CS60010: Deep Learning

Spring 2023

Sudeshna Sarkar

Module 2 Part B

Linear Models for Classification
Gradient Descent

19 Jan 2023

Linear Regression: Analytic Solution

$$L = \sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2 = \sum_{i=1}^{m}\left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

Since the loss is differentiable, we set

$$\frac{dL}{d\theta_0} = 0 \quad \text{and} \quad \frac{dL}{d\theta_1} = 0$$

We want the loss-minimizing values of $\boldsymbol{\theta}$, so we set

$$\frac{dL}{d\theta_1} = 0 = 2\theta_1 \sum_{i=1}^{N}\left(x^{(i)}\right)^2 + 2\theta_0 \sum_{i=1}^{N} x^{(i)} - 2\sum_{i=1}^{N} x^{(i)} y^{(i)}$$

$$\frac{dL}{d\theta_0} = 0 = 2\theta_1 \sum_{i=1}^{N} x^{(i)} + 2\theta_0 N - 2\sum_{i=1}^{N} y^{(i)}$$

These being linear equations of θ, have a unique closed form solution

$$\theta_1 = \frac{m\sum_{i=1}^{m} y^{(i)} x^{(i)} - \left(\sum_{i=1}^{m} x^{(i)}\right)\left(\sum_{i=1}^{m} y^{(i)}\right)}{m\sum_{i=1}^{m}(x^{(i)})^2 - \left(\sum_{i=1}^{m} x^{(i)}\right)^2}$$

$$\theta_0 = \frac{1}{m}\left(\sum_{i=1}^{m} y^{(i)} - \theta_1 \sum_{i=1}^{m} x^{(i)}\right)$$

Multivariate Linear Regression

$x \in \mathcal{R}^d$

$$\hat{y} = f(x;\ \theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ \ldots + \theta_d x_d$$

$$\begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix} = \begin{bmatrix} {x_0}^{(1)} & {x_1}^{(1)} & {x_2}^{(1)} & \cdots & {x_d}^{(1)} \\ {x_0}^{(2)} & {x_1}^{(2)} & {x_2}^{(2)} & \cdots & {x_d}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ {x_0}^{(m)} & {x_1}^{(m)} & {x_2}^{(m)} & \cdots & {x_d}^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_d \end{bmatrix}$$

$$\hat{\mathrm{y}} = \mathbf{X\theta}$$

Define $x_0 = 1$ $\qquad f(x;\ \theta) = \theta^T \mathbf{x}$

Cost Function:

$$J(\boldsymbol{\theta}) = J(\theta_0, \theta_1, \ldots, \theta_d) = \frac{1}{m}\left(\boldsymbol{\theta}^T \mathrm{x}^{(i)} - y^{(i)}\right)^2$$

Multivariate Linear Regression

$$J(\boldsymbol{\theta}) = \frac{1}{m}\left(\boldsymbol{\theta}^T \mathrm{x}^{(i)} - y^{(i)}\right)^2 = \frac{1}{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

$$= \frac{1}{m}\|\hat{y} - y\|_2^2 = \frac{1}{m}(\hat{y} - y)^T(\hat{y} - y)$$

$$= \frac{1}{m}(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

$$= \frac{1}{m}\{\theta^T(X^T X)\theta - \theta^T X^T y - y^T X\,\theta + y^T Y\}$$

$$= \frac{1}{m}\{\theta^T(X^T X)\theta - (X^T y)^T\theta - (X^T y)^T\,\theta + y^T Y\}$$

$$= \frac{1}{m}\{\theta^T(X^T X)\theta - 2(X^T y)^T\theta + y^T Y\}$$

Multivariate Linear Regression

- Equating the gradient of the cost function to 0,

$$\nabla_\theta J(\,\boldsymbol{\theta}\,) = \frac{1}{m}\{2\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} - 2\mathbf{X}^T\mathbf{y} + 0\} = 0$$

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} - \mathbf{X}^T\mathbf{y} = 0$$

$$\boldsymbol{\theta} = (\mathbf{X}^T\mathbf{X}\,)^{-1}\mathbf{X}^T\mathbf{y}$$

This gives a closed form solution, but another option is to use iterative solution

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j{}^{(i)}$$

Partial derivatives

- Let $y = f(\mathbf{x}) = f(x_1, x_2, \dots, x_n)$ be a multivariate function with *n* variables
  - The mapping is $f: \mathbb{R}^n \rightarrow \mathbb{R}$
- The ***partial derivative*** of *y* with respect to its *i*th parameter x_i is

$$\frac{\partial y}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, x_2, \dots, x_i + h, \dots, x_n) - f(x_1, x_2, \dots, x_i, \dots, x_n)}{h}$$

- To calculate $\frac{\partial y}{\partial x_i}$, we can treat $x_1, x_2, \dots, x_{i-1}, \quad x_{i+1} \dots, x_n$ as constants and calculate the derivative of *y* only with respect to x_i
- For notation of partial derivatives, the following are equivalent:

$$\frac{\partial y}{\partial x_i} = \frac{\partial f}{\partial x_i} = \frac{\partial}{\partial x_i} f(\mathbf{x}) = f_{x_i} = f_i = D_i f = D_{x_i} f$$

Multidimensional derivative: Gradient

Gradient vector: The gradient of the multivariate function $f(\mathbf{x})$ with respect to the *n*-dimensional input vector $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$, is a vector of *n* partial derivatives

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_n}\right]^T$$

- In ML, the gradient descent algorithm relies on the opposite direction of the gradient of the loss function $\mathcal{L}$ with respect to the model parameters θ ($\nabla_\theta \mathcal{L}$) for minimizing the loss function

$$f(x, y, z) = x^3zy^2 - xz + yz.$$

Which of the following is the gradient vector $\begin{pmatrix} \partial f/\partial x \\ \partial f/\partial y \\ \partial f/\partial z \end{pmatrix}$?

(a) $\begin{pmatrix} 3x^2y^2 \\ y \\ 3x^2y^2 - x \end{pmatrix}$

(b) $\begin{pmatrix} 3x^2zy^2 - z \\ 2x^3zy + z \\ x^3y^2 - x + y \end{pmatrix}$

(c) $\begin{pmatrix} 2x^3zy + z \\ x^3y^2 + y \\ 3x^2zy^2 - z + y \end{pmatrix}$

Gradient as directional derivative

$$f(x+h, y+h) \approx f(x,y) + h_1 \left.\frac{\partial f}{\partial x}\right|_{(x,y)} + h_2 \left.\frac{\partial f}{\partial y}\right|_{(x,y)}$$

$$= f(x,y) + (h_1, h_2) \cdot \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = f(x,y) + \vec{h} \cdot \nabla f$$

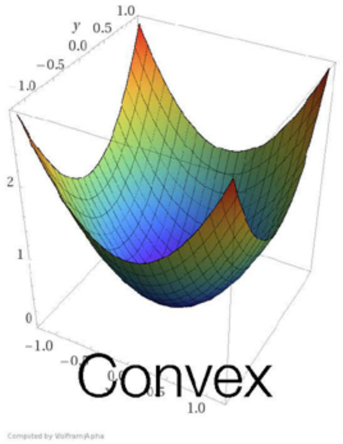
Gradient Descent: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \underset{\substack{\uparrow \\ \text{learning rate}}}{\eta} \nabla f$

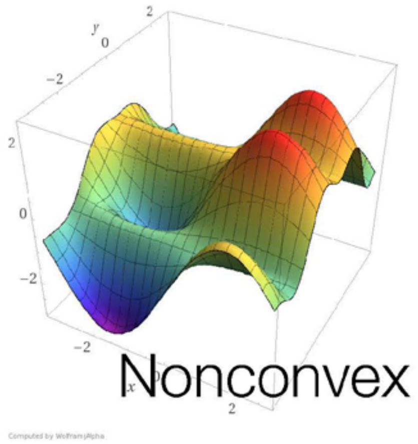
Gradient Descent Algorithm

Starting at some suitable $w^{(0)} \in \Re^d$ do for $t = 0,1,\ldots,T-1$
$\{ w^{(t+1)} \leftarrow w^{(t)} - \eta \cdot \nabla f|_{w^{(t)}} \}$

- "Like water flowing down a slope."
- Stops making progress when $\nabla f = 0$ ("Stationary point")
- Will reach minimizer of $f(x)$ if f is a convex function
- No guarantees for non convex f. At best find "local minima." (Finding global minimum is NP-hard in worst case).

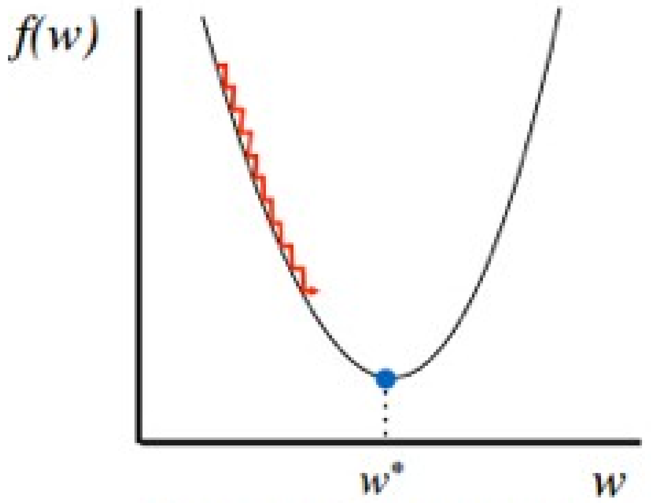
convex vs nonconvex losses was a big argument in AI/ML 15+ years ago. (Nonconvex side won)


Convex

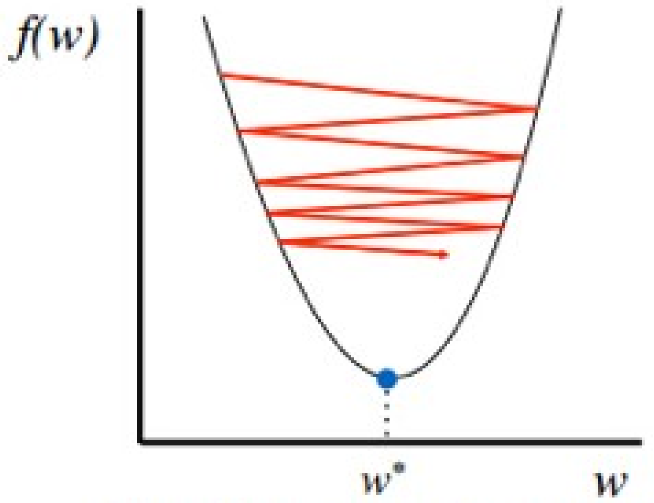

Nonconvex

Step Size

- Determines how quickly training loss goes down; hence “learning rate”



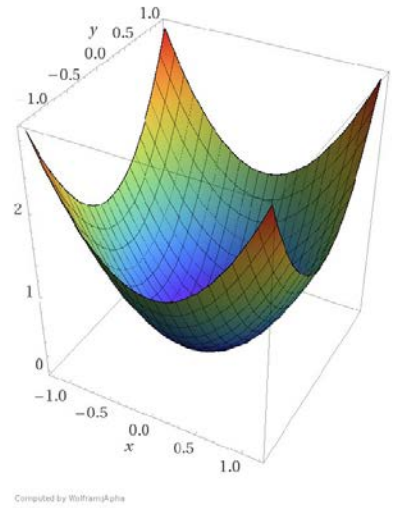
Too small: converge very slowly

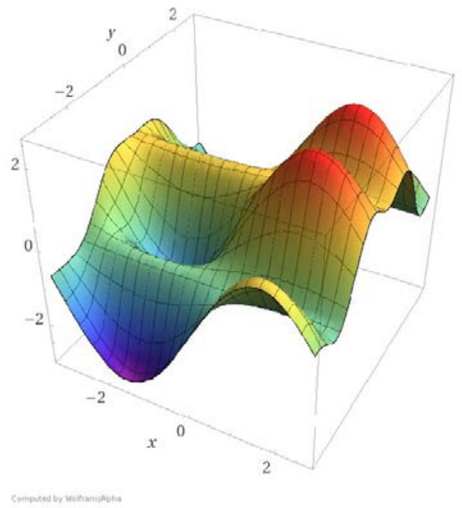


Too big: overshoot and even diverge

Importance of Initialization

- Simple convex functions have unique optimum.
- But in practice loss functions are non convex and have multiple optima (test accuracies all over the map). Initialization (and even choice of LR) affects where you end up!
- With non convex loss (e.g., in deep learning) good initialization is a must. No good theory for it. In deep learning, Gaussian initialization is common…





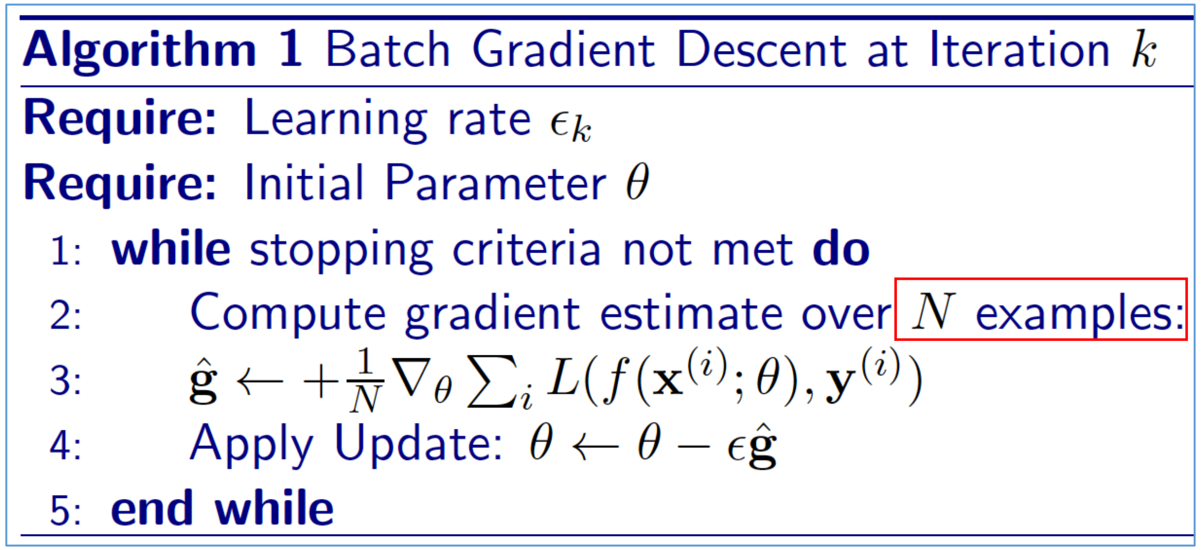
- Gradient is a linear operator.

$$\nabla\left(f_1(\vec{\theta}) + f_2(\vec{\theta})\right) = \nabla\left(f_1(\vec{\theta})\right) + \nabla\left(f_2(\vec{\theta})\right)$$

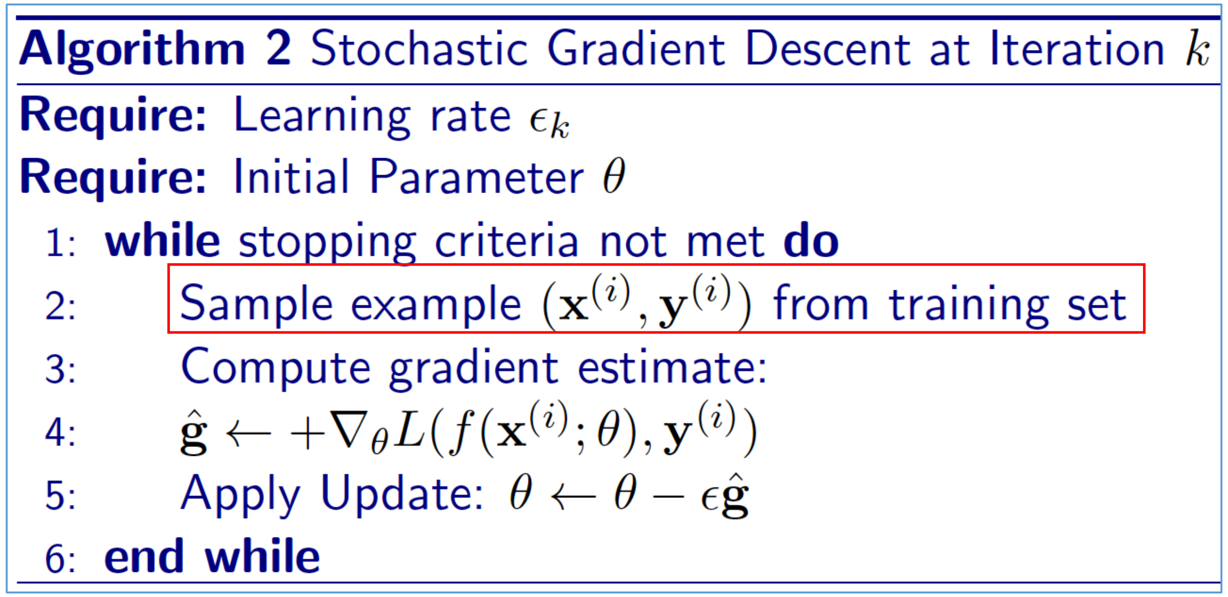
- Training Loss = Sum of Losses on individual datapoints.
- Gradient of training loss = sum of gradients of loss on individual datapoints.
- For large datasets evaluating exact derivative gets expensive
- **Stochastic GD:** Estimate the sum via a small random sample of datapoints, and use it instead of exact gradient. (Like an opinion poll over datapoints.)

$$\begin{pmatrix} \cdot \\ \cdot \\ \vdots \\ \cdot \end{pmatrix} = \begin{pmatrix} \cdot \\ \cdot \\ \vdots \\ \cdot \end{pmatrix} + \begin{pmatrix} \cdot \\ \cdot \\ \vdots \\ \cdot \end{pmatrix} + \cdots + \begin{pmatrix} \cdot \\ \cdot \\ \vdots \\ \cdot \end{pmatrix}$$

Batch, Stochastic and Mini-batch Stochastic Gradient Descent

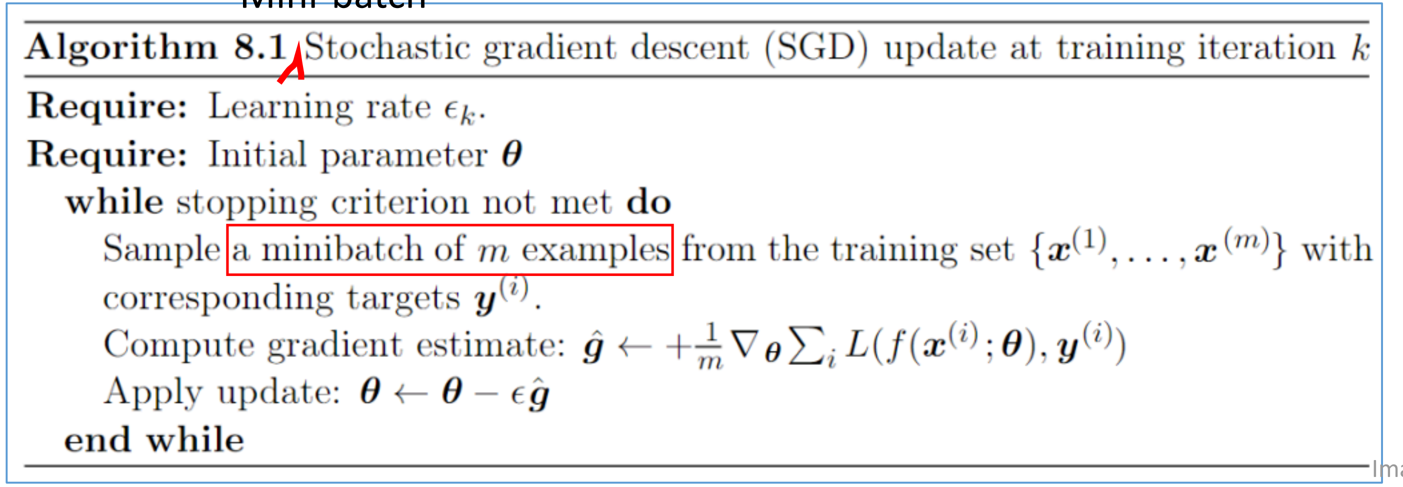


```
Algorithm 1 Batch Gradient Descent at Iteration $k$
Require: Learning rate $\epsilon_k$
Require: Initial Parameter $\theta$
1: while stopping criteria not met do
2:     Compute gradient estimate over $N$ examples:
3:     $\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta),\mathbf{y}^{(i)})$
4:     Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
5: end while
```



```
Algorithm 2 Stochastic Gradient Descent at Iteration $k$
Require: Learning rate $\epsilon_k$
Require: Initial Parameter $\theta$
1: while stopping criteria not met do
2:     Sample example $(\mathbf{x}^{(i)},\mathbf{y}^{(i)})$ from training set
3:     Compute gradient estimate:
4:     $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)};\theta),\mathbf{y}^{(i)})$
5:     Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
6: end while
```

Mini-batch



```
Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration $k$
Require: Learning rate $\epsilon_k$.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),\boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$
  end while
```

Images courtesy: Shubhendu Trivedi et. Al, Goodfellow et. al..

Iterative Gradient Descent for Regression

- Iterative Gradient Descent needs to perform many iterations and need to choose a stepsize parameter judiciously. But it works equally well even if the number of features (d) is large.
- For the least square solution, there is no need to choose the step size parameter or no need to iterate. But, evaluating $(\mathbf{X}^T\mathbf{X})^{-1}$ can be slow if d is large.

The loss “landscape”

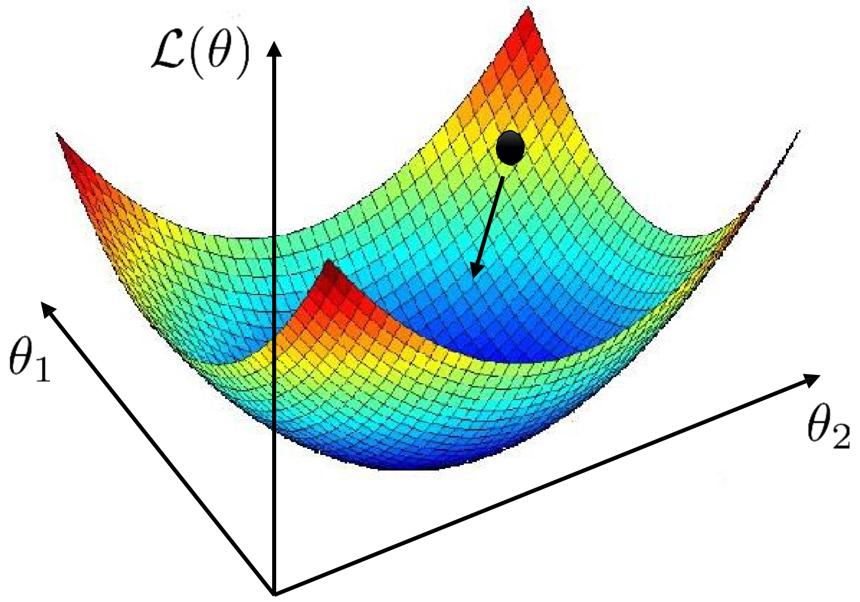
$$\theta^\star \leftarrow \arg\min_\theta \underbrace{-\sum_i \log p_\theta(y_i|x_i)}_{\mathcal{L}(\theta)}$$

let’s say θ is 2D

An algorithm:

1. Find a *direction* v where $\mathcal{L}(\theta)$ decreases
2. $\theta \leftarrow \theta + \alpha v$

some small constant called “learning rate” or “step size”

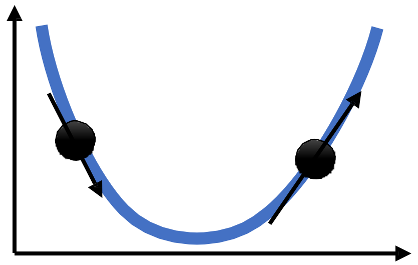


Gradient descent

An algorithm:

1. Find a *direction* v where $\mathcal{L}(\theta)$ decreases
2. $\theta \leftarrow \theta + \alpha v$

Which way does $\mathcal{L}(\theta)$ decrease?



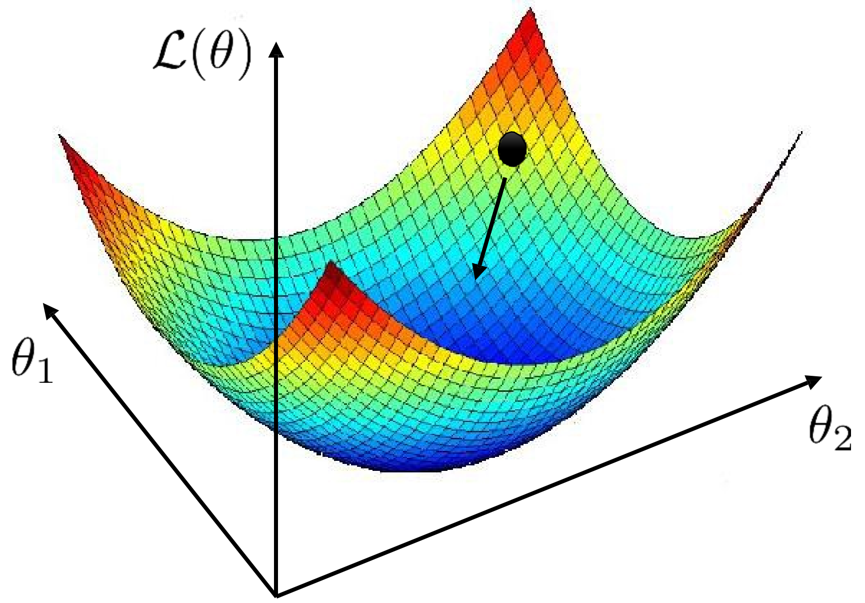
negative slope = go to the right

positive slope = go to the left

in general:

for each dimension, go in the direction opposite the slope **along that dimension**

$$v_1 = -\frac{d\mathcal{L}(\theta)}{d\theta_1} \quad v_2 = -\frac{d\mathcal{L}(\theta)}{d\theta_2}$$ etc.



gradient:

$$\nabla_\theta \mathcal{L}(\theta) = \begin{pmatrix} \frac{d\mathcal{L}(\theta)}{d\theta_1} \\ \frac{d\mathcal{L}(\theta)}{d\theta_2} \\ \vdots \\ \frac{d\mathcal{L}(\theta)}{d\theta_n} \end{pmatrix}$$

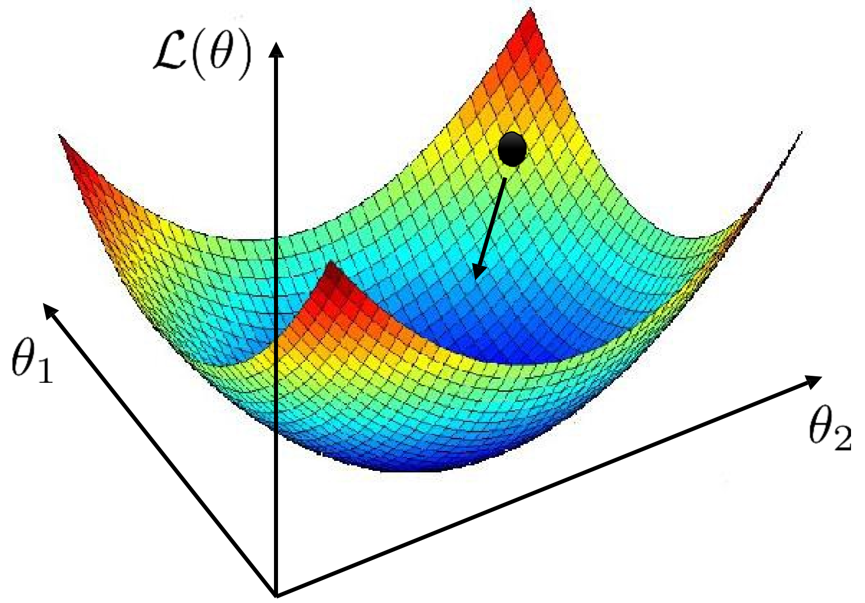
Gradient descent

An algorithm:

1. Find a *direction* v where $\mathcal{L}(\theta)$ decreases
2. $\theta \leftarrow \theta + \alpha v$

Gradient descent:

1. Compute $\nabla_\theta \mathcal{L}(\theta)$
2. $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$



$$\nabla_\theta \mathcal{L}(\theta) = \begin{pmatrix} \frac{d\mathcal{L}(\theta)}{d\theta_1} \\ \frac{d\mathcal{L}(\theta)}{d\theta_2} \\ \vdots \\ \frac{d\mathcal{L}(\theta)}{d\theta_n} \end{pmatrix}$$

Classification problems

Linear Models for Classification

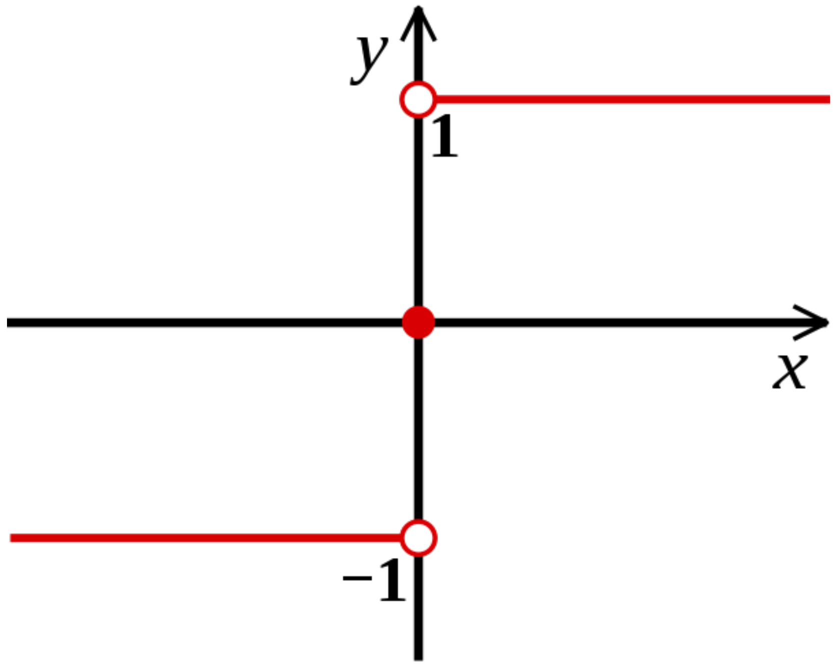
Logistic Regression

Binary classification

- Binary classification:
  - Training datapoints have one of two labels (0/1) or (-1/+1).
- Multiclass classification
  - Training datapoints have one of k labels (0, 1, 2,…, k-1)

Binary Classification

We want the possible outputs of $\boldsymbol{h_\theta}(\boldsymbol{x}) = \boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{x}$ to have values 0 or 1.



What is derivative of $f(x) = \text{sign}(x)$ in the interval $(0,\infty]$?

Logistic regression: output is a probability

Linear Regression

$$\begin{tikzcd} \mathbf{x} \arrow[r] & \boxed{\text{Model}} \arrow[r] & \mathbf{w}\cdot\mathbf{x} \end{tikzcd}$$

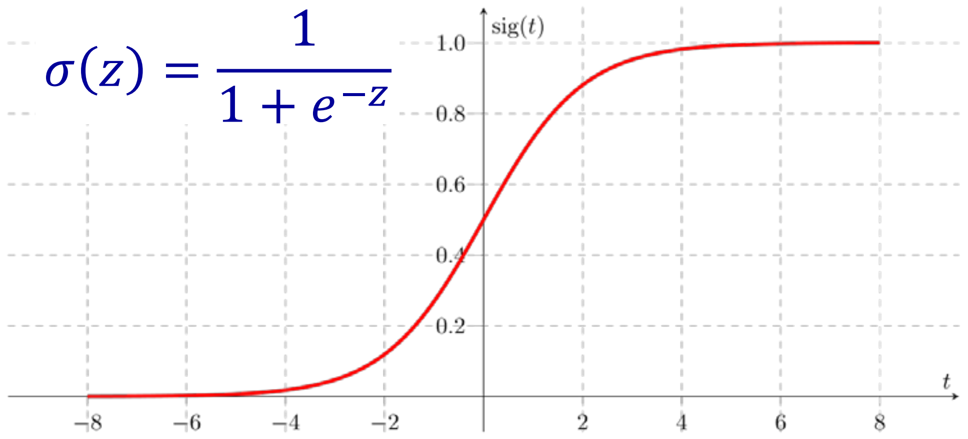
Logistic Model: Output is a probability

$$\begin{tikzcd} \mathbf{x} \arrow[r] & \boxed{\text{Model}} \arrow[r] & = \boldsymbol{Pr}[\text{label for } \mathbf{x} \text{ is } 1] \end{tikzcd}$$

Logistic Model: Output is a probability

$$\begin{tikzcd} \mathbf{x} \arrow[r] & \boxed{\text{Model}} \arrow[r] & = \boldsymbol{\sigma}(\mathbf{w}\cdot\mathbf{x}) \end{tikzcd}$$

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

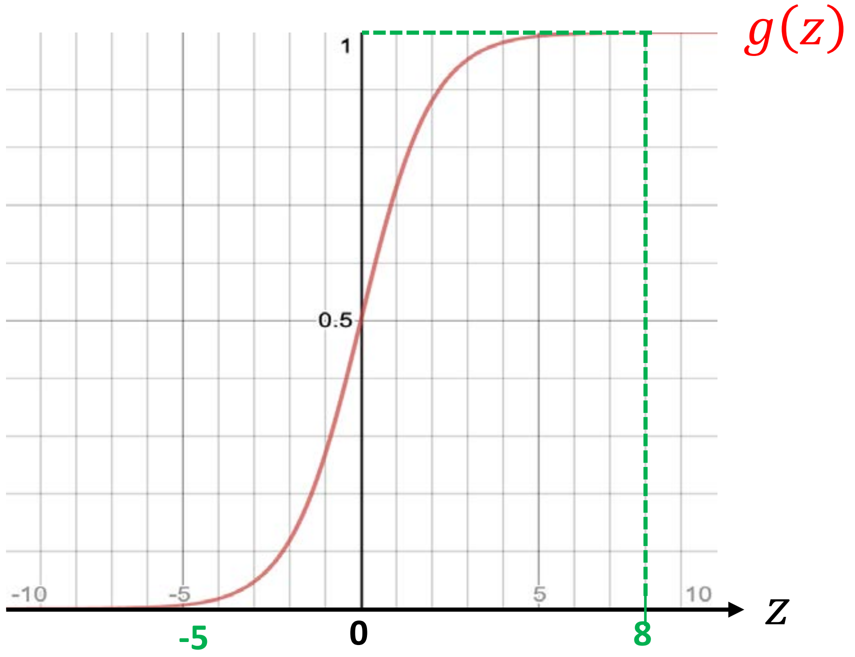


Logistic Loss

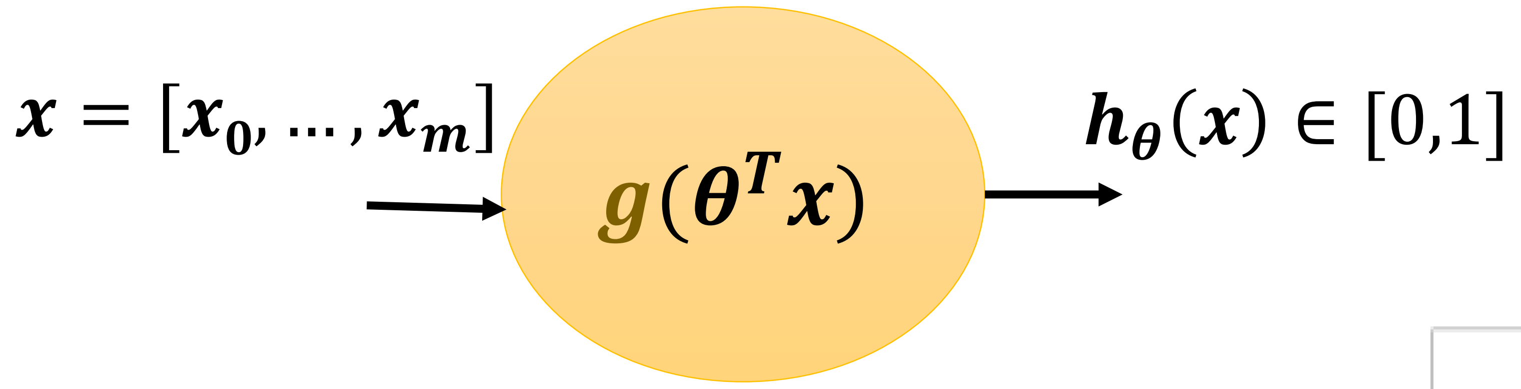
Use an ***activation function*** (e.g., ***sigmoid or logistic function***)

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

$z \in \mathbb{R}, but$
$g(z) \in [0,1]$



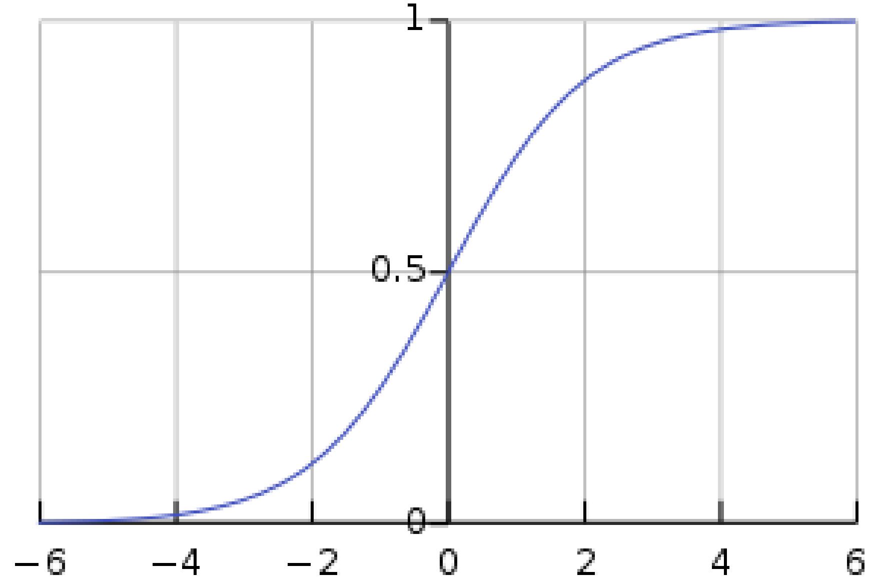
Classification



$\boldsymbol{x} = [\boldsymbol{x}_0, \dots, \boldsymbol{x}_m]$
$\boldsymbol{g(\theta^T x)}$
$\boldsymbol{h_\theta(x)} \in [0,1]$

$$h_\theta(x) = g(\theta^\top x)$$

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



Thresholding:

predict "y = 1" if $h_\theta(x) \geq 0.5$

$$\boldsymbol{z = \theta^\top x \geq 0}$$

predict "y = 0" if $h_\theta(x) < 0.5$

$$\boldsymbol{z = \theta^\top x < 0}$$

Alternative Interpretation: $h_\theta(x) =$ estimated probability that $y = 1$ on input x

Example: Probabilistic Weather Predictions

	Mon	Tue	Wed	Thu	Fri
Rained?	Y	N	Y	N	N
M1 Pr[rain]	60%	20%	90%	50%	40%
M2 Pr[rain]	70%	50%	80%	20%	60%

Which model seems to be a better predictor to you?

	Mon	Tue	Wed	Thu	Fri
Rained?	Y	N	Y	N	N
M1 Pr[rain]	60%	20%	90%	50%	40%
M2 Pr[rain]	70%	50%	80%	20%	60%

Max Likelihood Principle: The **better** model is one that assigned **more probability** to the events that actually happened (i.e., YNYNN)

Pr[*YNYNN*] = Pr[*Y* on M] · Pr[*N* on Tu] · Pr[*Y* on W] · Pr[*N* on Th] · Pr[*N* on F]

Pr[*YNYNN*] according to Model 1 =(0.6)(1 − 0.2)(0.9)(1 − 0.5)(1 − 0.4)

Pr[*YNYNN*] according to Model 2 =

Formulating loss function for logistic regression

- $\Pr[x = 1] = \sigma(z) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$
- $\Pr[x = 0] = 1 - \sigma(z) = 1 - \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}} = \frac{e^{-\mathbf{w}\cdot\mathbf{x}}}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$

$$= \frac{\frac{1}{e^{\mathbf{w}\cdot\mathbf{x}}}}{1 + \frac{1}{e^{\mathbf{w}\cdot\mathbf{x}}}} = \frac{1}{e^{\mathbf{w}\cdot\mathbf{x}}} \cdot \frac{e^{\mathbf{w}\cdot\mathbf{x}}}{1 + e^{\mathbf{w}\cdot\mathbf{x}}} = \frac{1}{1 + e^{\mathbf{w}\cdot\mathbf{x}}}$$

- Max Likelihood Fit: Find **w** that maximizes probability assigned to data.
- Probability assigned by model to the dataset

$$= \prod_i \frac{1}{1 + \exp(-y_i w \cdot x_i)}$$

Optimization for logistic loss

Max Likelihood Fit: Find $\mathbf{w}$ that maximizes probability assigned to data.

Training Dataset: $\{(x_i, y_i)\}$

Probability assigned by model to the dataset

$$= \prod_i \frac{1}{1 + \exp(-y_i \mathbf{w} \cdot x_i)}$$

- Maximizing this is equivalent to Minimizing the reciprocal
- Training Objective:

$$\min_{\mathbf{w}} \prod_{\boldsymbol{i}} \big(1 + \exp(-y_i \mathbf{w} \cdot x_i)\big)$$

$$\min_{\mathbf{w}} \sum_i \log\big(1 + \exp(-y_i \mathbf{w} \cdot x_i)\big)$$

Cost function for Logistic Regression

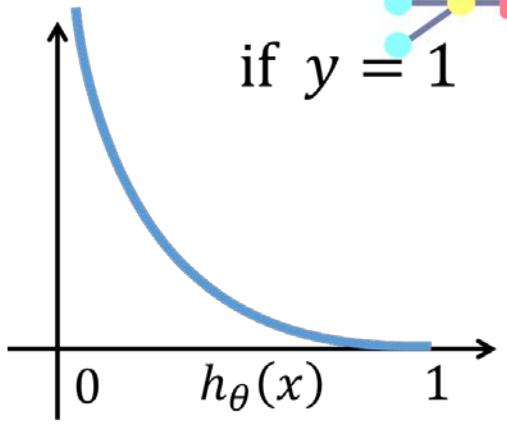
Logistic Regression

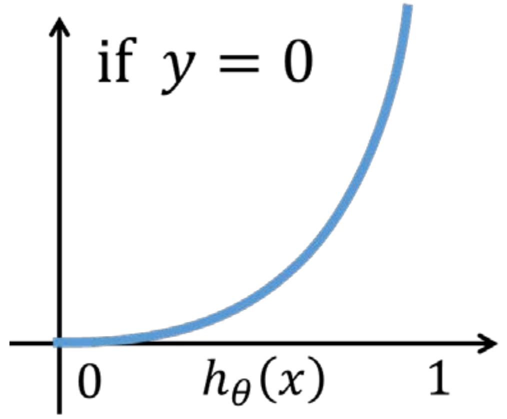
$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

$$= -\boldsymbol{y}\ \mathbf{log}\big(\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x})\big) - (\mathbf{1} - \mathbf{y})\ \mathbf{log}\big(\mathbf{1} - \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{x})\big)$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\ \log\left(h_\theta\left(x^{(i)}\right)\right) + (1 - y^{(i)})\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right]$$





Gradient descent

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right]$$

Goal: $\min_{\theta} J(\theta)$

Good news: Convex function!

Bad news: No analytical solution

Gradient descent

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\ \log\left(h_\theta\left(x^{(i)}\right)\right) + (1 - y^{(i)}) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right]$$

$$\boxed{\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}}$$

Gradient descent

(Simultaneously update all θ_j)

Repeat {

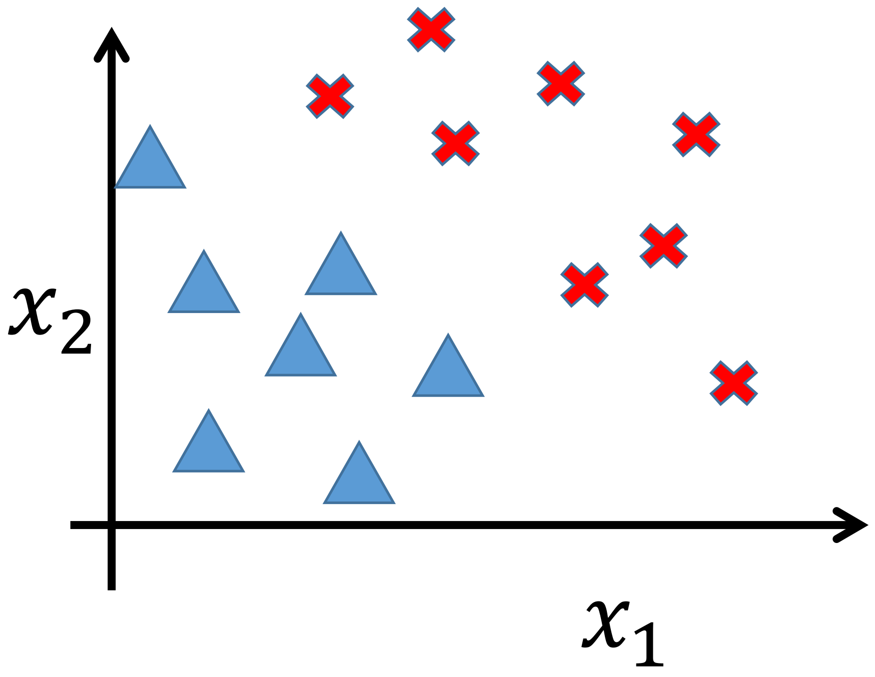
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

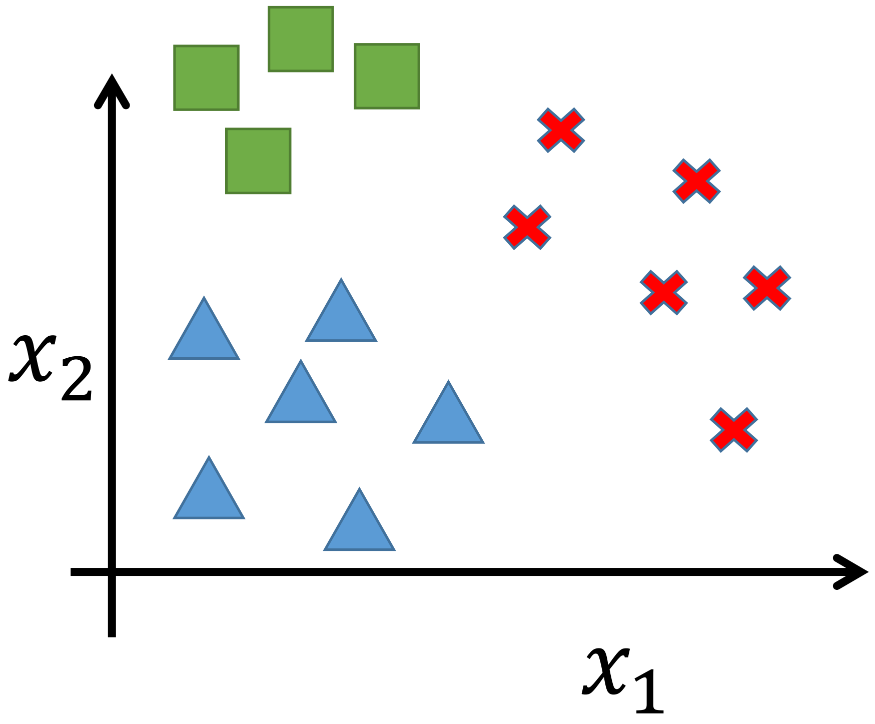
$$\boxed{\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}}$$

Multiclass classification

Binary classification



Multiclass classification



Multi-class Classification

- Multi-class Classification: y can take on K different values $\{1,2,\dots,k\}$
- $h_\theta(x)$ estimates the probability of belonging to each class

$$P(y=k|x,\theta) \propto \exp\left(\theta_k^T x\right)$$

$$\theta = \begin{bmatrix} \vdots & \vdots & \vdots \\ \theta_1 & \theta_2 & \theta_k \\ \vdots & \vdots & \vdots \end{bmatrix}$$

$$P(y=k|x,\theta) = \frac{\exp\left(\theta_k^T x\right)}{\sum_{j=1}^{K} \exp\left(\theta_j^T x\right)}$$

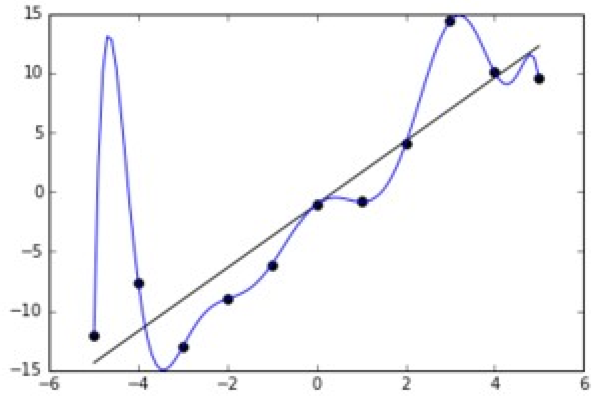
$$\boxed{\mathrm{J}(\theta) = -\left[\sum_{i=1}^{m}\sum_{j=1}^{K} 1\{y^{(i)}=k\}\log\frac{\exp(\theta_k^T x^{(i)})}{\sum_{j=1}^{K}\exp\left(\theta_j^T x^{(i)}\right)}\right]}$$

Empirical risk minimization

$$\text{Empirical risk} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(x_i, y_i, \theta) \approx E_{x\sim p(x), y\sim p(y|x)}[\mathcal{L}(x, y, \theta)]$$

Supervised learning is (usually) *empirical* risk minimization

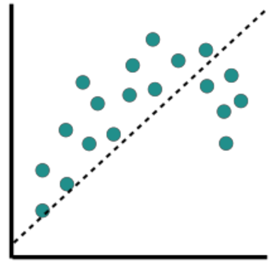
Is this the same as *true* risk minimization?



Overfitting: when the empirical risk is low, but the true risk is high

can happen if the dataset is too small

can happen if the model is too powerful (has too many parameters/capacity)



Underfitting: when the empirical risk is high, and the true risk is high

can happen if the model is too weak (has too few parameters/capacity)

can happen if your optimizer is not configured well (e.g., wrong learning rate)

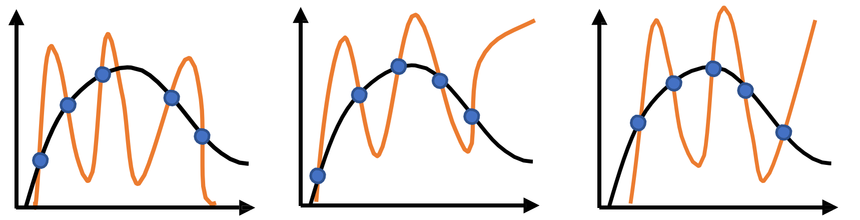
Overfitting and Underfitting

$$\mathcal{L}(\theta, x, y) = -\frac{1}{2}||f_\theta(x) - y||^2$$

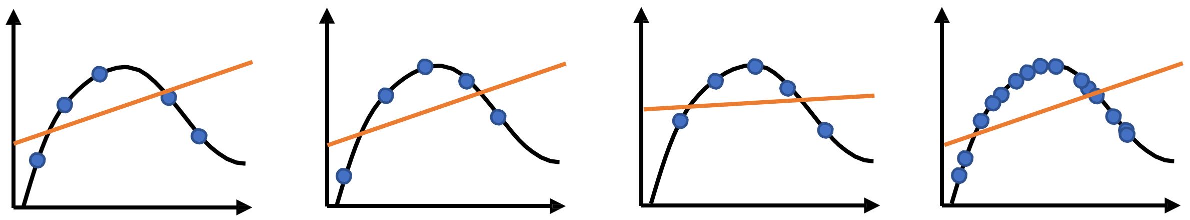
\

Question: how does the error change for different **training sets**?

overfitting



underfitting



- The training data is fitted well
- The true function is fitted poorly
- **The learned function looks different each time!**

- The training data is fitted poorly
- The true function is fitted poorly
- **The learned function looks similar, even if we pool together all the datasets!**