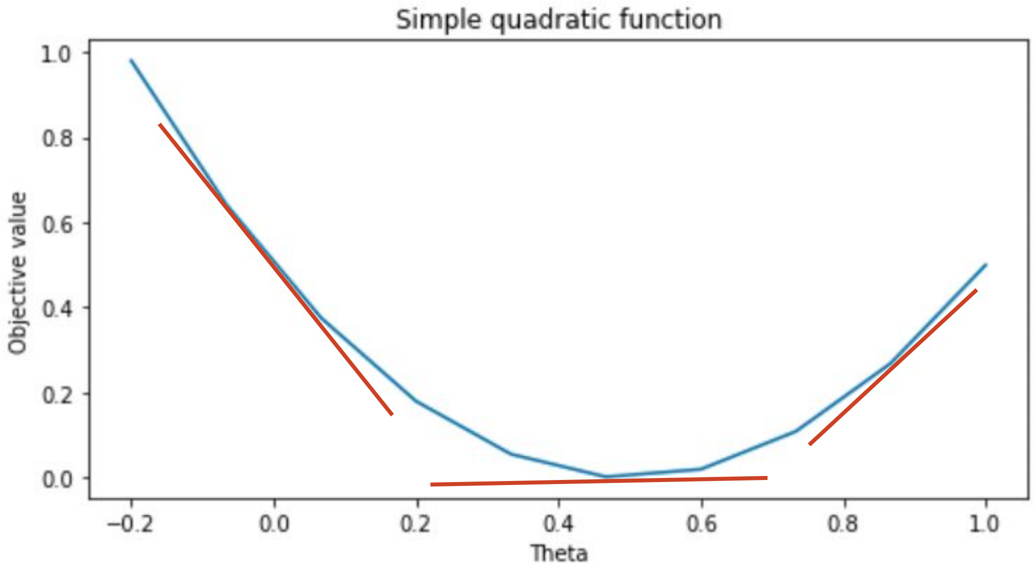
Gradient descent

Review: Derivatives



$$\frac{df(\theta_0)}{d\theta}$$

Review: Derivatives



$$\frac{df(\theta_0)}{d\theta}$$

The derivative of a function

- gives the **slope of the line tangent** to the function at any point
- gives the **instantaneous rate of change** of the function at any point

Review: Partial Derivatives & Gradient

The partial derivative

$$\frac{\partial f(\theta)}{\partial \theta_j}$$

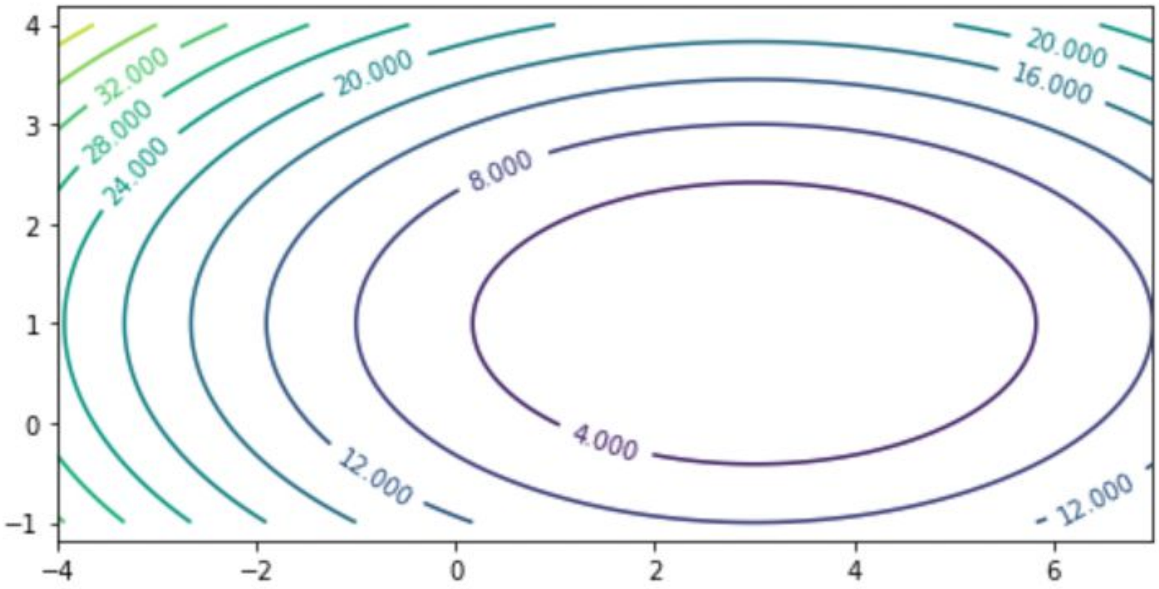
of a multivariate function $f : \mathbb{R}^d \to \mathbb{R}$ is the derivative of f with respect to θ_j while all the other dimensions θ_k for $k \neq j$ are fixed.

The gradient ∇f is the vector of all the partial derivatives:

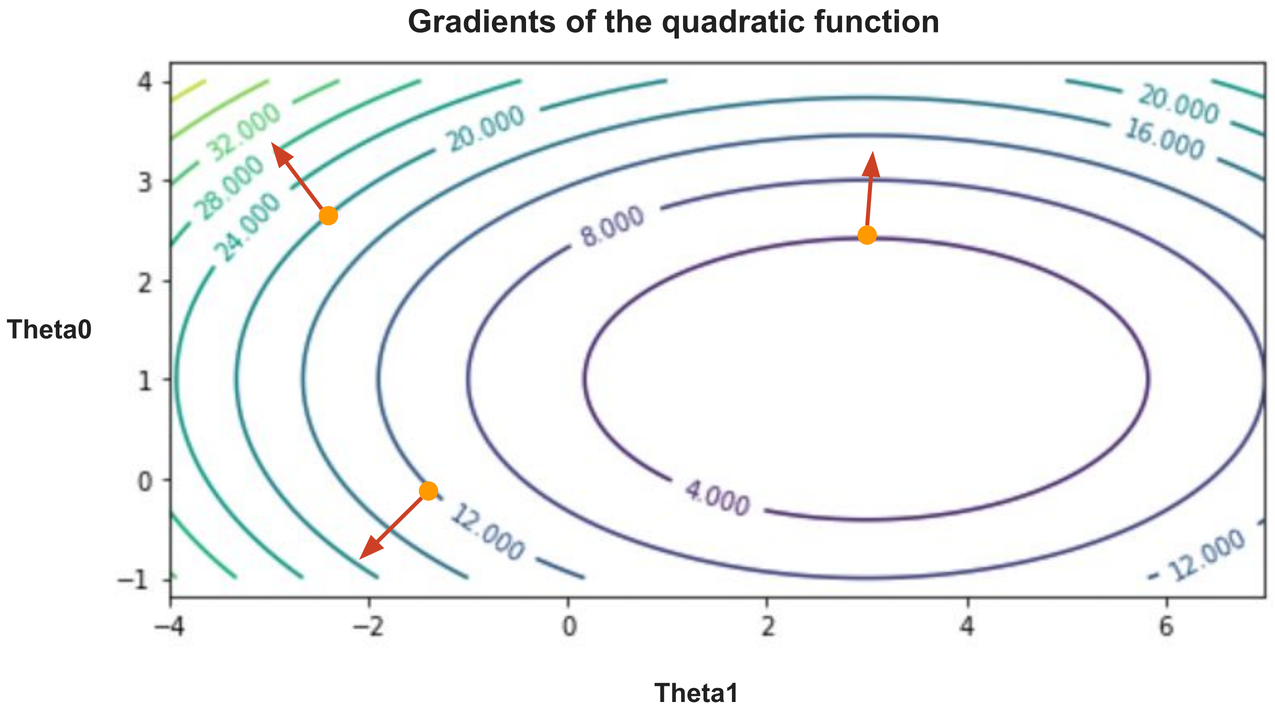
$$\nabla f(\theta) = \begin{bmatrix} \frac{\partial f(\theta)}{\partial \theta_1} \\ \frac{\partial f(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\theta)}{\partial \theta_d} \end{bmatrix}.$$

The j-th entry of the vector $\nabla f(\theta)$ is the partial derivative $\frac{\partial f(\theta)}{\partial \theta_i}$ of f with respect to the j-th component of θ.

A 2D example


32.000
28.000
24.000
20.000
8.000
4.000
12.000
20.000
16.000
12.000
4
3
2
1
0
−1
−4
−2
0
2
4
6

A 2D example



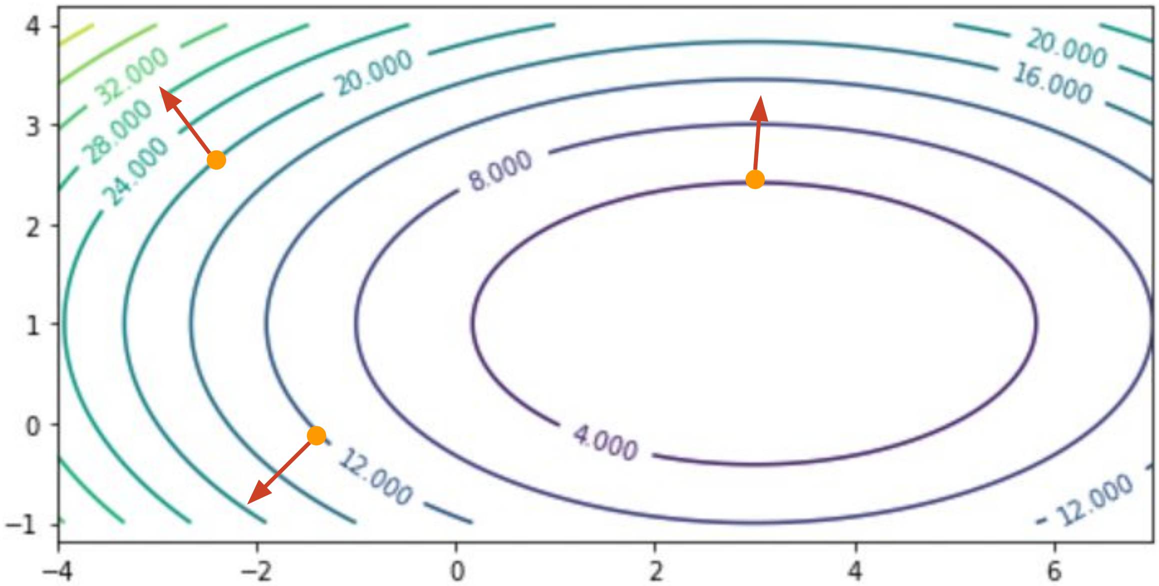
$$\nabla_\theta f(\theta_0) = \begin{bmatrix} \frac{\partial f(\theta_0)}{\partial \theta_1} \\ \frac{\partial f(\theta_0)}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\theta_0)}{\partial \theta_d} \end{bmatrix}$$

The direction of the gradient is the direction in which the function **increases most steeply**

A 2D example

Gradients of the quadratic function

Theta0



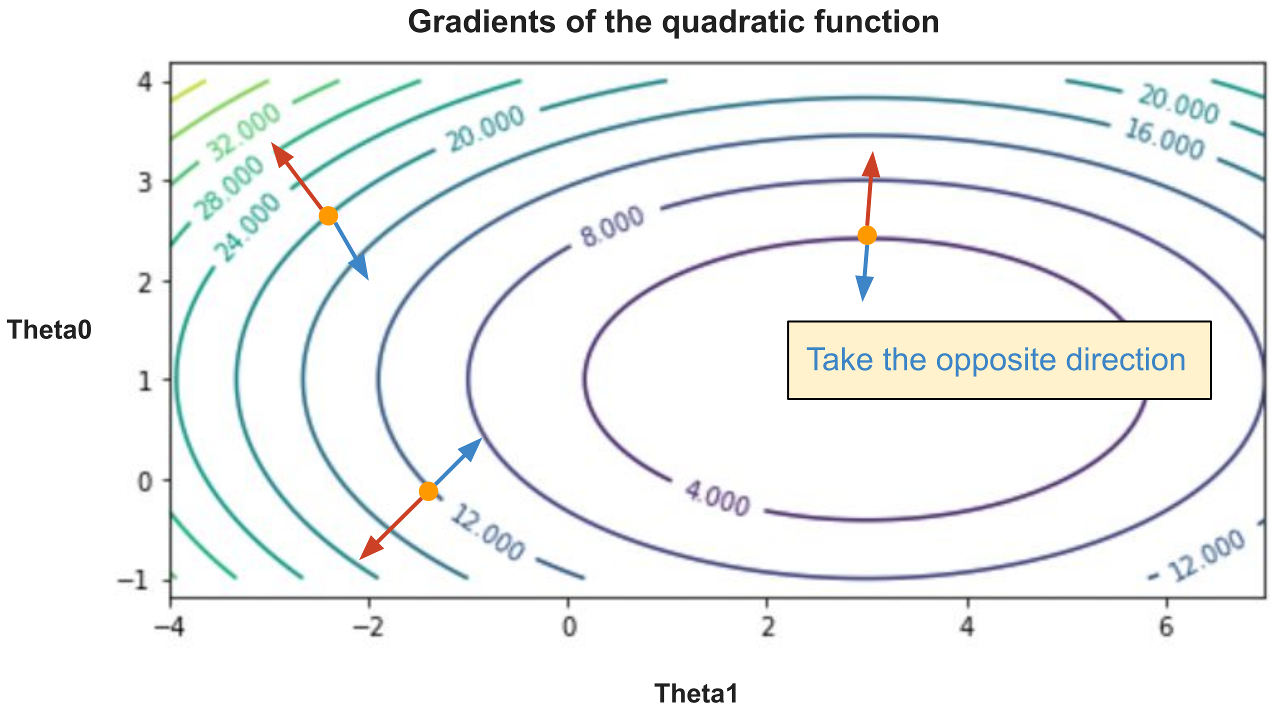
Theta1

$$\nabla_{\theta} f(\theta_0) = \begin{bmatrix} \frac{\partial f(\theta_0)}{\partial \theta_1} \\ \frac{\partial f(\theta_0)}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\theta_0)}{\partial \theta_d} \end{bmatrix}$$

The direction of the gradient is the direction in which the function ~~**increases**~~ **most steeply**

But we need to "**minimize**" the loss

A 2D example



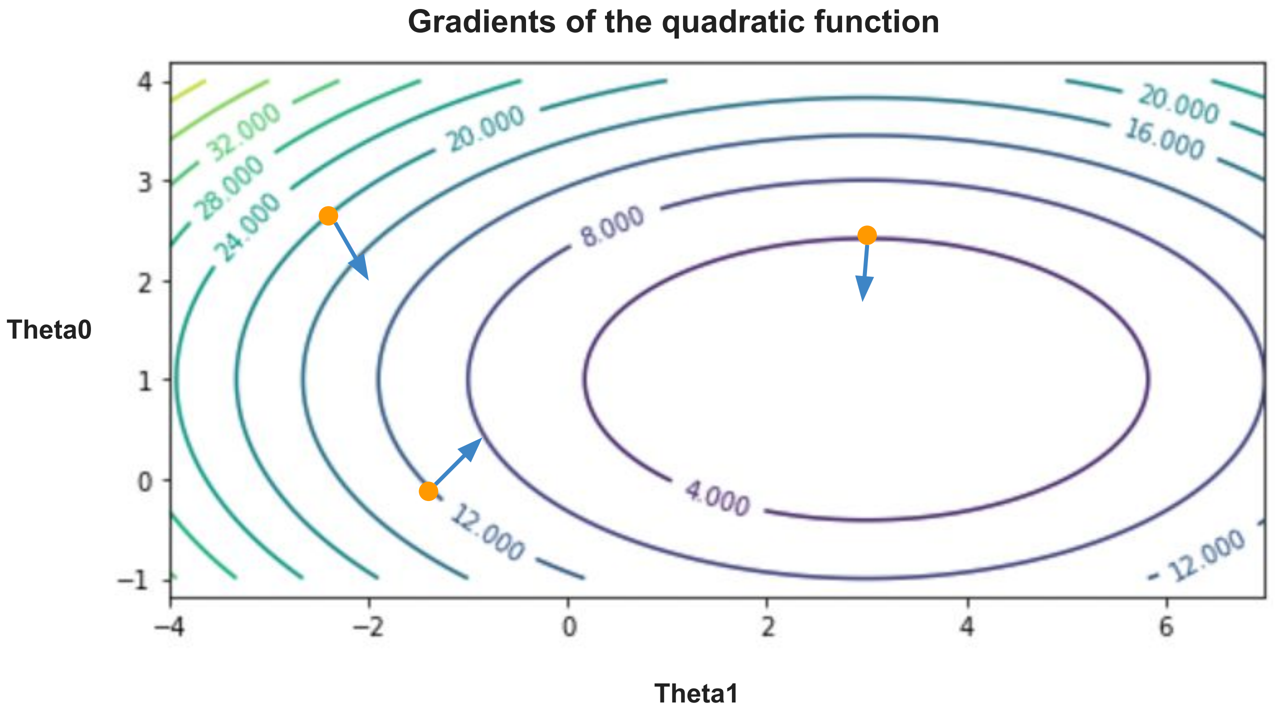
$$\nabla_\theta f(\theta_0) = \begin{bmatrix} \frac{\partial f(\theta_0)}{\partial \theta_1} \\ \frac{\partial f(\theta_0)}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\theta_0)}{\partial \theta_d} \end{bmatrix}$$

The direction of the gradient is the direction in which the function ~~**increases**~~ **most steeply**

But we need to "**minimize**" the loss

Gradient descent: intuition

Repeatedly obtaining the gradient to determine the direction in which the function decreases most steeply and take a step in that direction.



$$\nabla_\theta f(\theta_0) = \begin{bmatrix} \frac{\partial f(\theta_0)}{\partial \theta_1} \\ \frac{\partial f(\theta_0)}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\theta_0)}{\partial \theta_d} \end{bmatrix}$$

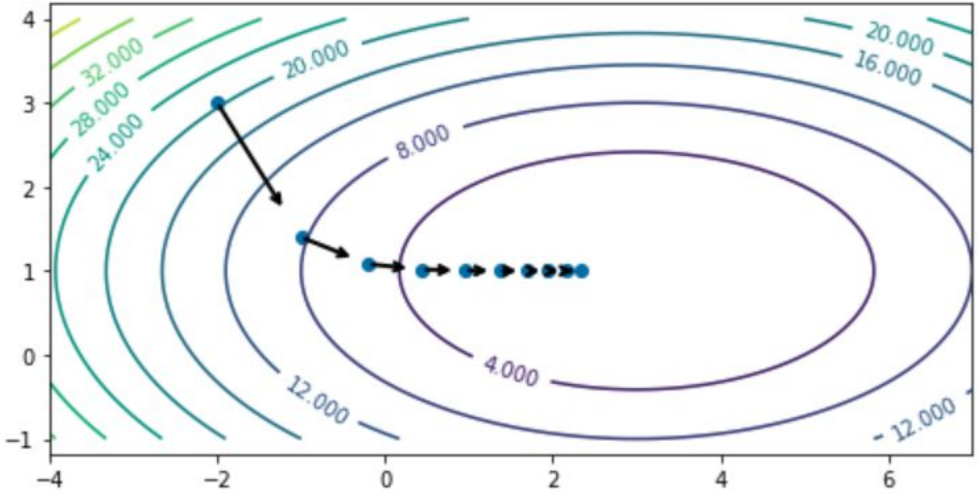
The direction of the gradient is the direction in which the function **increases most steeply**

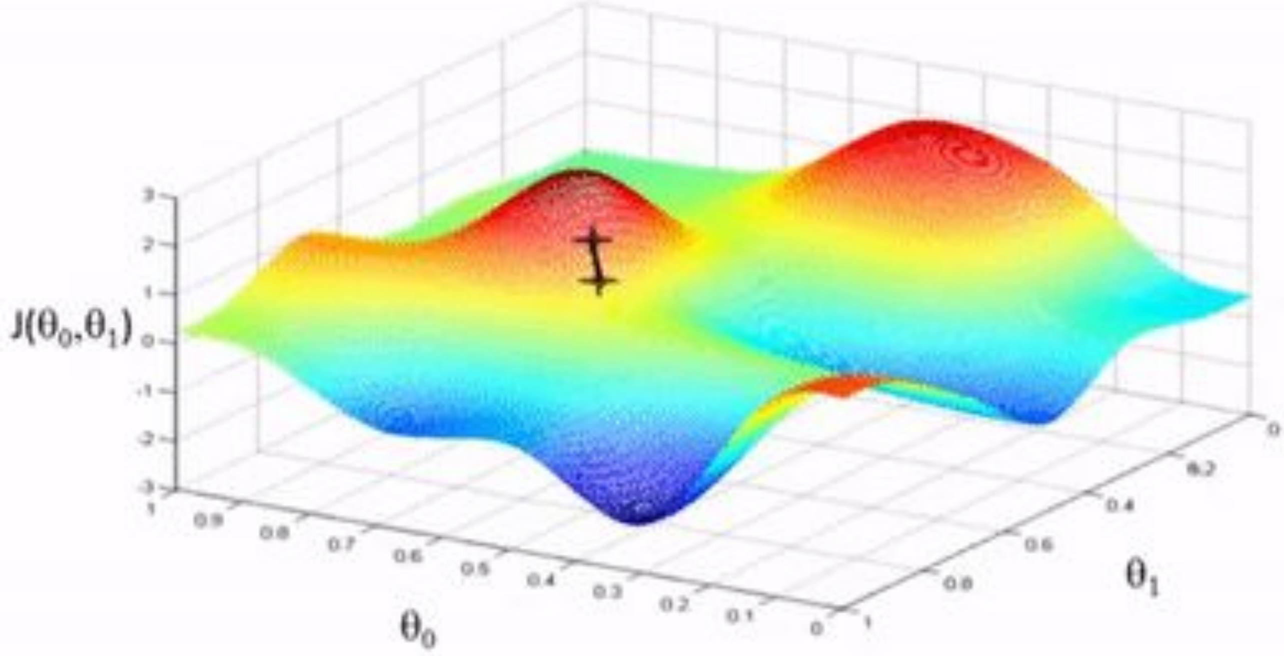
Gradient descent: algorithm

More formally, if we want to optimize $J(\theta)$, we start with an initial guess $\theta^{(0)}$ for the parameters and update the parameters at the *t*-th iteration ($\theta^{(t)}$) based on the parameters at the (*t*-1)-th iteration ($\theta^{(t-1)}$), until is no longer changing:

$$\theta^{(t)} = \theta^{(t-1)} - \alpha \cdot \nabla J(\theta^{(t-1)})$$

where a is the step size





Andrew Ng

Apply gradient descent to linear regression

Recall that a linear model has the form

$$y = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \ldots + \theta_d \cdot x_d$$

where $x \in \mathbb{R}^d$ is a vector of features and y is the target. The θ_j are the *parameters* of the model.

By using the notation $x_0 = 1$, we can represent the model in a vectorized form

$$f_\theta(x) = \sum_{j=0}^{d} \theta_j \cdot x_j = \theta^\top x.$$

We pick θ to minimize the mean squared error (MSE). Slight variants of this objective are also known as the residual sum of squares (RSS) or the sum of squared residuals (SSR).

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (y^{(i)} - \theta^\top x^{(i)})^2$$

Mean Squared Error: Partial Derivatives

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(y^{(i)} - \theta^\top x^{(i)})^2$$

Let's work out the derivatives for $\frac{1}{2}\left(f_\theta(x^{(i)}) - y^{(i)}\right)^2$, the MSE of a linear model f_θ for one training example $(x^{(i)}, y^{(i)})$, which we denote $J^{(i)}(\theta)$.

$$\begin{aligned}
\frac{\partial}{\partial\theta_j} J^{(i)}(\theta) &= \frac{\partial}{\partial\theta_j}\left(\frac{1}{2}\left(f_\theta(x^{(i)}) - y^{(i)}\right)^2\right) \\
&= \left(f_\theta(x^{(i)}) - y^{(i)}\right) \cdot \frac{\partial}{\partial\theta_j}\left(f_\theta(x^{(i)}) - y^{(i)}\right) \\
&= \left(f_\theta(x^{(i)}) - y^{(i)}\right) \cdot \frac{\partial}{\partial\theta_j}\left(\sum_{k=0}^{d}\theta_k \cdot x_k^{(i)} - y^{(i)}\right) \\
&= \left(f_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_j^{(i)}
\end{aligned}$$

Mean Squared Error: The Gradient

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(y^{(i)} - \theta^\top x^{(i)})^2$$

We can use this derivation to obtain an expression for the gradient of the MSE for a linear model

$$\nabla_\theta J^{(i)}(\theta) = \begin{bmatrix} \frac{\partial J^{(i)}(\theta)}{\partial \theta_0} \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_d} \end{bmatrix} = \begin{bmatrix} \left(f_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_0^{(i)} \\ \left(f_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_1^{(i)} \\ \vdots \\ \left(f_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_d^{(i)} \end{bmatrix} = \left(f_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$$

Note that the MSE over the entire dataset is $J(\theta) = \frac{1}{n}\sum_{i=1}^{n} J^{(i)}(\theta)$. Therefore:

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_d} \end{bmatrix} = \frac{1}{n}\sum_{i=1}^{n} \begin{bmatrix} \frac{\partial J^{(i)}(\theta)}{\partial \theta_0} \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_d} \end{bmatrix} = \frac{1}{n}\sum_{i=1}^{n} \left(f_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$$

Write everything in the language of linear algebra ...

Matrix form of input data

Machine learning algorithms are most easily defined in the language of linear algebra. Therefore, it will be useful to represent the entire dataset as one matrix $X \in \mathbb{R}^{n \times d}$, of the form:

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \vdots \\ x_1^{(n)} & x_2^{(n)} & \dots & x_d^{(n)} \end{bmatrix} = \begin{bmatrix} - & (x^{(1)})^\top & - \\ - & (x^{(2)})^\top & - \\ & \vdots & \\ - & (x^{(n)})^\top & - \end{bmatrix}.$$

Similarly, we can vectorize the target variables into a vector $y \in \mathbb{R}^n$ of the form

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}.$$

Matrix form of squared error

Recall that we may fit a linear model by choosing θ that minimizes the squared error:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \theta^\top x^{(i)})^2$$

We can write this sum in matrix-vector form as:

$$J(\theta) = \frac{1}{2}(y - X\theta)^\top (y - X\theta) = \frac{1}{2} \|y - X\theta\|^2,$$

where X is the design matrix and $\|\cdot\|$ denotes the Euclidean norm.

Gradient of the squared error

Objective function: $J(\theta) = \frac{1}{2}(y - X\theta)^\top (y - X\theta) = \frac{1}{2}\|y - X\theta\|^2,$

We can compute the gradient of the mean squared error as follows.

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta - y)^\top (X\theta - y) \\
&= \frac{1}{2}\nabla_\theta \left((X\theta)^\top (X\theta) - (X\theta)^\top y - y^\top (X\theta) + y^\top y\right) \\
&= \frac{1}{2}\nabla_\theta \left(\theta^\top (X^\top X)\theta - 2(X\theta)^\top y\right) \\
&= \frac{1}{2}\left(2(X^\top X)\theta - 2X^\top y\right) \\
&= (X^\top X)\theta - X^\top y
\end{aligned}
$$

We used the facts that $a^\top b = b^\top a$ (line 3), that $\nabla_x b^\top x = b$ (line 4), and that $\nabla_x x^\top A x = 2Ax$ for a symmetric matrix A (line 4).

Normal equations

Gradient: $\nabla_\theta J(\theta) = (X^\top X)\theta - X^\top y$

Setting the above derivative to zero, we obtain the *normal equations*:

$$(X^\top X)\theta = X^\top y.$$

Hence, the value θ^* that minimizes this objective is given by:

$$\theta^* = (X^\top X)^{-1} X^\top y.$$

This problem/algorithm is known as **Ordinary Least Squares**

- OLS is a type of linear least squares method for choosing the unknown parameters in a linear regression model by the principle of least squares: minimizing the sum of the squares of the differences between the observed and predicted values of y.

Why do we need gradient descent given the fact that we can compute the exact solution in a closed form?

Conclusion

- Linear regression

$$f_\theta(x) = \sum_{j=0}^{d} \theta_j \cdot x_j = \theta^\top x.$$

- Loss function

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (y^{(i)} - \theta^\top x^{(i)})^2$$

- Optimization
  - Gradient descent
  - Closed-form