

# SDM274: AI and Machine Learning

Zhiyun Lin



南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY



设计智造学院  
School of  
System Design and  
Intelligent Manufacturing

- Regression Problems
- Linear Regression

# Problems for Today

o What should I watch this Friday?



All

Movies, TV & Showtimes

Celebs, Events & Photos

News & Community

Watchlist



See More on IMDb Pro »

## The Martian (2015)

PG-13 | 144 min | Adventure, Comedy, Drama |  
2 October 2015 (USA)

**Your rating:** ★★★★★★★★ -/10  
Ratings: **8.1**/10 from 271,829 users Metascore: 80/100  
Reviews: 750 user | 499 critic | 46 from Metacritic.com

During a manned mission to Mars, Astronaut Mark Watney is presumed dead after a fierce storm and left behind by his crew. But Watney has survived and finds himself stranded and alone on the hostile planet. With only meager supplies, he must draw upon his ingenuity, wit and spirit to subsist and find a way to signal to Earth that he is alive.

**Director:** Ridley Scott  
**Writers:** Drew Goddard (screenplay), Andy Weir (book)  
**Stars:** Matt Damon, Jessica Chastain, Kristen Wiig |  
See full cast and crew »

+ Watchlist

Watch Trailer

Share...

9

# Problems for Today

o What should I watch this Friday?

All

Movies, TV & ShowtimesCelebs, Events & PhotosNews & CommunityWatchlist



See More on IMDb Pro »

## Point Break (2015)

PG-13 | 114 min | Action, Crime, Sport | 25 December 2015 (USA)

**Your rating:** ★★★★★★ 5.4 /10

Ratings: **5.4**/10 from 7,322 users Metascore: **34**/100

Reviews: 60 user | 84 critic | 19 from Metacritic.com

A young FBI agent infiltrates an extraordinary team of extreme sports athletes he suspects of masterminding a string of unprecedented, sophisticated corporate heists. "Point Break" is inspired by the classic 1991 hit.

**Director:** [Ericson Core](#)

**Writers:** [Kurt Wimmer](#) (screenplay), [Rick King](#) (story), [5 more credits](#) »

**Stars:** [Édgar Ramírez](#), [Luke Bracey](#), [Ray Winstone](#) | [See full cast and crew](#) »

+ WatchlistWatch TrailerShare...

**15**

# Problems for Today

- o **Goal:** Predict movie rating automatically!



IMDb Find Movies, TV shows, Celebrities and more... All

Movies, TV & Showtimes Celebs, Events & Photos News & Community Watchlist

**Point Break** (2015) PG-13 | 114 min | 25 December 2015

**5.4** **Our rating:** ★★★★★★★★★★ -/10  
Ratings: 5.4/10 from 7,322 users Metascore: 34/100  
Reviews: 60 user | 84 critic | 19 from Metacritic.com

A young FBI agent infiltrates an extraordinary team of extreme sports athletes he suspects of masterminding a string of unprecedented, sophisticated corporate heists. "Point Break" is inspired by the classic 1991 hit.

**Director:** Ericson Core  
**Writers:** Kurt Wimmer (screenplay), Rick King (story), 5 more credits »  
**Stars:** Édgar Ramírez, Luke Bracey, Ray Winstone | See full cast and crew »

See More on IMDb Pro »

+ Watchlist Watch Trailer Share...

**Predict this automatically!**

Factors:

- Year of release of the film in cinemas
- Length of the film in minutes
- Budget for the films production
- Number of positive votes received by viewers
- Genre of the film including Action, Animation, Comedy, Documentary, Drama, Romance and Short

# Problems for Today

- o **Goal:** How many followers will I get?

Red Leather Jacket

Updated on Jan 09, 2016



From This User

+1 282 VOTES

5 COMMENTS

67 FAVORITES

f Like 0

t Tweet

G+1 0

...

Pinterest 2

Tags

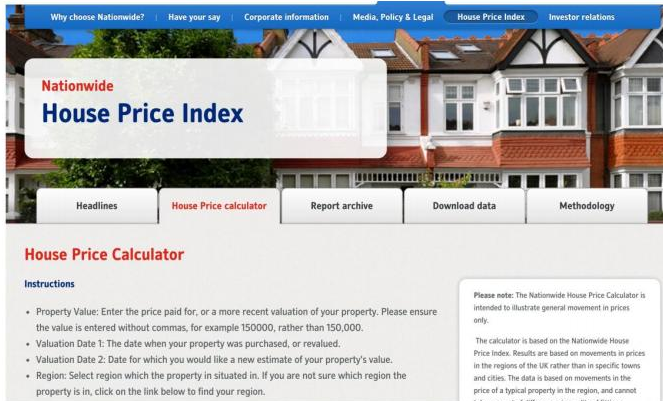
Chic  
Everyday  
Winter

SHARE



# Problems for Today

- **Goal:** Predict the price of the house



The screenshot shows the Nationwide House Price Index website. The header features a blue navigation bar with links: 'Why choose Nationwide?', 'Have your say', 'Corporate information', 'Media, Policy & Legal', 'House Price Index' (highlighted), and 'Investor relations'. Below the navigation bar is a large image of a row of houses. Overlaid on this image is a white box containing the text 'Nationwide House Price Index'. Below the image is a horizontal menu with five buttons: 'Headlines', 'House Price calculator' (highlighted), 'Report archive', 'Download data', and 'Methodology'. Below the menu is the 'House Price Calculator' section, which includes 'Instructions' and a 'Please note' box.

**Nationwide**  
**House Price Index**

Headlines House Price calculator Report archive Download data Methodology

### House Price Calculator

**Instructions**

- Property Value: Enter the price paid for, or a more recent valuation of your property. Please ensure the value is entered without commas, for example 150000, rather than 150,000.
- Valuation Date 1: The date when your property was purchased, or revalued.
- Valuation Date 2: Date for which you would like a new estimate of your property's value.
- Region: Select region which the property is situated in. If you are not sure which region the property is in, click on the link below to find your region.

**Please note:** The Nationwide House Price Calculator is intended to illustrate general movement in prices only.

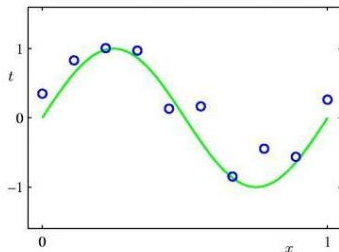
The calculator is based on the Nationwide House Price Index. Results are based on movements in prices in the regions of the UK rather than in specific towns and cities. The data is based on movements in the price of a typical property in the region, and cannot show movement of individual properties.

# Regression

- ❑ What do all these problems have in common?
  - Continuous **outputs**, we'll call these  $t$   
  
(e.g., a rating: a real number between 0-10, # of followers, house price)
- ❑ Predicting continuous outputs is called **regression**
- ❑ What do I need in order to **predict** these outputs?
  - **Features** (inputs), we'll call these  $x$  (or  $\mathbf{x}$  if vectors)
  - **Training examples**, many  $x(i)$  for which  $t(i)$  is known (e.g., many movies for which we know the rating)
  - A **model**, a function that represents the relationship between  $x$  and  $t$
  - A **loss** or a **cost** or an **objective** function, which tells us how well our model approximates the training examples
  - **Optimization**, a way of finding the parameters of our model that minimizes the loss function



# Simple 1-D regression



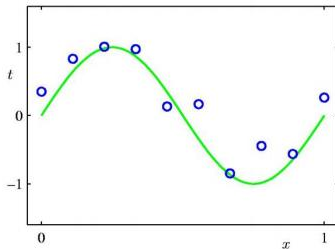
- Circles are data points (i.e., training examples) that are given to us
- The data points are uniform in  $x$ , but may be displaced in

$$t(x) = f(x) + \epsilon$$

with  $\epsilon$  some noise

- In **green** is the "true" curve that we don't know
- **Goal**: We want to fit a curve to these points

# Simple 1-D regression

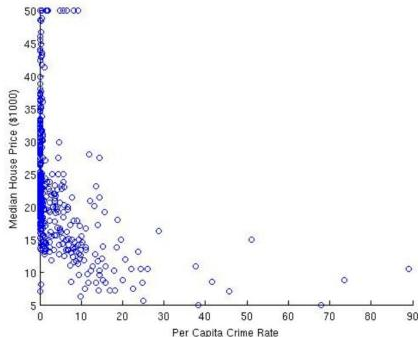


- Key Questions:

- ▶ How do we parametrize the **model**?
- ▶ What **loss (objective) function** should we use to judge the fit?
- ▶ How do we optimize fit to unseen test data (**generalization**)?

## Example: Boston Housing data

- Estimate median house price in a neighborhood based on neighborhood statistics
- Look at first possible attribute (feature): per capita crime rate



- Use this to predict house prices in other neighborhoods
- Is this a **good input (attribute) to predict** house prices?

# Represent the Data

- Data is described as pairs  $\mathcal{D} = \{(x^{(1)}, t^{(1)}), \dots, (x^{(N)}, t^{(N)})\}$ 
  - ▶  $x \in \mathbb{R}$  is the **input feature** (per capita crime rate)
  - ▶  $t \in \mathbb{R}$  is the **target output** (median house price)
  - ▶ <sup>(i)</sup> simply indicates the training examples (we have  $N$  in this case)
- Here  $t$  is continuous, so this is a **regression problem**
- Model outputs  $y$ , an estimate of  $t$

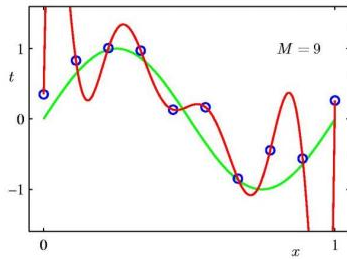
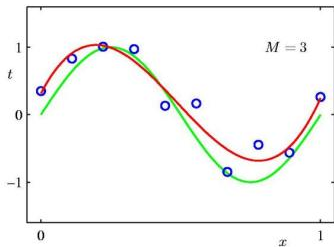
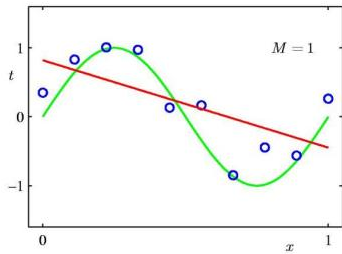
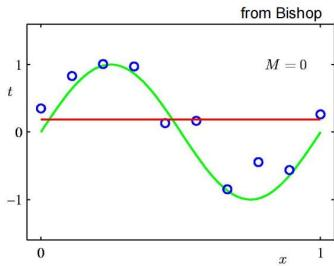
$$y(x) = w_0 + w_1x$$

- What type of **model** did we choose?
- Divide the dataset into training and testing examples
  - ▶ Use the training examples to construct hypothesis, or function approximator, that maps  $x$  to predicted  $y$
  - ▶ Evaluate hypothesis on test set

# Noise

- A simple model typically does not exactly fit the data
  - ▶ lack of fit can be considered **noise**
- Sources of noise:
  - ▶ Imprecision in data attributes (**input noise**, e.g., noise in per-capita crime)
  - ▶ Errors in data targets (**mis-labeling**, e.g., noise in house prices)
  - ▶ **Additional attributes** not taken into account by data attributes, affect target values (latent variables). In the example, what else could affect house prices?
  - ▶ **Model** may be **too simple** to account for data targets

# Which fit is best?



# Summary of Regression

## □ Regression: to predict continuous outputs $t$

- ▶ Consider proper **features** (inputs):  $x$  (or  $\mathbf{x}$  if vectors)
- ▶ **Training examples**, many  $x(i)$  for which  $t(i)$  is known (labeled)
- ▶ A **model**, a function that represents the relationship between  $x$  and  $t$

$$y = f(x, w)$$

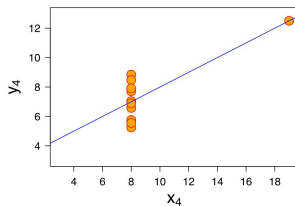
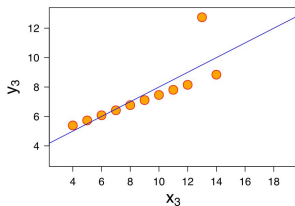
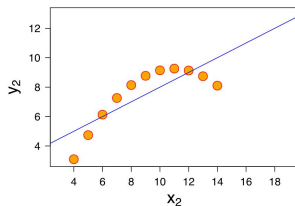
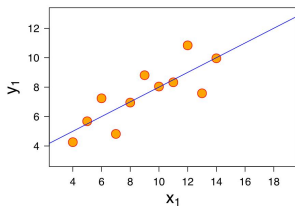
- ▶ A **loss** or a **cost** or an **objective** function, which tells us how well our model approximates the training examples
- ▶ **Optimization**, a way of finding the parameters  $w$  of our model that minimizes the loss function

# Linear Regression

- Linear regression
  - ▶ continuous outputs
  - ▶ simple model (linear)
- Introduce **key concepts**:
  - ▶ loss functions
  - ▶ generalization
  - ▶ optimization
  - ▶ model complexity
  - ▶ regularization



# Linear regression model



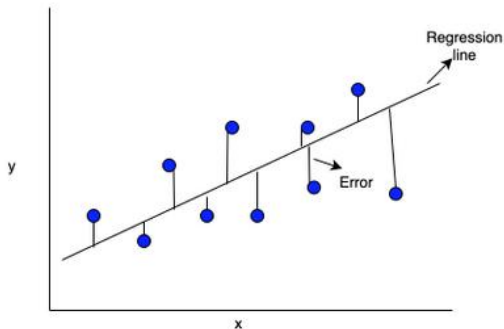
Input:  $\mathbb{R}$

Output:  $\mathbb{R}$

■ True model (unknown):  $t = f(x)$

■ Linear regression model:  $y(x) = w_0 + w_1 x$

## Loss function: Mean Squared Error (MSE)



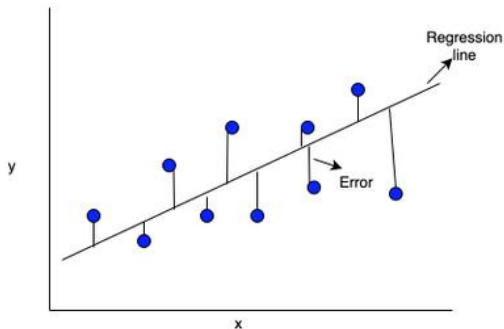
$$y(x) = w_0 + w_1 x$$

$$\text{Denote } w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

- Standard loss/cost/objective function measures the mean squared error between  $y$  and the true value  $t$

$$\begin{aligned} l(w) &= \frac{1}{N} \sum_{n=1}^N [t^{(n)} - y^{(n)}]^2 \\ &= \frac{1}{N} \sum_{n=1}^N [t^{(n)} - (w_0 + w_1 x^{(n)})]^2 \end{aligned}$$

## Loss function: Mean Squared Error (MSE)



The loss is the **mean of the squared vertical errors**

- Standard **loss/cost/objective function** measures the **mean squared error** between  $y$  and the true value  $t$

$$\begin{aligned} l(w) &= \frac{1}{N} \sum_{n=1}^N [t^{(n)} - y^{(n)}]^2 \\ &= \frac{1}{N} \sum_{n=1}^N [t^{(n)} - (w_0 + w_1 x^{(n)})]^2 \end{aligned}$$

## Loss function: Mean Squared Error (MSE)

- MSE loss:

$$l(w) = \frac{1}{N} \sum_{n=1}^N [t^{(n)} - y^{(n)}]^2$$

```
def mean_squared_error(true, pred):  
    squared_error = np.square(true - pred)  
    sum_squared_error = np.sum(squared_error)  
    mse_loss = sum_squared_error / true.size  
    return mse_loss
```

## Other loss function: Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{n=1}^N [t^{(n)} - y^{(n)}]^2}$$

```
def root_mean_squared_error(true, pred):  
    squared_error = np.square(true - pred)  
    sum_squared_error = np.sum(squared_error)  
    rmse_loss = np.sqrt(sum_squared_error / true.size)  
    return rmse_loss
```

## Other loss function: Relative Squared Error (RSE)

$$\text{RSE} = \frac{\sum_{n=1}^N [t^{(n)} - y^{(n)}]^2}{\sum_{n=1}^N [t^{(n)} - \bar{t}]^2}, \quad \bar{t} = \frac{1}{N} \sum_{n=1}^N t^{(n)}$$

- **Merit:** Insensitive to the mean and scale of samples

```
def relative_squared_error(true, pred):  
    true_mean = np.mean(true)  
    squared_error_num = np.sum(np.square(true - pred))  
    squared_error_den = np.sum(np.square(true - true_mean))  
    rse_loss = squared_error_num / squared_error_den  
    return rse_loss
```

## Other loss function: Mean Absolute Error (MAE)

$$\text{MAE} = \frac{1}{N} \sum_{n=1}^N |t^{(n)} - y^{(n)}|$$

```
def mean_absolute_error(true, pred):  
    abs_error = np.abs(true - pred)  
    sum_abs_error = np.sum(abs_error)  
    mae_loss = sum_abs_error / true.size  
    return mae_loss
```

## Other loss function: Relative Absolute Error (RAE)

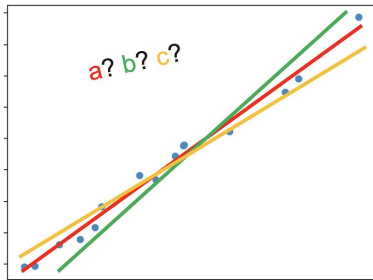
$$\text{RAE} = \frac{\sum_{n=1}^N |t^{(n)} - y^{(n)}|}{\sum_{n=1}^N |t^{(n)} - \bar{t}|}, \quad \bar{t} = \frac{1}{N} \sum_{n=1}^N t^{(n)}$$

```
def relative_absolute_error(true, pred):  
    true_mean = np.mean(true)  
    squared_error_num = np.sum(np.abs(true - pred))  
    squared_error_den = np.sum(np.abs(true - true_mean))  
    rae_loss = squared_error_num / squared_error_den  
    return rae_loss
```



# Optimization in training

- Linear regression model:  $y(x) = w_0 + w_1 x$
- MSE loss:  $l(w) = \frac{1}{N} \sum_{n=1}^N [t^{(n)} - y^{(n)}]^2$
- How do we obtain weights  $w$ ? Find  $w$  that **minimizes loss**  $l(w)$

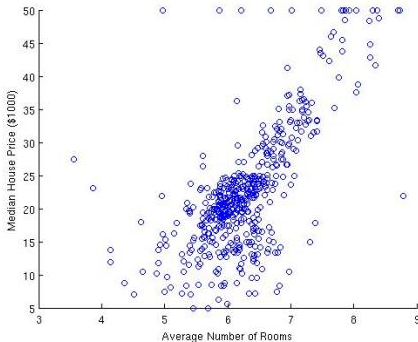


# Multi-dimensional Inputs

- One method of extending the model is to consider other input dimensions

$$y(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2$$

- In the Boston housing example, we can look at the number of rooms



# Linear Regression with Multi-dimensional Inputs

- Imagine now we want to predict the median house price from these multi-dimensional observations
- Each house is a data point  $n$ , with observations indexed by  $j$ :

$$\mathbf{x}^{(n)} = \left( x_1^{(n)}, \dots, x_j^{(n)}, \dots, x_d^{(n)} \right)$$

- We can incorporate the bias  $w_0$  into  $\mathbf{w}$ , by using  $x_0 = 1$ , then

$$y(\mathbf{x}) = w_0 + \sum_{j=1}^d w_j x_j = \mathbf{w}^T \mathbf{x}$$

- We can then solve for  $\mathbf{w} = (w_0, w_1, \dots, w_d)$ . How?

- Linear regression model:

$$y(x) = w^T \mathbf{x}, \quad \mathbf{x} = \begin{bmatrix} 1 \\ x \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}, \quad w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$$

- MSE loss:

$$\begin{aligned} l(w) &= \frac{1}{2N} \sum_{n=1}^N [t^{(n)} - y(x^{(n)})]^2 \\ &= \frac{1}{2N} \sum_{n=1}^N [t^{(n)} - w^T \mathbf{x}^{(n)}]^2 \end{aligned}$$

- How do we obtain weights  $w$ ? Find  $w$  that **minimizes loss**  $l(w)$

- We can concatenate partial derivatives of a multivariate function with respect to all its variables to obtain the **gradient** vector of the function.

- Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ . Then the gradient of the function is

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

where  $\nabla_{\mathbf{x}} f(\mathbf{x})$  is often replaced by  $\nabla f(\mathbf{x})$  when there is no ambiguity.

## Detour: Convex function

**Definition 1.** A function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is convex if its domain is a convex set and for all  $x, y$  in its domain, and all  $\lambda \in [0, 1]$ , we have

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y).$$

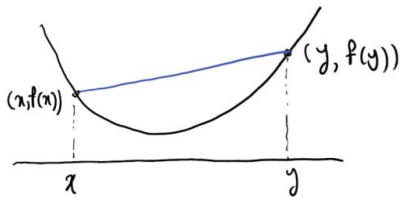
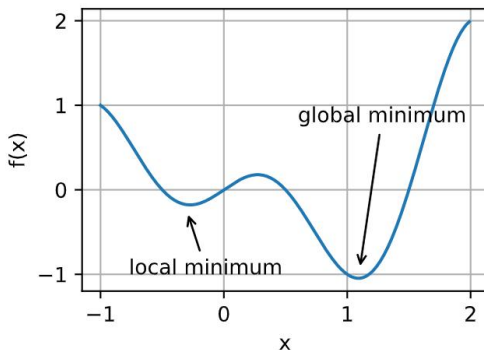


Figure 1: An illustration of the definition of a convex function

## Detour: Local minima

$$f(x) = x \cdot \cos(\pi x) \text{ for } -1.0 \leq x \leq 2.0$$



- The cost function usually has **many local optima**.

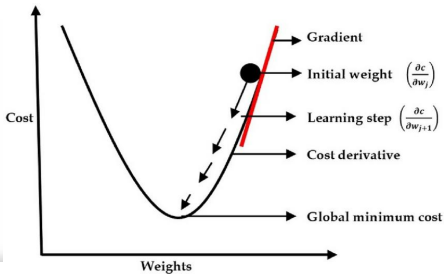
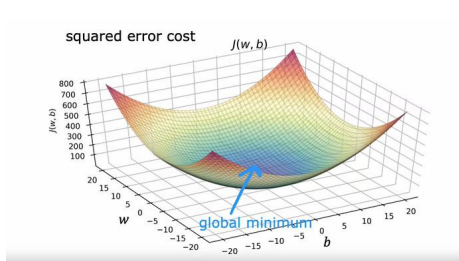
$$\nabla f(\bar{x}) = 0$$

# Detour: Global optimal

**Corollary 1.** *Consider an unconstrained optimization problem*

$$\begin{aligned} \min f(x) \\ \text{s.t. } x \in \mathbb{R}^n, \end{aligned}$$

where  $f$  is convex and differentiable. Then, any point  $\bar{x}$  that satisfies  $\nabla f(\bar{x}) = 0$  is a global minimum.





## Back to our problem

- Linear regression model:

$$y(x) = w^T \mathbf{x}, \quad \mathbf{x} = \begin{bmatrix} 1 \\ x \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}, \quad w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$$

- MSE loss:

$$\begin{aligned} l(w) &= \frac{1}{2N} \sum_{n=1}^N [t^{(n)} - y(x^{(n)})]^2 \\ &= \frac{1}{2N} \sum_{n=1}^N [t^{(n)} - w^T \mathbf{x}^{(n)}]^2 \end{aligned}$$

- Find  $w$  that **minimizes loss**  $l(w)$

□ Let  $\mathbf{x}$  be an  $n$ -dimensional vector, the following rules are often used.

- For all  $\mathbf{A} \in \mathbb{R}^{m \times n}$ ,  $\nabla_{\mathbf{x}} \mathbf{A} \mathbf{x} = \mathbf{A}^{\top}$ ,
- For all  $\mathbf{A} \in \mathbb{R}^{n \times m}$ ,  $\nabla_{\mathbf{x}} \mathbf{x}^{\top} \mathbf{A} = \mathbf{A}$ ,
- For all  $\mathbf{A} \in \mathbb{R}^{n \times n}$ ,  $\nabla_{\mathbf{x}} \mathbf{x}^{\top} \mathbf{A} \mathbf{x} = (\mathbf{A} + \mathbf{A}^{\top}) \mathbf{x}$ ,
- $\nabla_{\mathbf{x}} \|\mathbf{x}\|^2 = \nabla_{\mathbf{x}} \mathbf{x}^{\top} \mathbf{x} = 2\mathbf{x}$ .

## Least square solution

- The cost function  $l(w)$  is convex

$$\begin{aligned}l(w) &= \frac{1}{2N} \sum_{n=1}^N [t^{(n)} - y(x^{(n)})]^2 \\ &= \frac{1}{2N} \sum_{n=1}^N [t^{(n)} - w^T \mathbf{x}^{(n)}]^2\end{aligned}$$

- We take the gradient and let it equal to 0. Then find the solution.

$$\nabla l(w) = -\frac{1}{N} \sum_{n=1}^N (t^{(n)} - w^T \mathbf{x}^{(n)}) \mathbf{x}^{(n)} = 0$$

- In the matrix form, we have

$$\nabla l(w) = -\frac{1}{N} \mathbf{X}^T (t - \mathbf{X}w) = 0$$

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ \vdots \\ (\mathbf{x}^{(N)})^T \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{x}_1^{(1)} & \cdots & \mathbf{x}_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathbf{x}_1^{(N)} & \cdots & \mathbf{x}_d^{(N)} \end{bmatrix}, \quad t = \begin{bmatrix} t^{(1)} \\ \vdots \\ t^{(N)} \end{bmatrix}$$

## Least square solution

- Take  $N = 2$  for an example:

$$\begin{aligned}\nabla l(w) &= -\frac{1}{N} \left\{ [t^{(1)} - w^T \mathbf{x}^{(1)}] \mathbf{x}^{(1)} + [t^{(2)} - w^T \mathbf{x}^{(2)}] \mathbf{x}^{(2)} \right\} \\ &= -\frac{1}{N} \left\{ \begin{bmatrix} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} \end{bmatrix} \begin{bmatrix} t^{(1)} - w^T \mathbf{x}^{(1)} \\ t^{(2)} - w^T \mathbf{x}^{(2)} \end{bmatrix} \right\} \\ &= -\frac{1}{N} \mathbf{X}^T (t - \mathbf{X}w) = 0\end{aligned}$$




$$w = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T t$$


# Linear regression with least square solution

```
import numpy as np
import matplotlib.pyplot as plt
```

```
class OLSLinearRegression:
```

 Explain | Doc | Test | X


```
def _ols(self, X, y):
    # Least square estimation
    return np.linalg.inv(X.T @ X) @ X.T @ y
```

 Explain | Doc | Test | X


```
def _preprocess_data_X(self, X):
    # Extend X by adding a constant 1 as the first element
    m, n = X.shape # m represent the sample number, and n represent the feature number
    X_ = np.empty([m, n+1])
    X_[:, 0] = 1
    X_[:, 1:] = X

    return X_
```

# Linear regression with least square solution

 Explain | Doc | Test | X

```
def train(self, X_train, y_train):  
    # Train the model  
    X_train = self._preprocess_data_X(X_train)  
    self.W = self._ols(X_train, y_train)
```

 Explain | Doc | Test | X

```
def predict(self, X):  
    # Predict the output for a given input  
    X = self._preprocess_data_X(X)  
    return X @ self.W
```

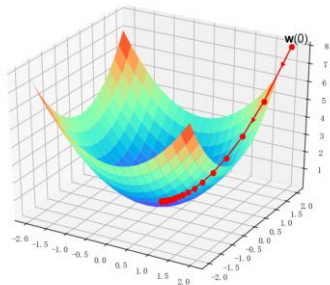
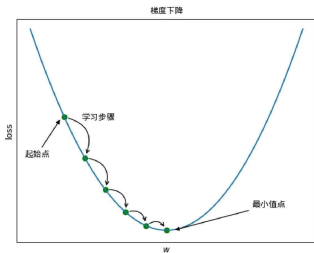
# Gradient descent idea

■ One straightforward method: **gradient descent**

- initialize  $w$  (e.g., randomly)
- repeatedly update  $w$  based on the gradient

$$w \leftarrow w - \lambda \nabla l(w)$$

- $\lambda$  is the **learning rate**

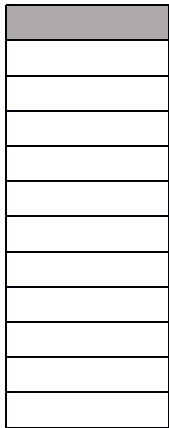




# Three ways of gradient descent

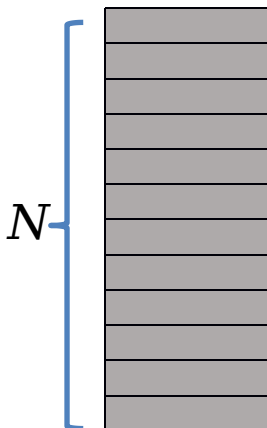
Stochastic  
Gradient Descent:

SGD



Batch Gradient  
Descent:

BGD



Mini-Batch Gradient  
Descent:

MBGD



# Stochastic Gradient Descent (SGD)

## Algorithm 1. Stochastic gradient descent (SGD)

1. Initialize  $w$  (e.g., randomly)
2. **for**  $i = 1$  to  $n\_epoch$  **do**
3.     Randomly shuffle and pick one sample  $(x^{(n)}, t^{(n)})$  in the training set
4.     Update:

$$w \leftarrow w + \lambda \left[ \underbrace{t^{(n)} - y(x^{(n)})}_{\text{error}} \right] \mathbf{x}^{(n)}$$

5. **end for**

- SGD: update the parameters for each sample in turn, according to its own gradient
- As error approaches zero, so does the update ( $w$  stops changing)

# Batch Gradient Descent (BGD)

## Algorithm 2. Batch gradient descent (BGD)

1. Initialize  $w$  (e.g., randomly)

2. **for**  $i = 1$  to  $n\_epoch$  **do**

3.     Update:

$$w \leftarrow w + \lambda \frac{1}{N} \sum_{n=1}^N \left[ t^{(n)} - y(x^{(n)}) \right] \mathbf{x}^{(n)}$$

4. **end for**

- BGD: average updates across every sample in training set, then change the parameters according to the gradient
- As error approaches zero, so does the update ( $w$  stops changing)

**Algorithm 2.** Batch gradient descent (BGD)

- Update in the matrix form:

$$w \leftarrow w + \lambda \frac{1}{N} \mathbf{X}^T (t - \mathbf{X}w)$$

# Mini-Batch Gradient Descent (MBGD)

## Algorithm 3. Mini-Batch gradient descent (MBGD)

1. Initialize  $w$  (e.g., randomly)
2. **for**  $i = 1$  to  $n\_epoch$  **do**
3.     shuffle the training set and partition into a number of mini-batches
4.     **for**  $j = 1$  to  $\text{floor}(\frac{N}{m})$ , **do**
5.         Update:
$$w \leftarrow w + \lambda \frac{1}{m} \sum_{n \in \mathcal{B}_j} \left[ t^{(n)} - y(x^{(n)}) \right] \mathbf{x}^{(n)}$$
6.     **end for**
7. **end for**

# Mini-Batch Gradient Descent (MBGD)

**Algorithm 3.** Mini-Batch gradient descent (MBGD)

- Update in the matrix form:

$$w \leftarrow w + \lambda \frac{1}{m} \mathbf{X}_{\mathcal{B}_i}^T (t_{\mathcal{B}_i} - \mathbf{X}_{\mathcal{B}_i} w)$$

# Why gradient descent works?

- Gradient descent in one dimension is an excellent example to explain why the gradient descent algorithm may reduce the value of the objective function.

$$f(x + \epsilon) = f(x) + \epsilon f'(x) + \mathcal{O}(\epsilon^2). \quad \text{Taylor expansion}$$

$$\text{choose } \epsilon = -\eta f'(x).$$



$$f(x - \eta f'(x)) = f(x) - \eta f'^2(x) + \mathcal{O}(\eta^2 f'^2(x)).$$

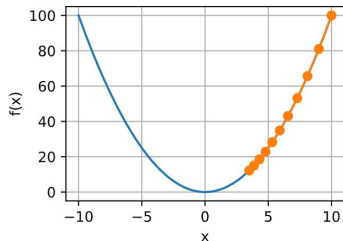


$$f(x - \eta f'(x)) \lesssim f(x).$$



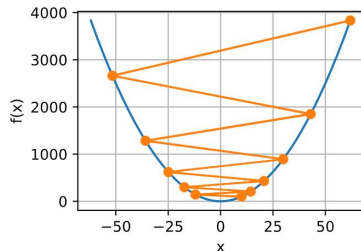
$$x \leftarrow x - \eta f'(x) \quad \text{then } f(x) \text{ decreases}$$

# Learning rate is a hyper-parameter



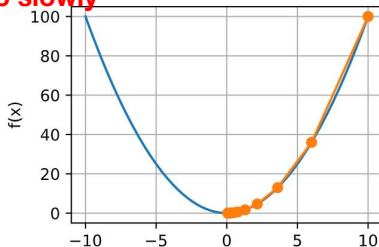
$\eta = 0.05$ .

**converge too slowly**



$\eta = 1.1$

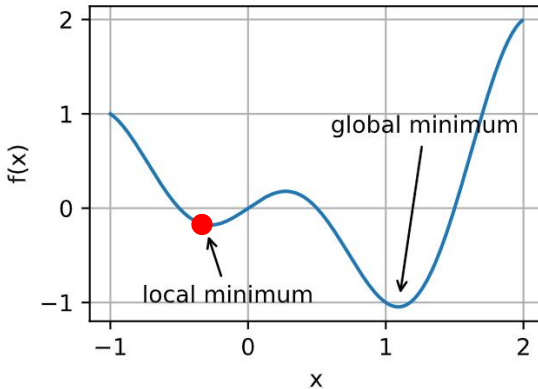
**diverge**



$\eta = 0.2$ .




# Trap into local minima



# Linear regression with BGD

```
31 class GDLinearRegression:
```

```
32  Explain | Doc | Test | X
```

```
33 def __init__(self, n_feature = 1, n_iter = 200, lr = 1e-3, tol = None):
```


```
34     self.n_iter = n_iter      # Maximum iteration steps
```

```
35     self.lr = lr             # Learning rate
```

```
36     self.tol = tol           # Threshold for stopping iteration
```


```
37     self.W = np.random.random(n_feature + 1) * 0.05      # Model parameters
```

```
38     self.loss = []          # The loss value
```

```
39  Explain | Doc | Test | X
```

```
40 def _loss(self, y, y_pred):
```

```
41     return np.sum((y_pred - y) ** 2) / y.size
```

```
42  Explain | Doc | Test | X
```

```
43 def _gradient(self, X, y, y_pred):
```

```
44     return (y_pred - y) @ X / y.size
```

```
45
```

# Linear regression with BGD

```
46 def _preprocess_data(self, X):  
47     m, n = X.shape  
48     X_ = np.empty([m, n+1])  
49     X_[:, 0] = 1  
50     X_[:, 1:] = X  
51  
52     return X_  
53
```

 Explain | Doc | Test | X

```
54 def _predict(self, X):  
55     return X @ self.W  
56
```

 Explain | Doc | Test | X

```
57 def predict(self, X):  
58     X = self._preprocess_data(X)  
59     return X @ self.W
```

# Linear regression with BGD

```
61 def batch_update(self, X, y):
62
63     if self.tol is not None:
64         loss_old = np.inf
65
66     for iter in range(self.n_iter):
67         y_pred = self._predict(X)
68         loss = self._loss(y, y_pred)
69         # print(loss)
70         self.loss.append(loss)
71
72         if self.tol is not None:
73             if np.abs(loss_old - loss) < self.tol:
74                 break
75             loss_old = loss
76
77     grad = self._gradient(X, y, y_pred)
78     self.W = self.W - self.lr * grad
79
```

# Linear regression with BGD

```
80     def train(self, X_train, y_train):
81         X_train = self._preprocess_data(X_train)
82         self.batch_update(X_train, y_train)
83
84     def plot_loss(self):
85         plt.plot(self.loss)
86         plt.grid()
87         plt.show()
88
89 if __name__ == '__main__':
90
91     X_train = np.arange(100).reshape(100,1)
92     a, b = 1, 10
93     y_train = a * X_train + b
94     y_train = y_train.flatten()
95     _, n_feature = X_train.shape
96     print(n_feature)
97
98     gd_lreg_1 = GDLinearRegression(n_feature=n_feature, n_iter=3000, lr=0.001, tol=0.00001)
99     gd_lreg_1.train(X_train, y_train)
100    gd_lreg_1.plot_loss()
101    print(f'Learned weights are {gd_lreg_1.W}')
```