Introduction to Data Mining

CS 145

Lecture 2:

Linear Regression &

Backpropagation

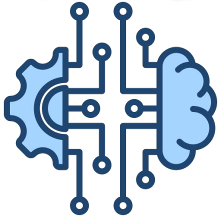
Demo
Colab

Supervised Learning

Data: X

Target Signal: Y

()



Logistic Regression
Decision Tree,
Deep Neural Nets (CNN, Transformer, etc), …

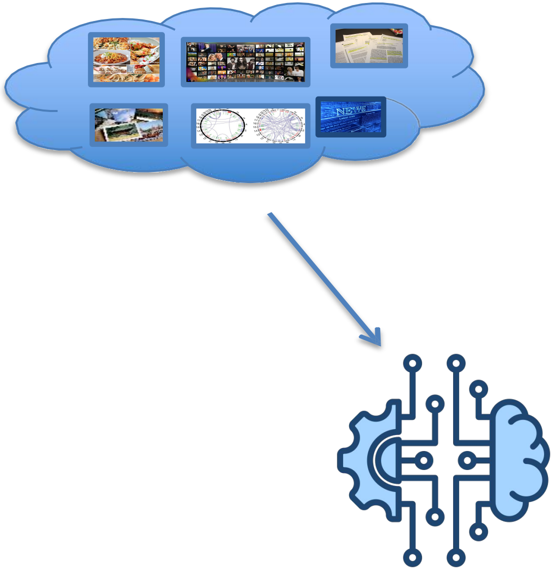
f ()

$f(x) \approx y$

For each (x,y) in X, Y

Aside: Unsupervised Learning

Data: X



No supervised target!

~~Loss Function~~ $\cancel{L\left(y_i, f(x_i \mid w,b)\right)}$

Learning objective is usually to reconstruction / compression of data (e.g., model P(x)).

Recap: Basic Supervised Learning

- Training Data: $S = \{(x_i, y_i)\}_{i=1}^{N}$ $\quad x \in R^D \quad y \in \{-1, +1\}$

  Sometimes we need a pre-defined **feature engineer** to get proper x (e.g. bag-of-word) from raw data

- Model Class: $f(x \mid w, b) = w^T x - b$ Linear Models

- Loss Function: $L\big(y_i, f(x_i \mid w, b)\big)$ Squared Loss

- Learning Objective: $\underset{w,b}{\operatorname{argmin}} \sum_{i=1}^{N} L\big(y_i, f(x_i \mid w, b)\big)$

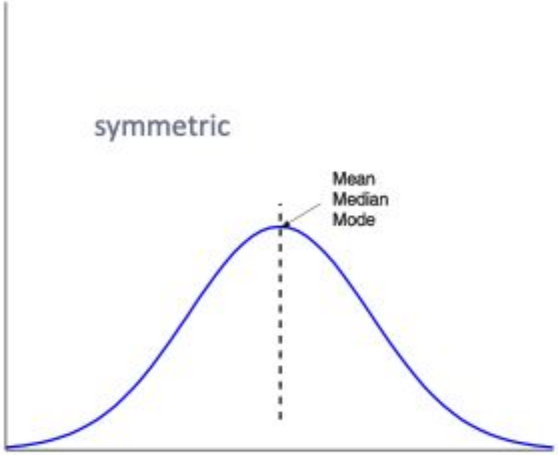
  Optimization Problem

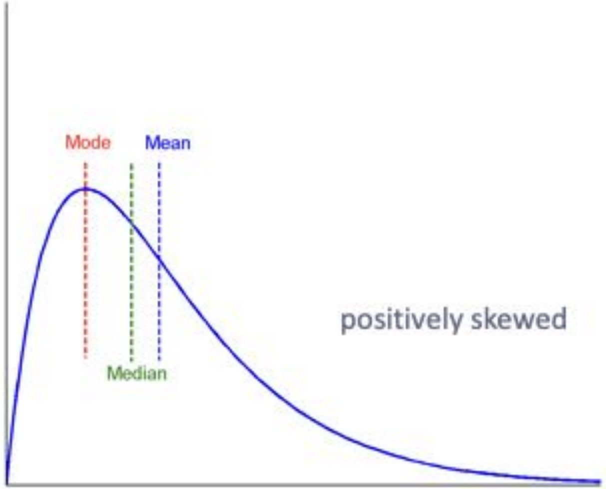
Vector Data: Prediction

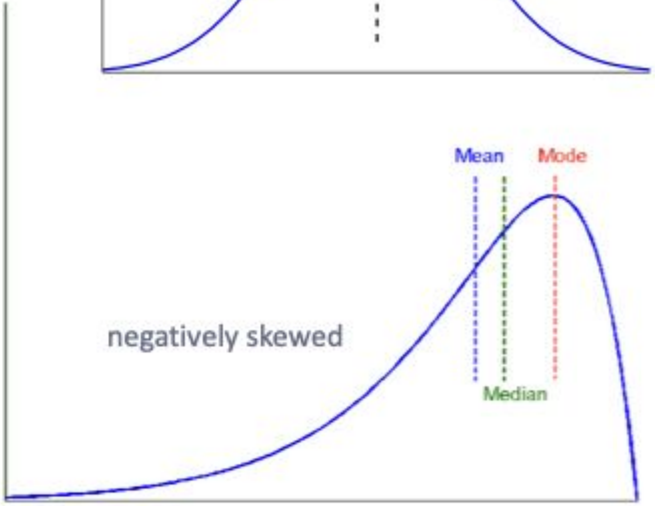
- Vector Data
- Linear Regression Model
- Model Evaluation and Selection
- Summary

Symmetric vs. Skewed Data

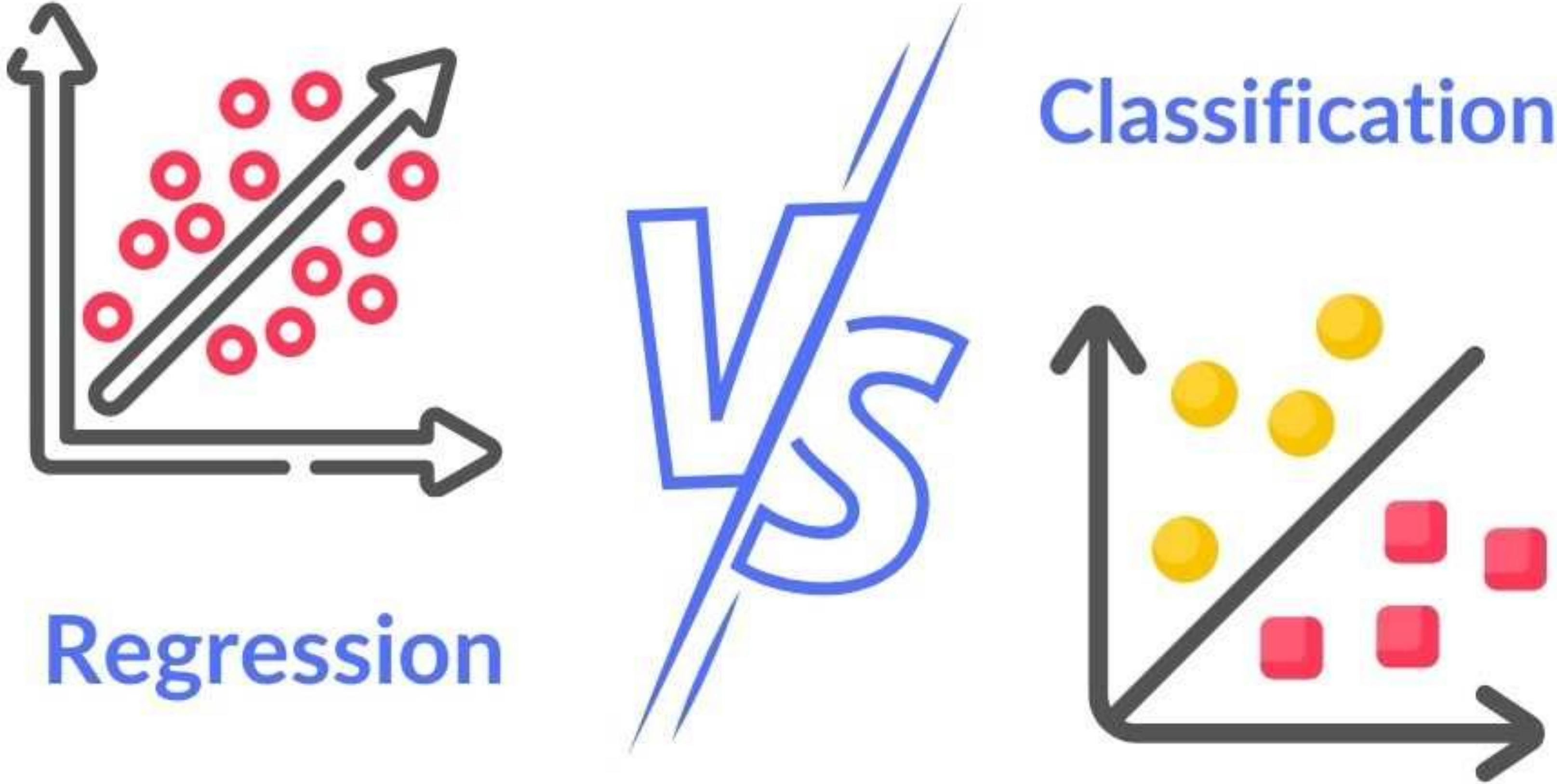
- Median, mean and mode of symmetric, positively and negatively skewed data







Demo
Colab


Regression
VS
Classification

Two Basic Supervised ML Problems

- **Classification** $f(x \mid w, b) = \text{sign}(w^T x - b)$
  - Predict which class an example belongs to
  - E.g., spam filtering example
- **Regression** $f(x \mid w, b) = w^T x - b$
  - Predict a real value or a probability
  - E.g., probability of being spam
- **Highly inter-related**
  - Train on Regression => Use for Classification

Vector Data: Prediction

- Vector Data
- Linear Regression Model
- Model Evaluation and Selection
- Summary

Example of House Price

Living Area (sqft)	# of Beds	Has pool	Price (1000$)
2104	3	Yes	400
1600	3	No	330
2400	3	No	369
1416	2	No	232
3000	4	Yes	540

$\boldsymbol{x} = (x_1, x_2, x_3)'$

y

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

Example of House Price

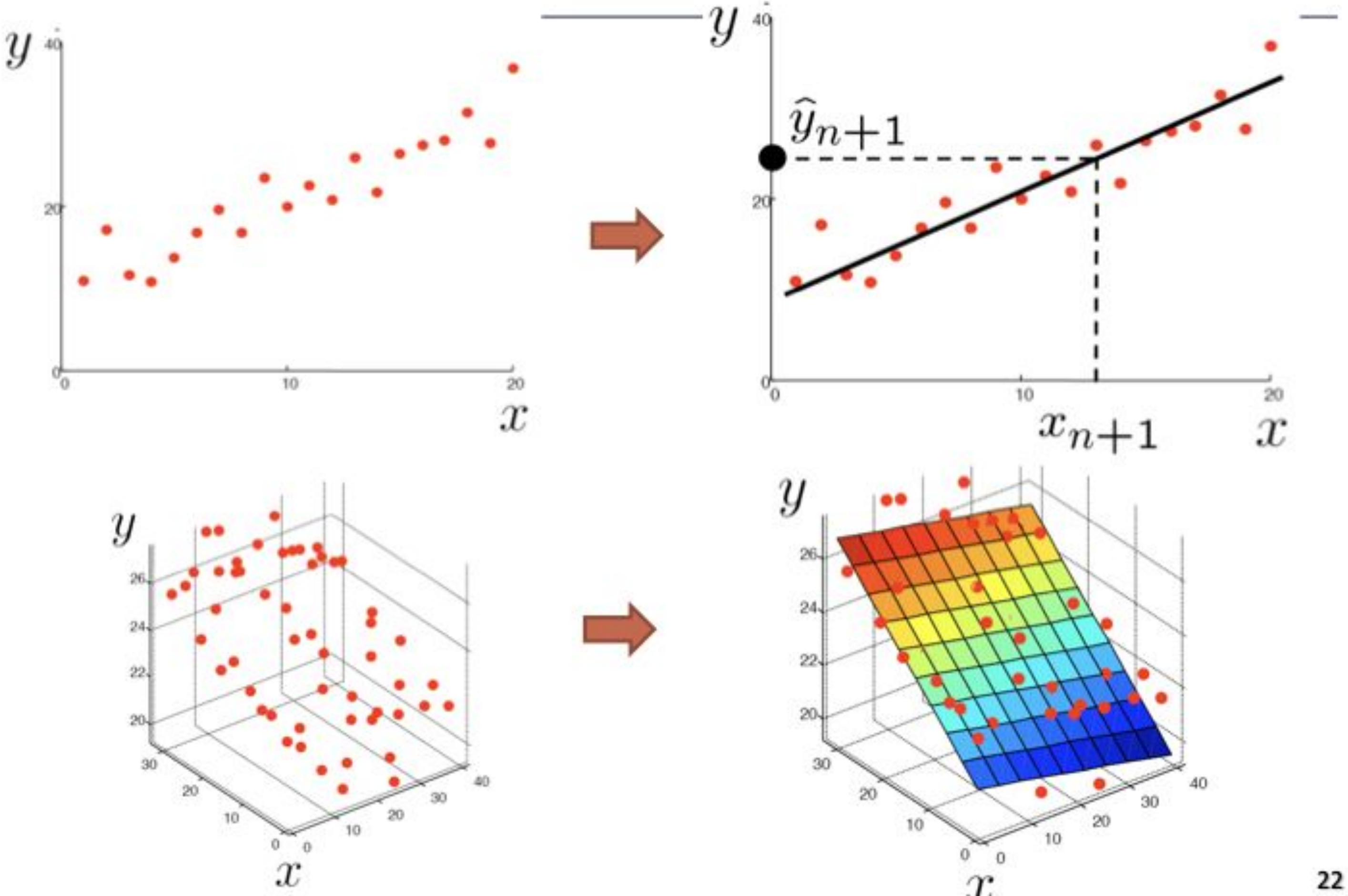
Living Area (sqft)	# of Beds	Has pool	Price (1000$)
2104	3	Yes	400
1600	3	No	330
2400	3	No	369
1416	2	No	232
3000	4	Yes	540

$\boldsymbol{x}=(x_1, x_2, x_3)'$ $\qquad$ y

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

Q: *how to handle* "Has pool" *attribute here?*

Illustration



Formalization

- Data: n independent data points $\{\boldsymbol{x}_i, y_i\}_{i=1}^{n}$
  - *y_i, dependent variable*
  - $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \dots, x_{ip})^T$, *explanatory variables*
- Model:
  - For any data point $(\boldsymbol{x}, y)$
    - *Shared weight vector*: $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)^T$
    - *Predicted outcome*: $y = \boldsymbol{x}^T\boldsymbol{\beta} + \beta_0 = \beta_0 + x_1\beta_1 + x_2\beta_2 + \cdots + x_p\beta_p$
  - For convenience, include bias term β_0 into $\boldsymbol{\beta}$
    - $\boldsymbol{x} = (1, x_1, x_2, \dots, x_p)^T$ (a column vector!)
    - $\boldsymbol{\beta} = (\beta_0, \beta_1, \dots, \beta_p)^T$
    - $y = \boldsymbol{x}^T\boldsymbol{\beta}$

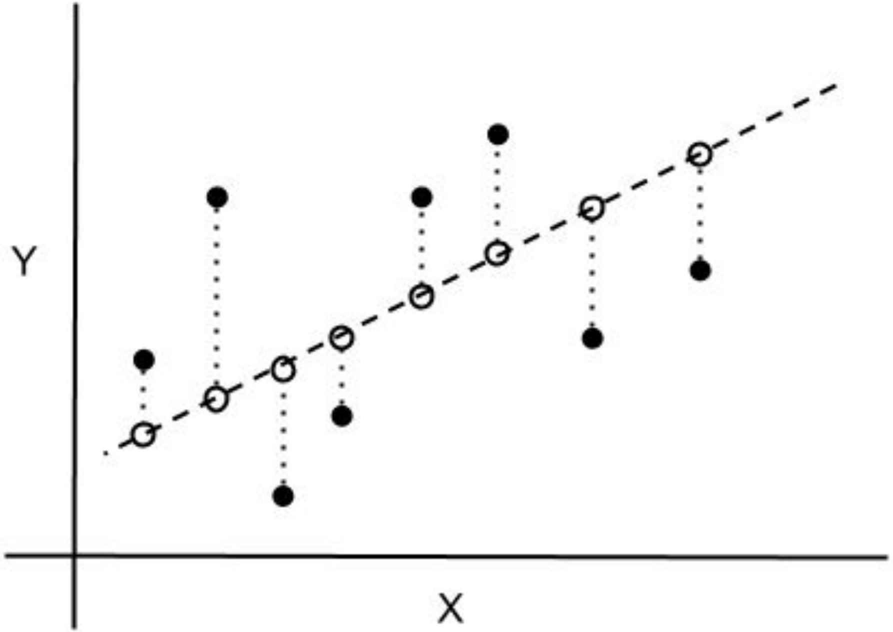
A 3-step Process

- Model Construction
  - Use training data to find the best parameter $\boldsymbol{\beta}$, denoted as $\widehat{\boldsymbol{\beta}}$
- Model Selection
  - Use validation data to select the best model
    - E.g., Feature selection
- Model Usage
  - Apply the model to the unseen data (test data): $\hat{y}_{new} = \boldsymbol{x}_{new}^T \widehat{\boldsymbol{\beta}}$

Least Square Estimation

Loss function (Mean Square Error):

$$L(\boldsymbol{\beta}) = \frac{1}{2}\sum_i \left(\boldsymbol{x}_i^T\boldsymbol{\beta} - y_i\right)^2 / n$$



Least Square Estimation

- Cost function (Mean Square Error):

$$\mathrm{L}(\boldsymbol{\beta}) = \frac{1}{2}\sum_i \left(\boldsymbol{x}_i^T \boldsymbol{\beta} - y_i\right)^2 / n$$

- Matrix form:

$$\mathrm{L}(\boldsymbol{\beta}) = (\mathrm{X}\boldsymbol{\beta} - \boldsymbol{y})^T (X\boldsymbol{\beta} - \boldsymbol{y})/2n$$

$$or \left|\left|\mathrm{X}\boldsymbol{\beta} - \boldsymbol{y}\right|\right|^2 /2n$$

$$\begin{bmatrix} 1, x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1, x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1, x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix} \qquad \begin{pmatrix} 1 \\ x_{i1} \\ \vdots \\ x_{in} \end{pmatrix} \qquad \begin{pmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{pmatrix}$$

$\boldsymbol{X}$: $\boldsymbol{n}\times(\boldsymbol{p}+\mathbf{1})$ **matrix** $\qquad$ $\mathbf{x_i}$: $(\boldsymbol{p}+\mathbf{1})\times\mathbf{1}$ **vector** $\qquad$ **y**: $\boldsymbol{n}\times\mathbf{1}$ **vector**

Ordinary Least Squares (OLS)

- Goal: find $\widehat{\boldsymbol{\beta}}$ that minimizes $\text{L}(\boldsymbol{\beta})$
- Set first derivative of $\text{L}(\boldsymbol{\beta})$ as 0
- $\text{L}(\boldsymbol{\beta}) = \frac{1}{2n}(X\boldsymbol{\beta} - y)^T(X\boldsymbol{\beta} - y)$

$$= \frac{1}{2n}(\boldsymbol{\beta}^T X^T X\boldsymbol{\beta} - y^T X\boldsymbol{\beta} - \boldsymbol{\beta}^T X^T y + y^T y)$$

- $\frac{\partial \text{L}}{\partial \boldsymbol{\beta}} = (X^T X\boldsymbol{\beta} - X^T y)/n = 0$
- $\Rightarrow \widehat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T y$

z	$\frac{\partial z}{\partial x}$
$\mathbf{Ax}$	$\mathbf{A}^T$
$\mathbf{x}^T\mathbf{A}$	$\mathbf{A}$
$\mathbf{x}^T\mathbf{x}$	$2\mathbf{x}$
$\mathbf{x}^T\mathbf{Ax}$	$\mathbf{Ax} + \mathbf{A}^T\mathbf{x}$

Other Practical Issues

- What if $X^T X$ is not invertible?
  - Add a small portion of identity matrix, λI, to it
    - ridge regression or linear regression with l2 norm regularization

$$\sum_i (y_i - \mathbf{x}_i^T \beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \quad \Longrightarrow \quad \hat{\beta} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

- What if non-linear correlation exists?
  - Transform features, say, x to x^2

Ordinary Least Squares (OLS)

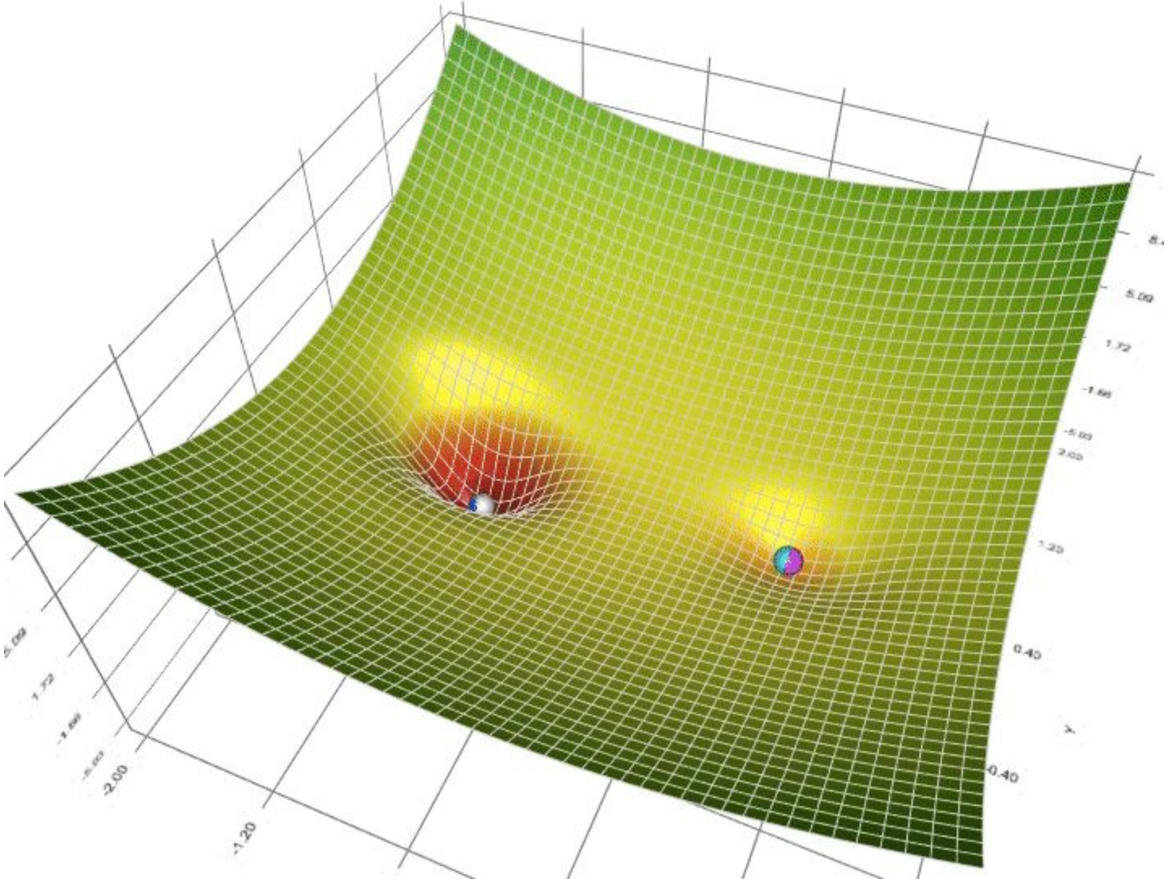
- Goal: find $\widehat{\boldsymbol{\beta}}$ that minimizes $\mathrm{L}(\boldsymbol{\beta})$
  - $\mathrm{L}(\boldsymbol{\beta}) = \frac{1}{2n}(\mathrm{X}\boldsymbol{\beta} - y)^T(X\boldsymbol{\beta} - y)$
    $= \frac{1}{2n}(\boldsymbol{\beta}^T X^T X\boldsymbol{\beta} - y^T X\boldsymbol{\beta} - \boldsymbol{\beta}^T X^T y + y^T y)$
- Ordinary least squares
  - Set first derivative of $J(\boldsymbol{\beta})$ as 0
  - $\frac{\partial \mathrm{L}}{\partial \boldsymbol{\beta}} = (X^T X\boldsymbol{\beta} - X^T y)/n = 0$
  - $\Rightarrow \widehat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T y$

z	$\frac{\partial z}{\partial x}$
$\mathbf{Ax}$	$\mathbf{A}^T$
$\mathbf{x}^T\mathbf{A}$	$\mathbf{A}$
$\mathbf{x}^T\mathbf{x}$	$2\mathbf{x}$
$\mathbf{x}^T\mathbf{Ax}$	$\mathbf{Ax} + \mathbf{A}^T\mathbf{x}$

More about matrix calculus:
https://atmos.washington.edu/~dennis/MatrixCalculus.pdf

Q: What if $(X^T X)$ is not invertible?

(Stochastic) Gradient Descent



Back to Optimizing Objective Functions

- Training Data: $S = \{(x_i, y_i)\}_{i=1}^{N}$ $\quad x \in R^D$ $\quad y \in \{-1,+1\}$

- Model Class: $f(x \mid w,b) = w^T x - b$ **Linear Models**

- Loss Function: $L(a,b) = (a-b)^2$ **Squared Loss**

- Learning Objective: $\underset{w,b}{\operatorname{argmin}} \sum_{i=1}^{N} L\big(y_i, f(x_i \mid w,b)\big)$

Optimization Problem

Back to Optimizing Objective Functions

$$\underset{w,b}{\operatorname{argmin}} L(w,b) \equiv \sum_{i=1}^{N} L\left(y_i, f(x_i \mid w,b)\right)$$

- Typically, requires optimization algorithm.
- Simplest: **Gradient Descent**

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta L(\theta_t)$$

Gradient Review for Squared Loss

$$\partial_w L(w,b) = \partial_w \sum_{i=1}^{N} L\left(y_i, f(x_i \mid w,b)\right)$$

$$= \sum_{i=1}^{N} \partial_w L\left(y_i, f(x_i \mid w,b)\right)$$

Linearity of Differentiation

$$= \sum_{i=1}^{N} -2(y_i - f(x_i \mid w,b))\partial_w f(x_i \mid w,b)$$

$$L(a,b) = (a-b)^2$$

Chain Rule

$$= \sum_{i=1}^{N} -2(y_i - f(x_i \mid w,b))x_i$$

$$f(x \mid w,b) = w^T x - b$$

Gradient Descent

- Initialize: $w^1 = 0$, $b^1 = 0$
- For t = 1…

$$w^{t+1} = w^t - \eta^{t+1} \partial_w L(w^t, b^t)$$

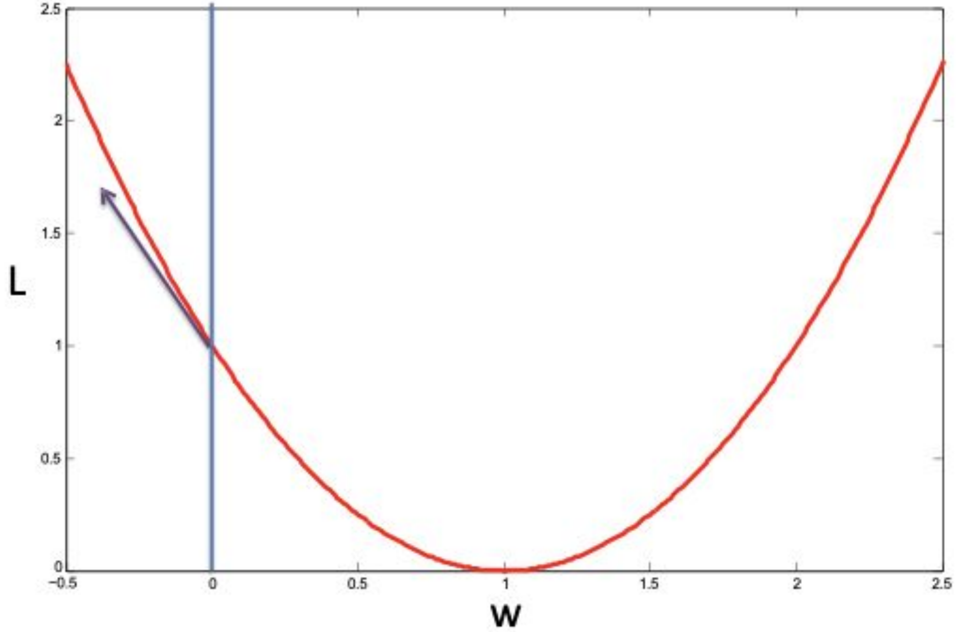
$$b^{t+1} = b^t - \eta^{t+1} \partial_b L(w^t, b^t)$$

"Step Size"

How to Choose Step Size?

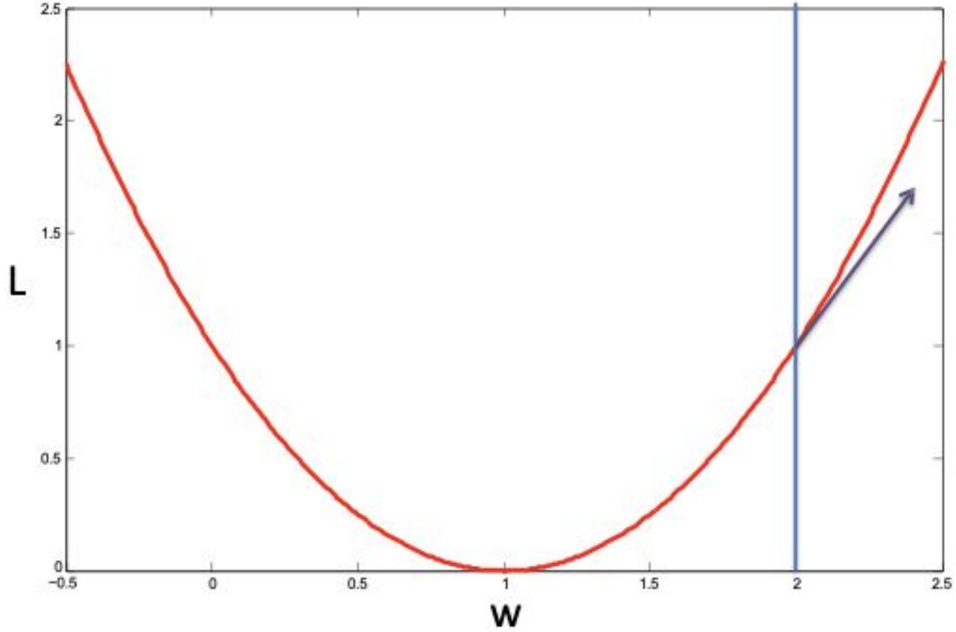
$\eta = 1$

$$\partial_w L(w) = -2(1 - w)$$



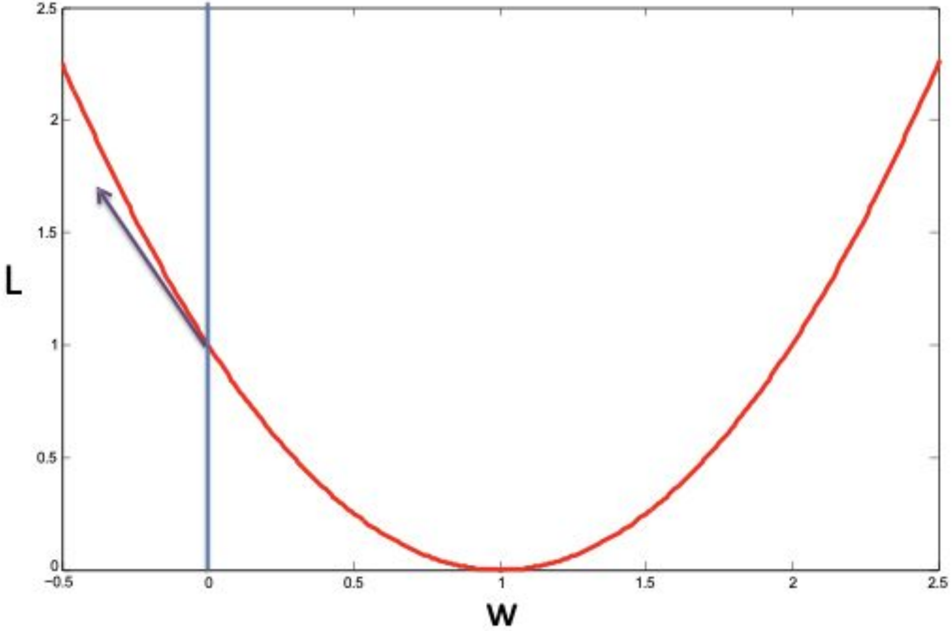
How to Choose Step Size?

$\eta = 1$ $\partial_w L(w) = -2(1 - w)$



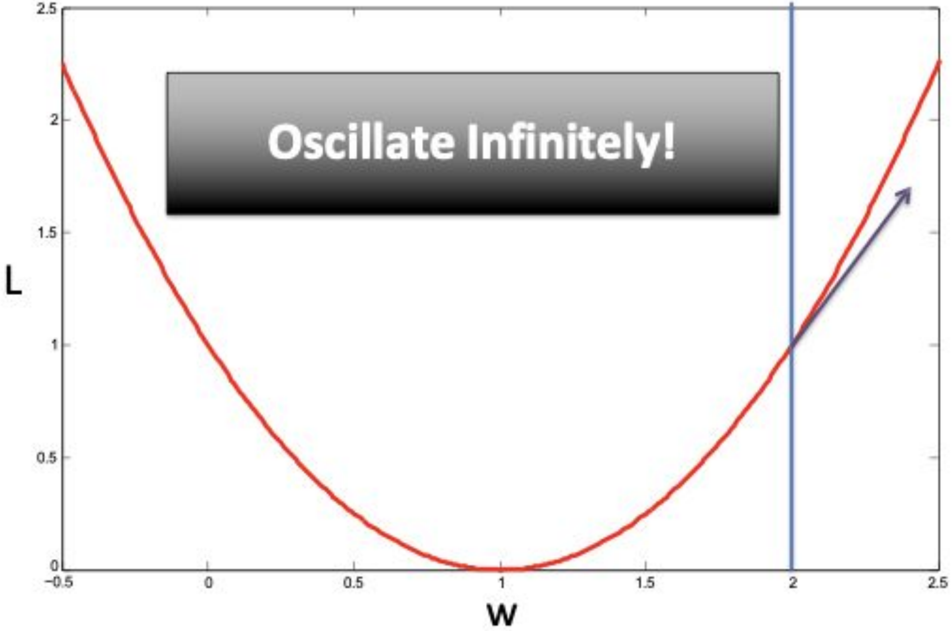
How to Choose Step Size?

$\eta = 1$ $\partial_w L(w) = -2(1 - w)$



How to Choose Step Size?

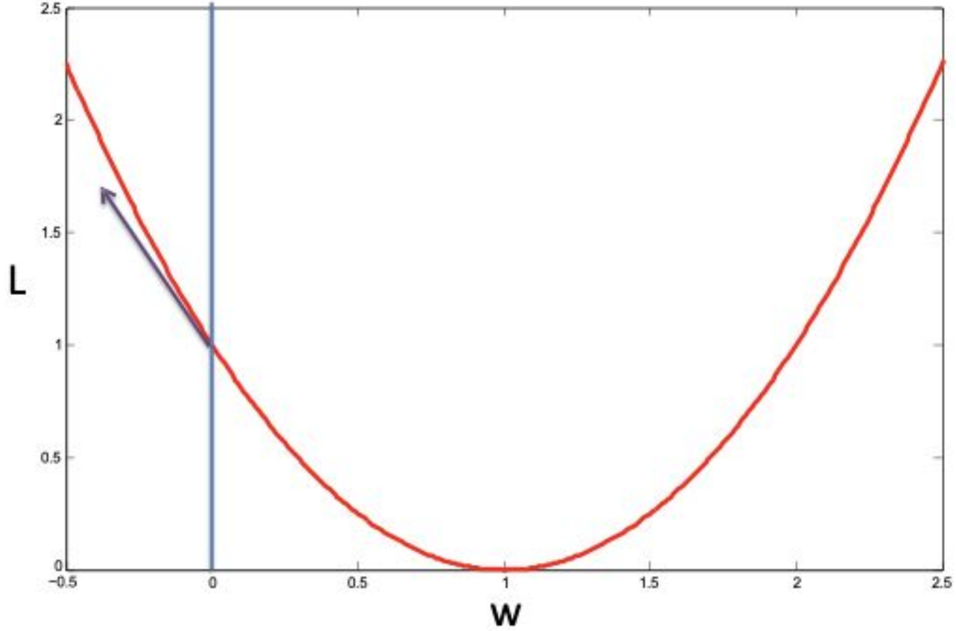
$\eta = 1$ $\qquad \partial_w L(w) = -2(1 - w)$



How to Choose Step Size?

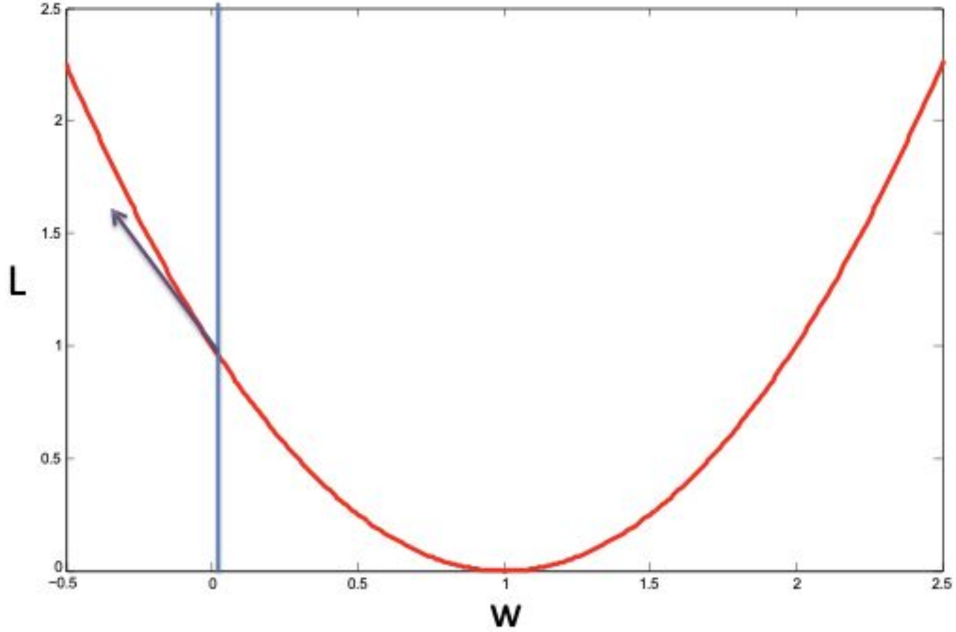
$\eta = 0.0001$

$\partial_w L(w) = -2(1-w)$



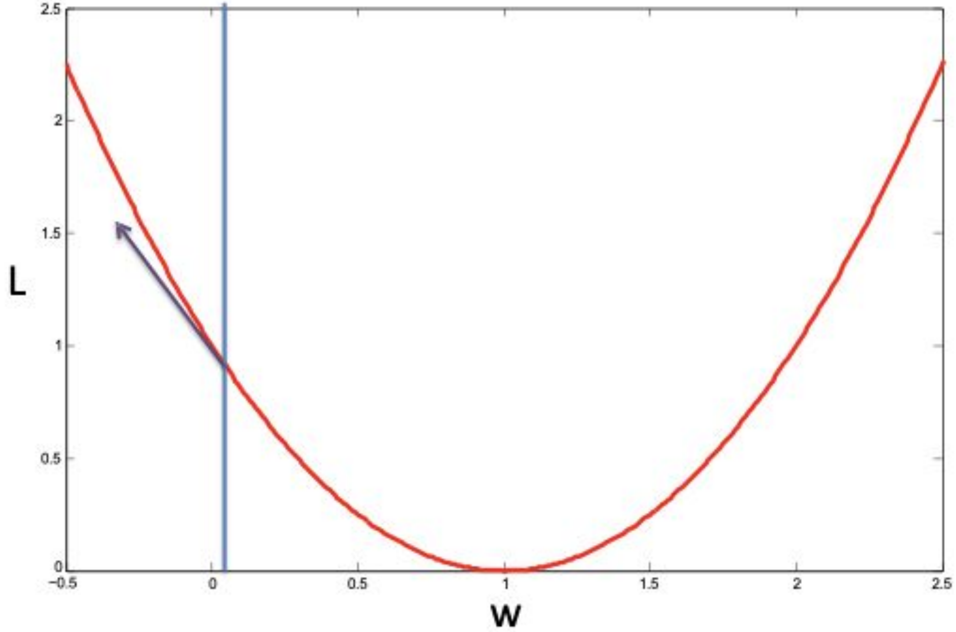
How to Choose Step Size?

$$\eta = 0.0001 \qquad \partial_w L(w) = -2(1-w)$$



How to Choose Step Size?

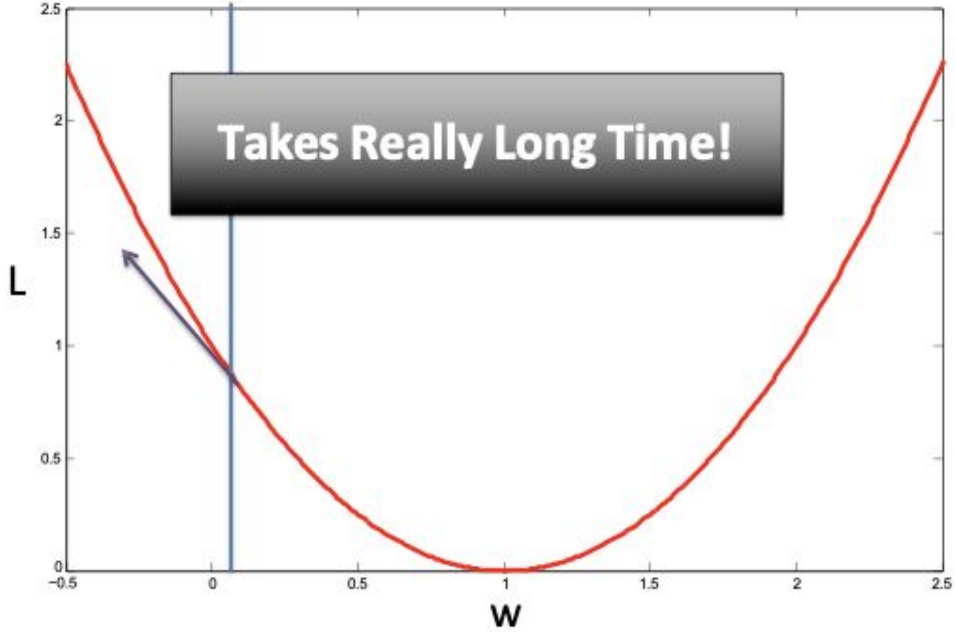
$$\eta = 0.0001 \qquad \partial_w L(w) = -2(1-w)$$



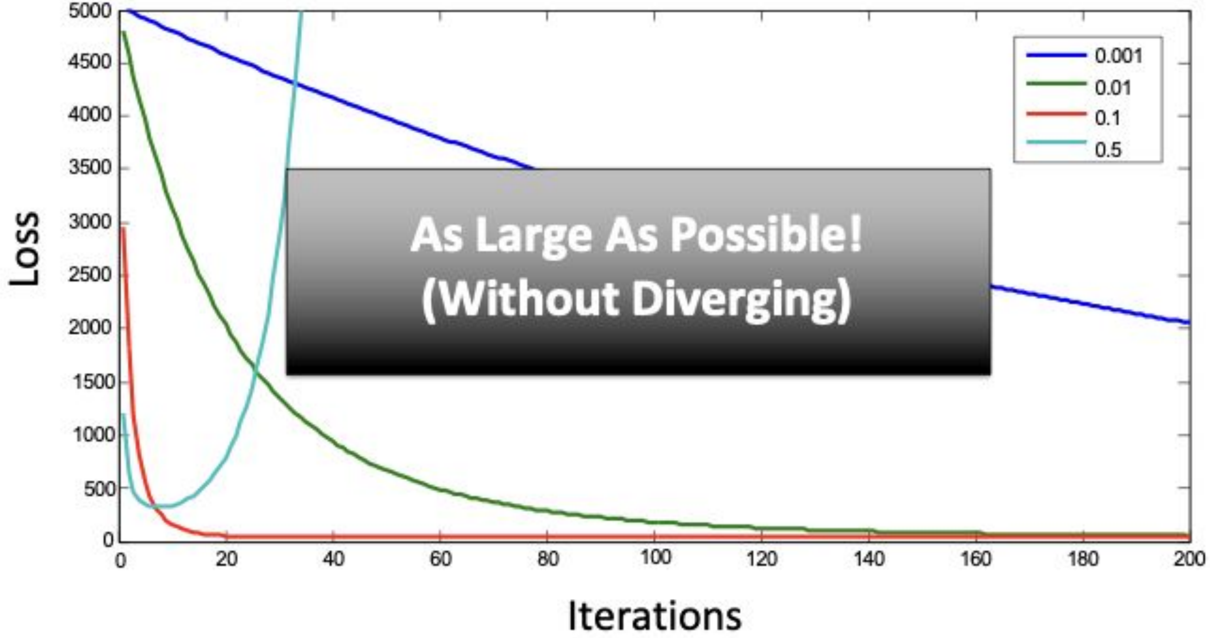
How to Choose Step Size?

$\eta = 0.0001$

$\partial_w L(w) = -2(1 - w)$



How to Choose Step Size?



Note that the absolute scale is not meaningful
Focus on the relative magnitude differences

Aside: Convexity

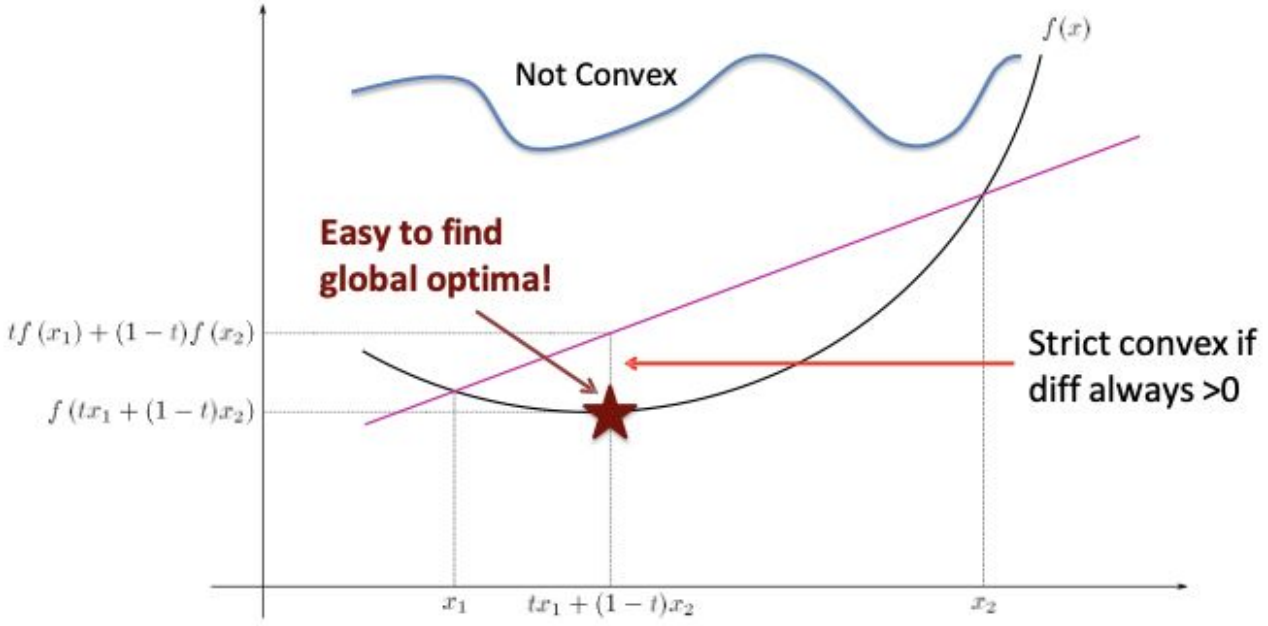
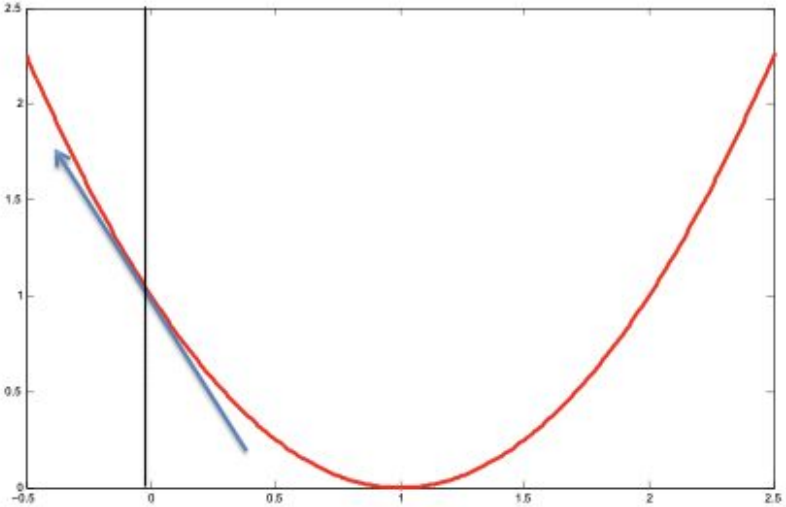


Image Source: http://en.wikipedia.org/wiki/Convex_function

Aside: Convexity

$$L(x_2) \geq L(x_1) + \nabla L(x_1)^T (x_2 - x_1)$$

Function is always above the locally linear extrapolation



Aside: Convexity

- All local optima are global optima:

Gradient Descent will find optimum

Assuming step size chosen safely

- Strictly convex: unique global optimum:

- Almost all standard objectives are (strictly) convex:
  - Squared Loss, SVMs, LR, Ridge, Lasso
  - We will see non-convex objectives later (e.g., deep learning)

Limitation of Gradient Descent

- Requires full pass over training set per iteration

$$\partial_w L(w, b \mid S) = \partial_w \sum_{i=1}^{N} L\left(y_i, f(x_i \mid w, b)\right)$$

- Very expensive if training set is huge

- **Do we need to do a full pass over the data?**

Stochastic Gradient Descent

- Suppose Loss Function Decomposes Additively

$$L(w,b) = \frac{1}{N}\sum_{i=1}^{N} L_i(w,b)$$

Each L_i corresponds to a single data point

- Gradient = expected gradient of sub-functions

$$\partial_w L(w,b) = \partial_w \operatorname{E}_i \left[L_i(w,b) \right] = \operatorname{E}_i \left[\partial_w L_i(w,b) \right]$$

$$L_i(w,b) \equiv \left(y_i - f(x_i \mid w,b\right)^2$$

Stochastic Gradient Descent

- Suffices to take random gradient update
  - So long as it matches the true gradient in expectation
- Each iteration t:
  - Choose i at random

Expected Value is: $\partial_w L(w,b)$

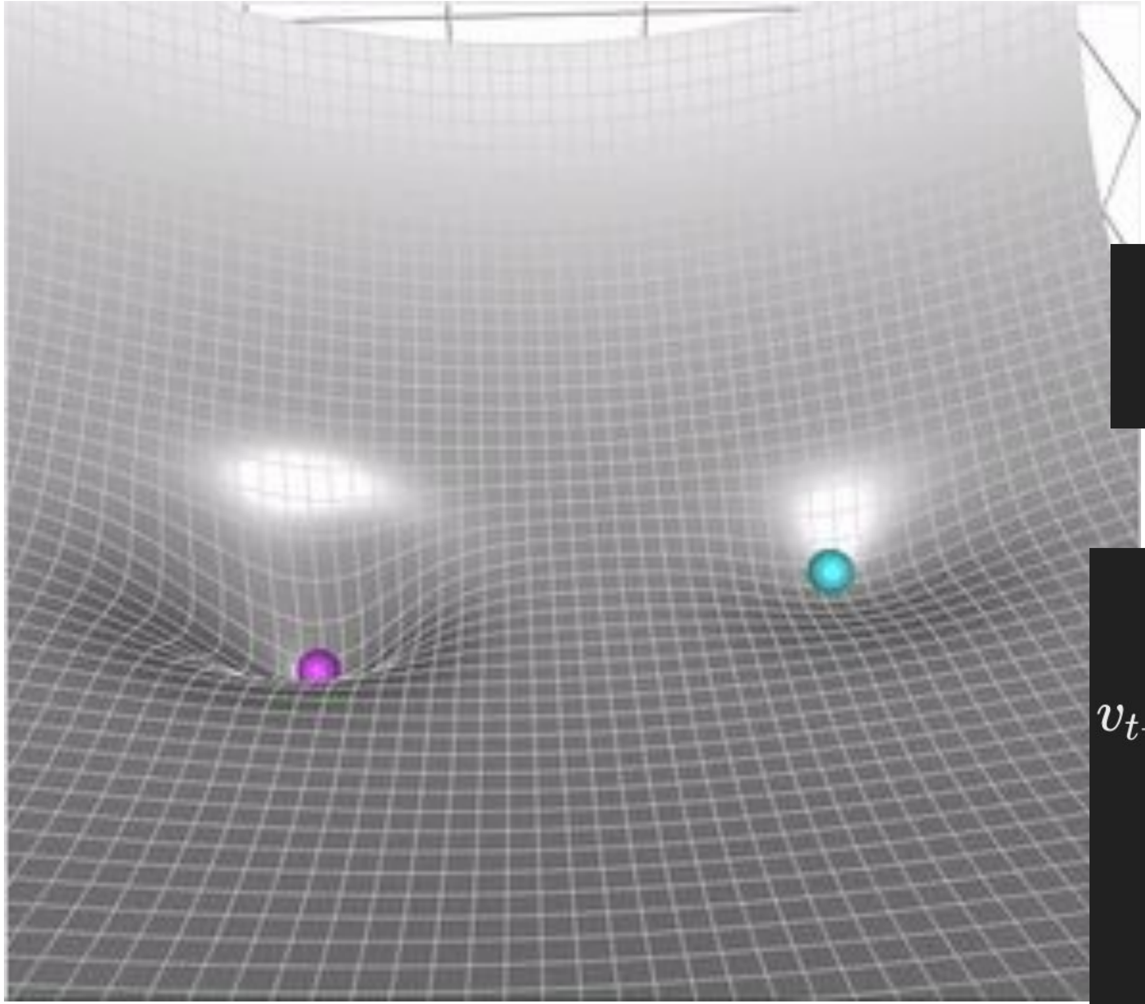
$$w^{t+1} = w^t - \eta^{t+1}\partial_w L_i(w,b)$$

$$b^{t+1} = b^t - \eta^{t+1}\partial_b L_i(w,b)$$

- **SGD is an online learning algorithm!**

Mini-Batch SGD

- Each L_i is a small batch of training examples
  - E.g,. 500-1000 examples
  - Can leverage vector operations
  - Decrease volatility of gradient updates
- Industry state-of-the-art
  - Everyone uses mini-batch SGD variants
    - Most common is Adam: https://arxiv.org/abs/1412.6980
  - Often parallelized
    - (e.g., different cores work on different mini-batches)



Standard Gradient Descent

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta L(\theta_t)$$

Gradient Descent with velocity

$$v_{t+1} = \beta \cdot v_t + (1 - \beta) \cdot \nabla_\theta L(\theta_t)$$

velocity

$$\theta_{t+1} = \theta_t - \eta \cdot v_{t+1}$$

Adam: Gradient Descent with velocity & **Momentum**

1. **Initialize two moment vectors** $m_0 = 0$ and $v_0 = 0$, and a timestep $t = 0$.
2. **Compute gradient:** At each step t, compute the gradient $g_t = \nabla_\theta L(\theta_{t-1})$ with respect to the parameters at the previous timestep.
3. **Update biased first moment estimate:**

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

1. **Update biased second raw moment estimate:**

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

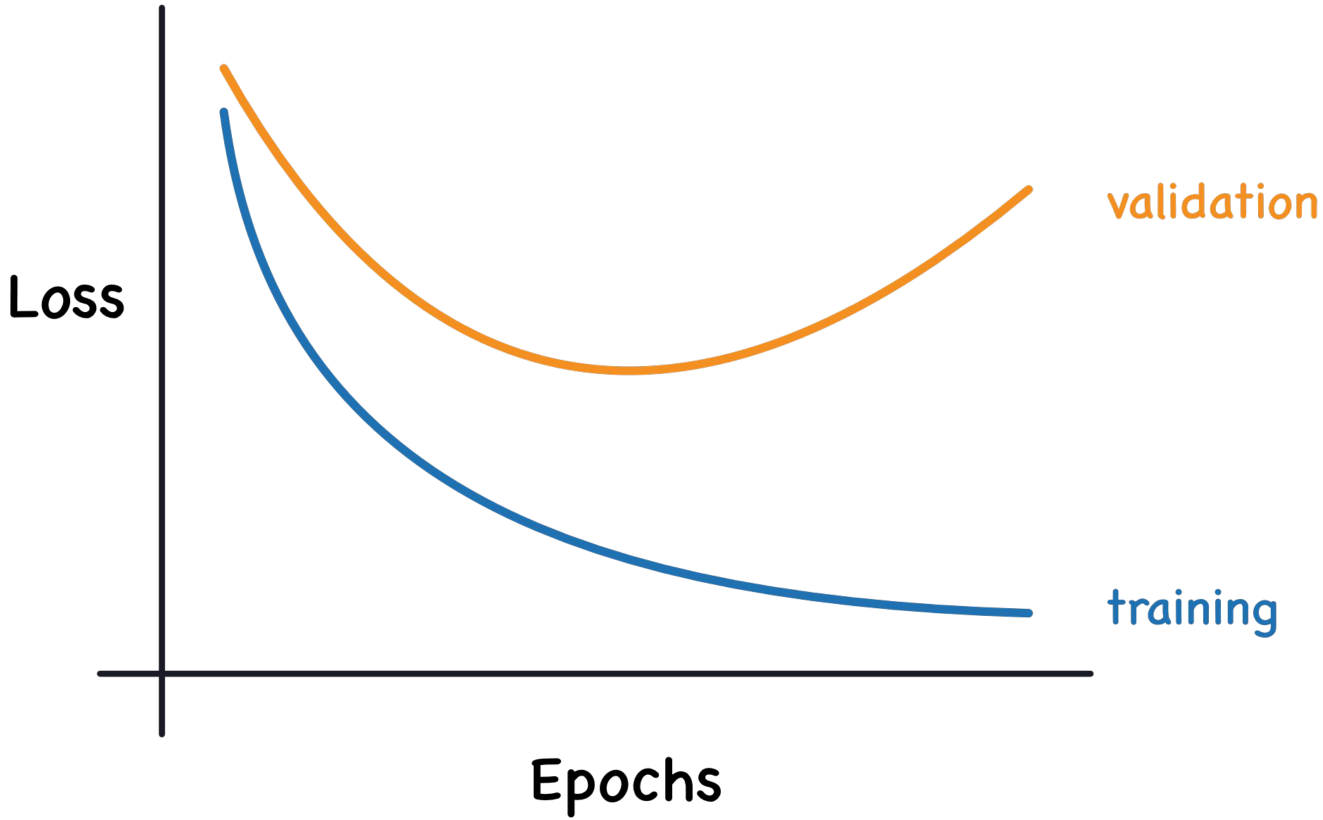
$$\theta_t = \theta_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

https://youtu.be/iIYd4TAzNoU

Checking for Convergence

- How to check for convergence?
  - Evaluating loss on entire training set seems expensive…
- Don’t check after every iteration
  - E.g., check every 1000 iterations
- Evaluate loss on a subset of training data
  - E.g., the previous 5000 examples.

The Learning Curves



Overfitting

- Very accurate model

- But crashed on live test!
- Model w only cared about staying between two green patches

Test Error

- **"True" distribution:** P(x,y)
  - Unknown to us

"All possible emails"

- **Train:** f(x) = y
  - Using training data: $S = \{(x_i, y_i)\}_{i=1}^{N}$
  - Sampled identically and independently from P(x,y)
- **Test Error:**

$$L_P(f) = E_{(x,y)\sim P(x,y)}\left[L(y, f(x))\right]$$

Prediction Loss on all possible emails

- **Overfitting:** Test Error >> Training Error

Test Error

- **Test Error:**

$$L_P(f) = E_{(x,y)\sim P(x,y)}\left[L(y, f(x))\right]$$

- **Treat f_S as random variable:** (randomness over *S*)

$$f_S = \underset{w,b}{\operatorname{argmin}} \sum_{(x_i,y_i)\in S} L\big(y_i, f(x_i \mid w, b)\big)$$

- **Expected Test Error:**

$$E_S\left[L_P(f_S)\right] = E_S\lfloor\lceil E_{(x,y)\sim P(x,y)}\left[L(y, f_S(x))\right]^{\#}\rfloor$$

Bias and Variance

True predictor $f(x)$: $x^T\boldsymbol{\beta}$

Estimated predictor $\hat{f}(x)$: $x^T\widehat{\boldsymbol{\beta}}$

- Bias: $E(\hat{f}(x)) - f(x)$
  - How far away is the expectation of the estimator to the true value? The smaller the better.
- Variance: $Var\left(\hat{f}(x)\right) = E[\left(\hat{f}(x) - E\left(\hat{f}(x)\right)\right)^2]$
  - How variant is the estimator? The smaller the better.
- Reconsider mean square error
  - $J(\widehat{\boldsymbol{\beta}}) = \sum_i \left(\boldsymbol{x}_i^T\widehat{\boldsymbol{\beta}} - y_i\right)^2/n$
  - Can be considered as
    - $E[\left(\hat{f}(x) - f(x) - \varepsilon\right)^2] = bias^2 + variance + noise$

Note $E(\varepsilon) = 0, Var(\varepsilon) = \sigma^2$

Recap: Stochastic Gradient Descent

- Conceptually:
  - Decompose Loss Function Additively
  - Choose a Component Randomly
  - Gradient Update
- Benefits:
  - Avoid iterating entire dataset for every update
  - Gradient update is consistent (in expectation)
- Industry Standard

Recap: Complete Pipeline

$$S = \left\{(x_i, y_i)\right\}_{i=1}^{N}$$

Training Data

$$f(x \mid w, b) = w^T x - b$$

Model Class(es)

$$L(a,b) = (a-b)^2$$

Loss Function

$$\underset{w,b}{\operatorname{argmin}} \sum_{i=1}^{N} L\left(y_i, f(x_i \mid w, b)\right)$$ **Use SGD!**

Cross Validation & Model Selection

Profit!

Next Two Lectures

- Classification
  - Log Loss (Logistic Regression)
- Primer on Non-linear model classes
  - Neural Nets
- Regularization, Sparsity, Lasso