# Lecture 3: Training Models

Dr.-Ing. Sudchai Boonto, Assistant Professor

February 2, 2018

Department of Control System and Instrument Engineering, KMUTT

# Linear Regression

# Performance Learning

- the sum of squared prediction errors

$$V(\theta, Z^N) = \frac{1}{N}\sum_{k=p}^{N}(y(k) - \hat{y}(k|k-1,\theta))^2 = \frac{1}{2N}\sum_{k=p}^{N}\varepsilon^2(k,\theta),$$

  - $Z^N = \{y(k), \varphi(k); k = 1, \ldots, N\}$ represents a set of $N$ samples of measured input and output data,
  - $y(k)$ is the measured output at sampling instant $k$,
  - $\hat{y}(k)$ the output predicted by the network, and the parameter vector $\theta \in \mathbb{R}^{n_p}$ contains all adjustable network parameters, i.e. weights and bias values.

# Linear Regression model prediction

$$\hat{y}(x_i) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n,$$

- $\hat{y}$ is the predicted value
- $n$ is the number of features.
- $x_i$ is the $i^{\text{th}}$ feature value
- $\theta_i$ the $j^{\text{th}}$ model parameter (including the bias term $\theta_0$ and the feature weights $\theta_1, \theta_2, \ldots, \theta_n$)

This can be written using a vectorized form

$$\hat{y} = h_\theta(x) = \theta^T x$$

We need to minimize defined cost function. We use normal letter for both scalar and vector.

# Normal Equation

# Mean Square Error (MSE)

The most common used objective function is the Mean Square Error (MSE):

$$V(\theta) = \frac{1}{2N}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 = \frac{1}{2N}\sum_{i=1}^{m}\left(\theta^T x_i - y_i\right)^2 = \frac{1}{2N}E^T E,$$

where $E = (X\theta - y)$ To minimize this, we use a simple gradient

$$\begin{aligned}\frac{\partial}{\partial\theta}V(\theta) &= \frac{\partial}{\partial\theta}\frac{1}{2N}(X\theta - y)^T(X\theta - y) = 0\\ X^T X\theta &= X^T y\\ \theta &= \left(X^T X\right)^{-1} X^T y\end{aligned}$$

Since the data set $y$ is always deteriorate by noise, we have

$$\hat{\theta} = \left(X^T X\right)^{-1} X^T y$$

as approximation of the real $\theta$. We call the equation *Normal Equation*. Where $\hat{\theta}$ is the value of $\theta$ that minimizes the cost function and $y$ is the vector of target value.

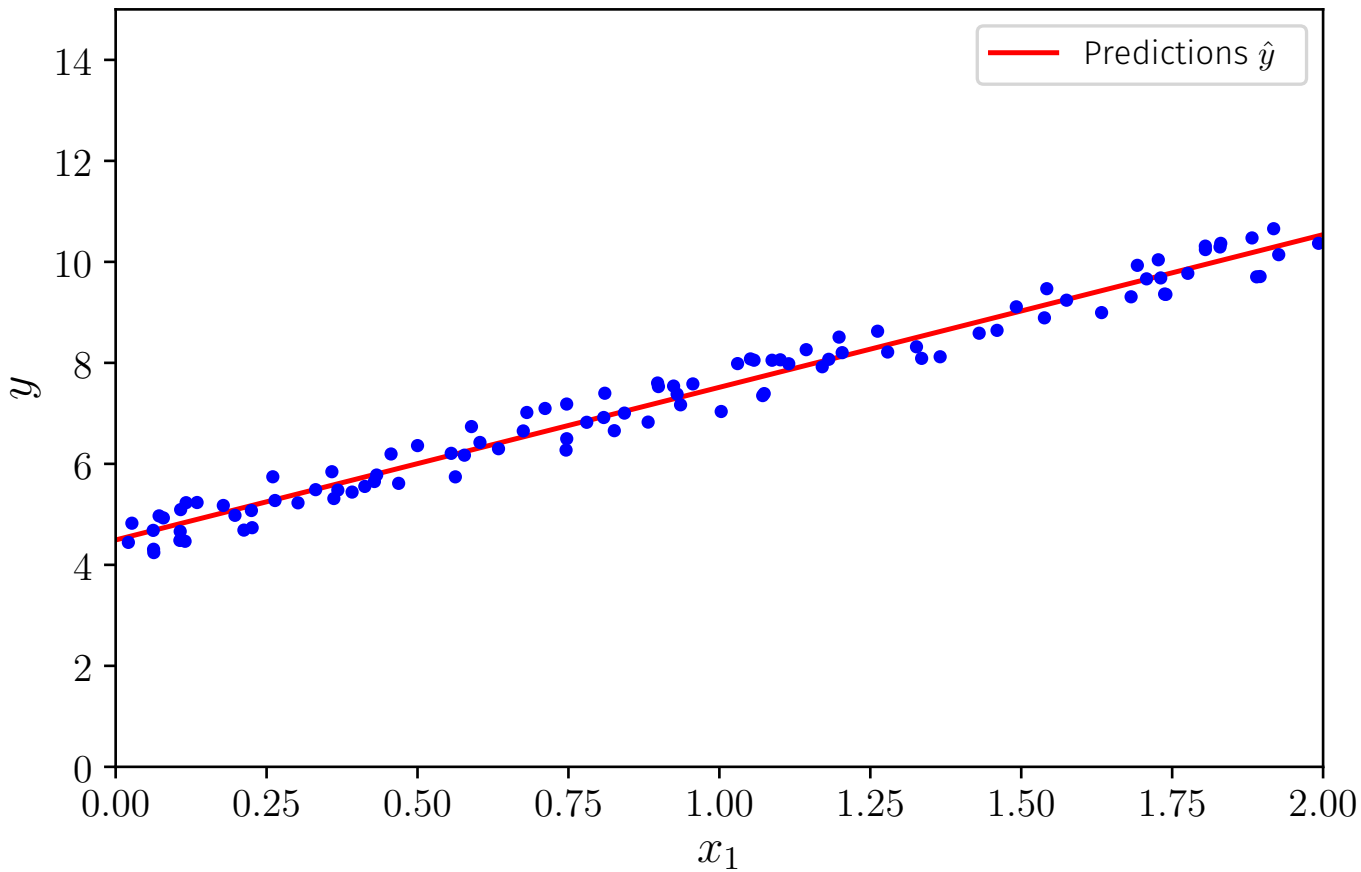
# Normal Equation

Example of Python code

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

# generate training data
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.rand(100,1)

X_b = np.c_[np.ones((100,1)), X] # add x0 = 1 to each instance
# normal equation
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)

# testing model
X_new = np.array([[0], [2]])
X_new_b = np.c_[np.ones((2,1)), X_new]
y_predict = X_new_b.dot(theta_best)
```

# Normal Equation

```
# plot data
plt.plot(X_new, y_predict, "r-")
plt.plot(X, y, "b.")
plt.show()
```




Normal equation gets very slow when the number of features grows large.

# Gradient Based Methods: Basic Idea

# Performance Learning

The objective is to find the minimizing value

$$\hat{\theta} = \arg\min_{\theta} V(\theta, Z^N)$$

In the neighborhood of a given value $\theta^0$ of the parameter vector, the performance index can be expanded into a Taylor series

$$V(\theta) = V(\theta^0) + \nabla V^T(\theta)\Big|_{\theta^0}(\theta - \theta^0) + \frac{1}{2}(\theta - \theta^0)^T \nabla^2 V(\theta)\Big|_{\theta^0}(\theta - \theta^0) + \dots$$

- The gradient of $V(\theta)$ is denoted by

$$\nabla V(\theta) = \left[\frac{\partial V}{\partial \theta_1} \quad \cdots \quad \frac{\partial V}{\partial \theta_1}\right]^T$$

# Hessian

- The Hessian of the function $V(\theta)$ is denoted by

$$\nabla^2 V(\theta) = \begin{bmatrix} \frac{\partial^2 V}{\partial \theta_1^2} & \frac{\partial^2 V}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 V}{\partial \theta_1 \partial \theta_{n_p}} \\ \frac{\partial^2 V}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 V}{\partial \theta_2^2} & \cdots & \frac{\partial^2 V}{\partial \theta_2 \partial \theta_{n_p}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 V}{\partial \theta_{n_p} \partial \theta_1} & \frac{\partial^2 V}{\partial \theta_{n_p} \partial \theta_2} & \cdots & \frac{\partial^2 V}{\partial \theta_{n_p}^2} \end{bmatrix}$$

# Steepest Descent

Starting from an initial guess $\theta(0)$, an iterative search for the best estimate of the parameter vector at iteration step $l$ generally takes the form

$$\theta(l+1) = \theta(l) + \alpha f(l) = \theta(l) + \Delta\theta(l)$$

- $f(l) \in \mathbb{R}^{n_p}$ is called the **search direction** at iteration step $l$
- a constant $\alpha > 0$ is called the **learning rate**.

**Directional derivative:**

- the element $\dfrac{\partial V}{\partial \theta_i}$ of the gradient vector and $\dfrac{\partial^2 V}{\partial \theta_i^2}$ of the diagonal of the Hessian are the first and second derivative of $V$ along the $\theta_i$ axis.
- the firest and second derivatives along the direction of an arbitrary vector $f$ – the directional derivatives along $f$ – are given by

$$\frac{f^T \nabla V(\theta)}{\|f\|} \quad \text{and} \quad \frac{f^T \nabla^2 V(\theta) f}{\|f\|^2}$$

respectively.

# Directional Derivative: Example

Consider the function

$$V(\theta) = \theta_1^2 + 9\theta_2^2, \quad \theta^0 = \begin{bmatrix} 1.5 \\ 0.5 \end{bmatrix}, \quad f = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

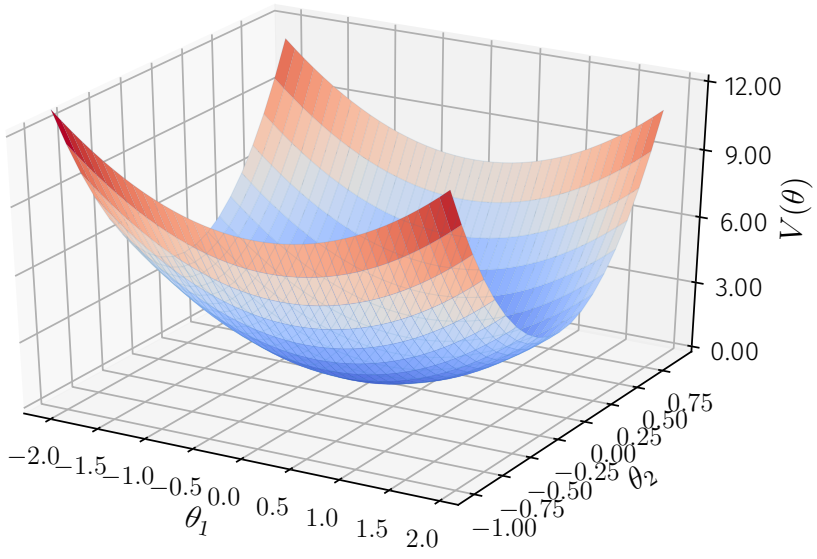
The gradient of $V(\theta)$ at $\theta^0$ is

$$\nabla V|_{\theta^0} = \begin{bmatrix} 2\theta_1 \\ 18\theta_2 \end{bmatrix}\Bigg|_{\theta^0} = \begin{bmatrix} 3 \\ 9 \end{bmatrix}$$

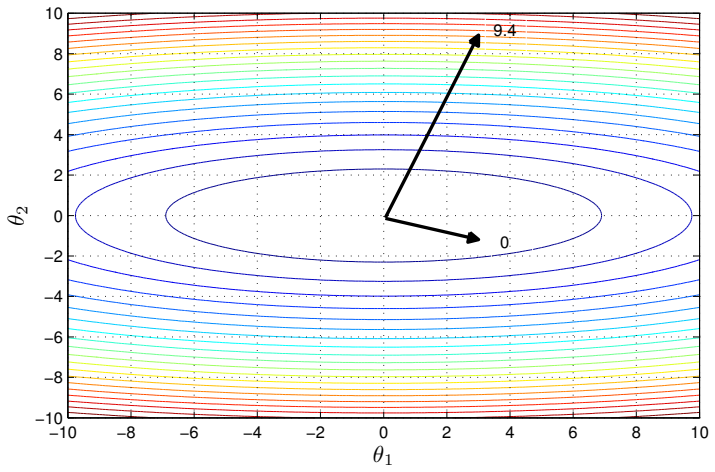
The directional derivative along $f$ is therefore

$$\frac{1}{\|f\|} \begin{bmatrix} 3 & -1 \end{bmatrix} \begin{bmatrix} 3 \\ 9 \end{bmatrix} = 0$$

# Directional Derivative




(a) Surface plot of $V(\theta)$




(b) Contour plot of $V(\theta)$

- the directional derivative along $f$ is zero because $f$ points in the direction of the tangent of the level curve at $\theta^0$
- the slope has its maximum value in the direction of the gradient, and its minimum value in the opposite direction.

# Conditions for Minima

The objective of performance learning is to minimize the network performance index $V(\theta)$. A point $\theta^0$ is called

- a strong minimum of $V(\theta)$ if a scalar $\beta$ exists such that $V(\theta^0) < V(\theta^0 + \Delta\theta)$ for all $\Delta\theta$ such that $\beta > \|\Delta\theta\| > 0$
- a weak minimum of $V(\theta)$ if it is not a strong minimum and if a scalar $\beta$ exists such that $V(\theta^0) \leq V(\theta^0 + \Delta\theta)$ for all $\Delta\theta$ such that $\beta > \|\Delta\theta\| > 0$
- a global minimum of $V(\theta)$ if $V(\theta^0) < V(\theta^0 + \Delta\theta)$ for all $\Delta\theta \neq 0$.

A linear approximation of $V(\theta)$ in the neighborhood of $\theta^0$ is

$$V(\theta) = V(\theta^0 + \Delta\theta) \approx V(\theta^0) + \nabla V(\theta)^T \Big|_{\theta^0} \Delta\theta$$

A necessary (but not sufficient) condition for $\theta^0$ to be a strong minimum is

$$\nabla V(\theta)|_{\theta^0} = 0$$

Points satisfying this condition are called stationary points of $V(\theta)$.

# Conditions for Minima cont.

Whether or not a stationary point $V(\theta)$ is a minimum depends on the higher order terms of the Taylor series expansion

$$V(\theta^0 + \Delta\theta) = V(\theta^0) + \frac{1}{2}\Delta\theta^T \nabla^2 V(\theta)\Big|_{\theta^0} \Delta\theta + \cdots$$

In a small neighborhood of $\theta^0$ we may neglect third and higher order terms, and a sufficient condition for $\theta^0$ to be a strong minimum is

$$\nabla V^2(\theta)\big|_{\theta^0} > 0$$

i.e. the Hessian at $\theta^0$ is positive definite. Note that this is not a necessary condition, since $\theta^0$ can still be a strong minimum even if the second order term in the Taylor series is zero.

# Quadratic Functions

A quadratic function has the form

$$F(x) = \frac{1}{2} x^T Q x + p^T x + r$$

where $F$ is scalar function of $x \in \mathbb{R}^n$, $Q = Q^T \in \mathbb{R}^{n \times n}$ and $p$ and $r$ are a column vector and a scalar, respectively. We have

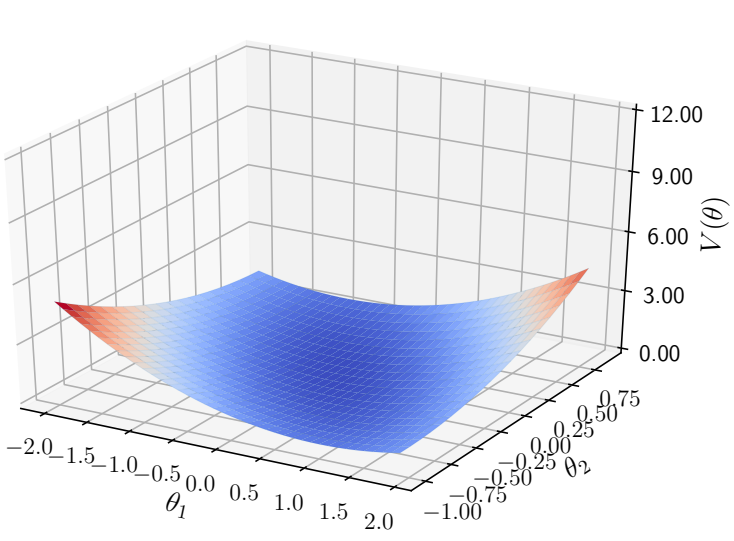
$$\nabla F(x) = Qx + p \quad \text{and} \quad \nabla^2 F(x) = Q$$

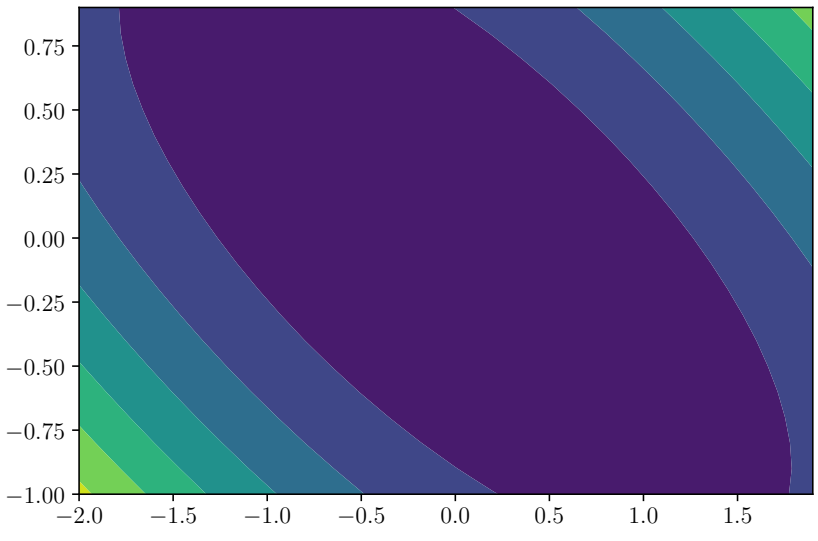
The following three quadratic functions all have a stationary point at $x = 0$; they illustrate how the Hessian determines the character of the stationary point. The Hessian of the quadratic function

$$F(x) = \frac{1}{2} x^T \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} x$$

has eigenvalues 3 and 1 and is positive definite. Thus, the origin is a strong minimum. The eigenvectors of the Hessian point in the direction of the principal axes of the ellipse-shaped level curves; the second derivatives (curvature) in theses directions are 3 and 1.

# Quadratic Functions



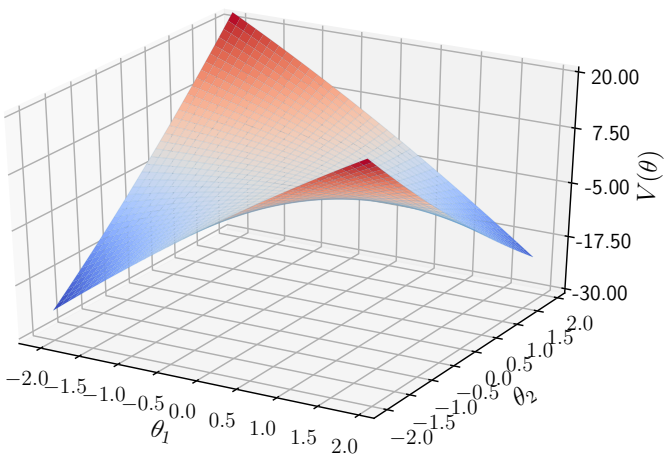



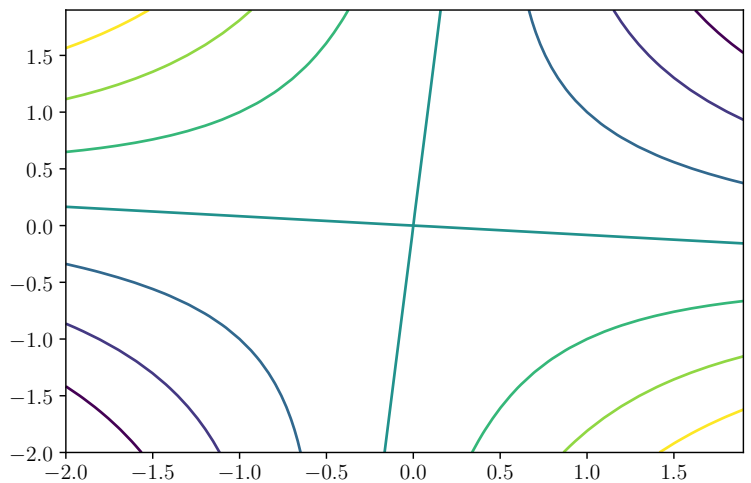
# Quadratic Functions

The Hessian of

$$F(x) = \frac{1}{2} x^T \begin{bmatrix} -1 & -6 \\ -6 & -1 \end{bmatrix} x$$

has eigenvalues 5 and -7 and is indefinite. The stationary point $\theta = 0$ is a minimum in the direction of the eigenvector corresponding to eigenvalue 5, and a maximum in the direction of the eigenvector corresponding to eigenvalue -7. Such a point is called a **saddle point**.



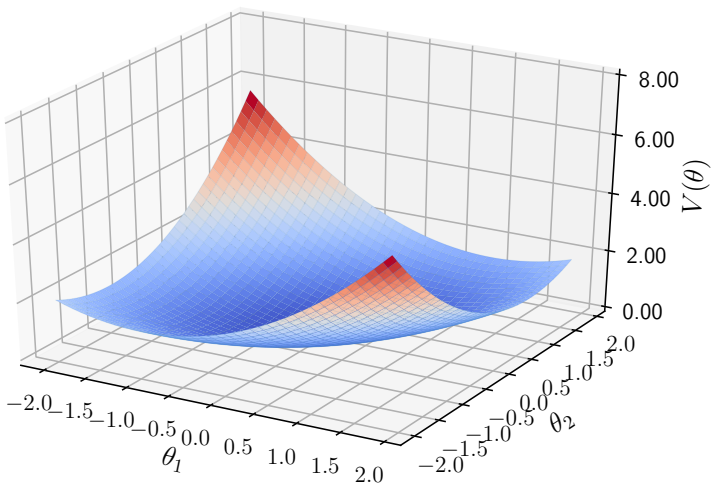



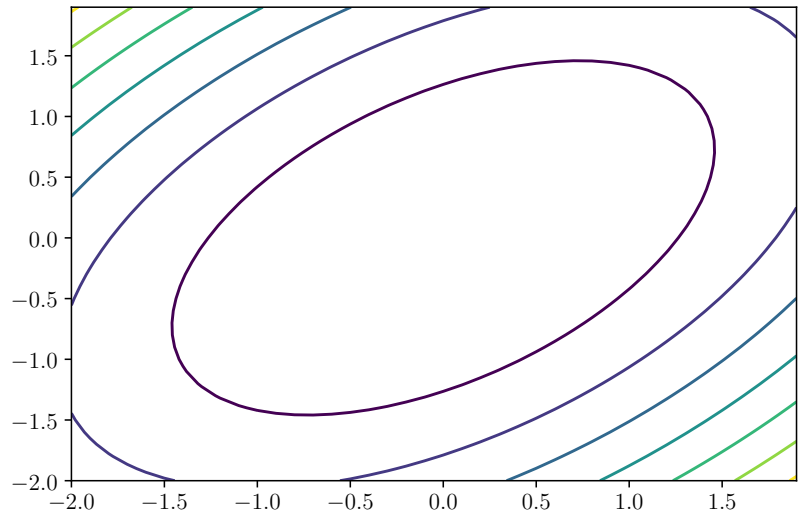
# Quadratic Functions

The Hessian of

$$F(x) = \frac{1}{2} x^T \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} x$$

has eigenvalues 2 and 0 and is positive semidefinite. The origin is a weak minimum. The valley has the direction of the eigenvector corresponding to the zero eigenvalue.



# Gradient descent variants

# Batch gradient descent

The cost function:

$$V(\theta) = \frac{1}{N} \sum_{i-1}^{N} \left( \theta^T x_{(i)} - y_{(i)} \right)^2$$

The gradient descent step (negative gradient)

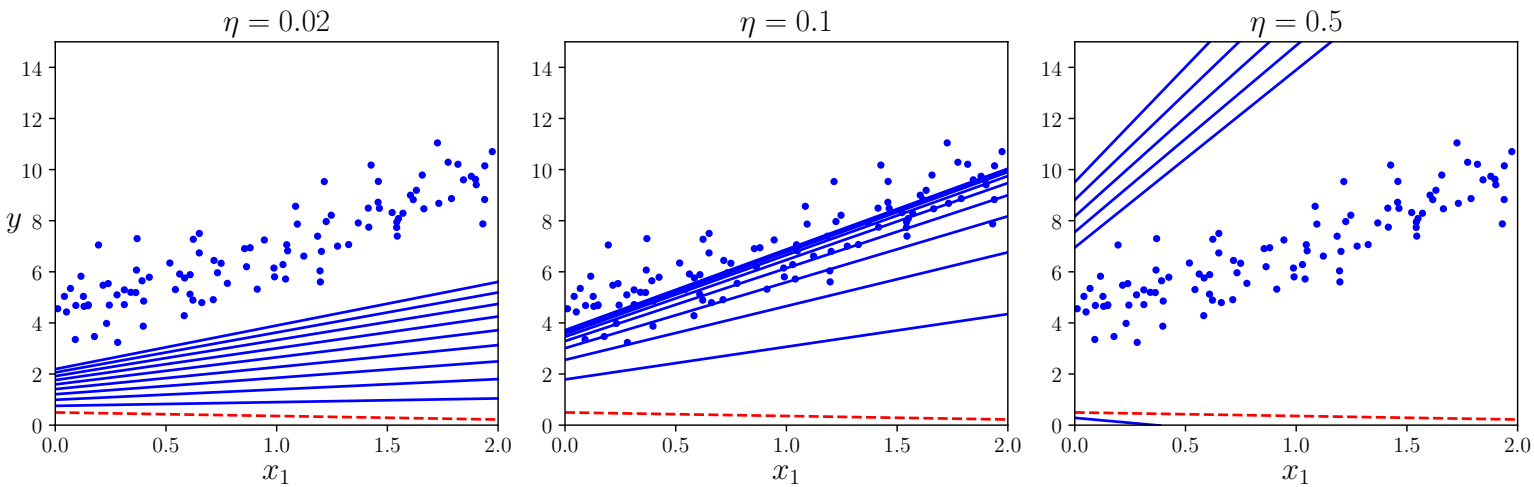
$$\theta_{(k+1)} = \theta_{(k)} - \eta \nabla_\theta V(\theta)$$

We need to calculate the gradients for the whole dataset to perform just one update, batch gradient descent can be very slow and is intractable for datasets that do not fit in memory. Batch gradient descent also does not allow us to update our model online.

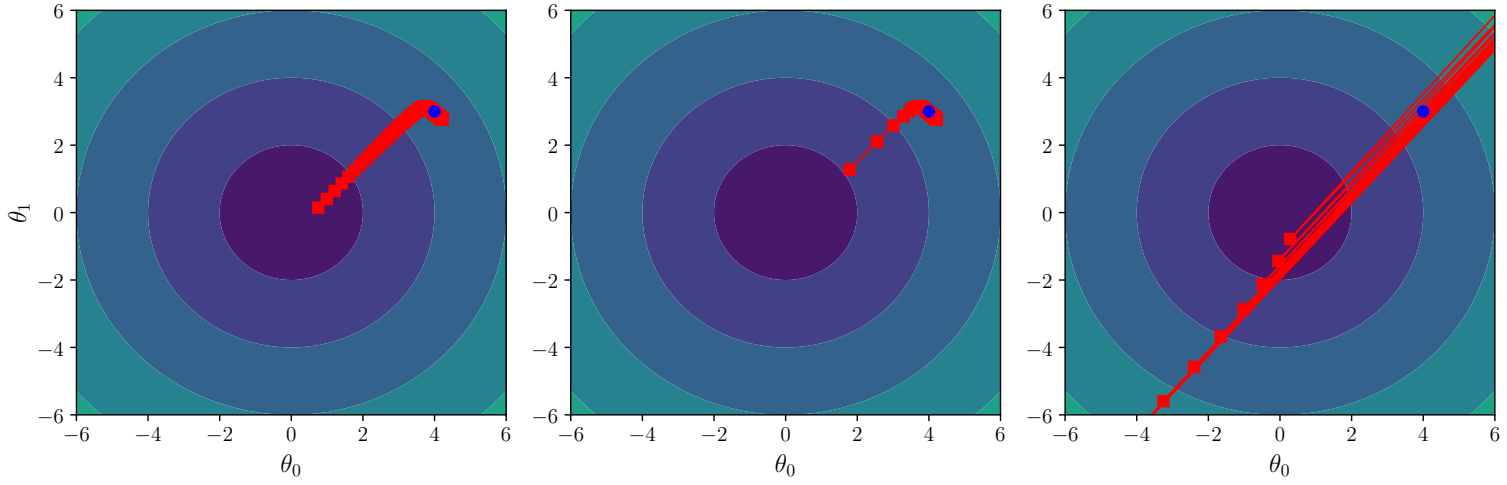
$$\nabla_\theta V(\theta) = \begin{bmatrix} \frac{\partial}{\partial \theta_0} V(\theta) \\ \frac{\partial}{\partial \theta_1} V(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} V(\theta) \end{bmatrix} = \frac{2}{N} X^T (X\theta - y)$$

# Batch gradient descent

```
eta = 0.1  #learing rate
n_iterations = 1000
m = 100

theta = np.random.rand(2,1)  # random initialization

for iteration in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - eta * gradients
```



# Batch gradient descent




- In the left hand side, the learning rate is too low: the algorithm will eventually reach the solution, but it will take a long time.
- In the middle, it is the perfect learning rate.
- On the right, the learning rate is too high: the algorithm diverges, jumping all the place and actually it is unstable.

# Stochastic gradient descent

The **Stochastic Gradient Descent (SGD)** can improve the speed of the batch gradient descent (BGD).

$$\theta_{(k+1)} = \theta_{(k)} - \eta \nabla_\theta V(\theta, x_{(i)}, y_{(i)}),$$

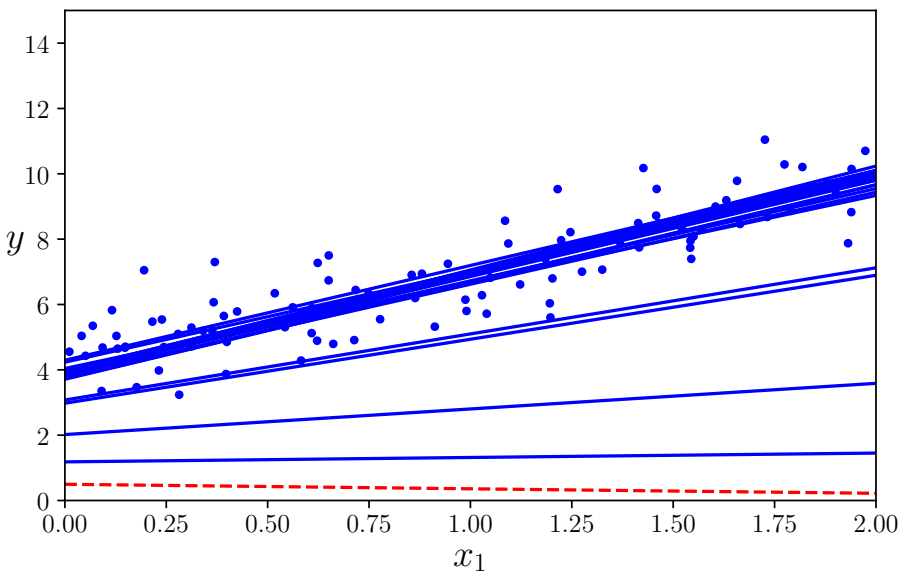
where $x_{(i)}$, and $y_{(i)}$ are each random training sample instant.

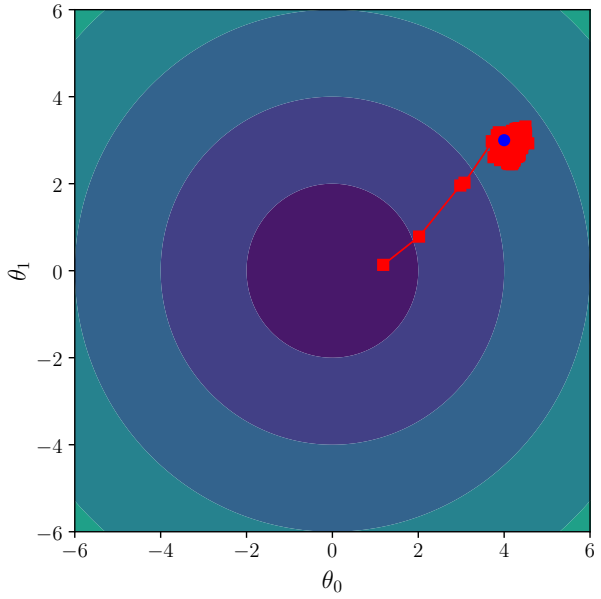
- It is must fast than the BGD. It is possible to train on huge training sets, since only one instance needs to be in memory at iteration.
- the algorithm is much less regular than BGD: instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average.
- Over time it will end up very close to the minimum, but one it gets there it will continue to bounce arond, never settling down.
- The SGD can jump out of local minima from the random manner, so SGD has a better chance of finding the global minimum than BGD.

# Stochastic gradient descent

```
m = 100
n_epochs = 50
t0, t1 = 5, 50  # learning schedule hyperparameters

def learning_schedule(t):
    return t0 / (t + t1)

theta = np.random.randn(2,1)        # random initialization
for epoch in range(n_epochs):
    for i in range(m):
        random_index = np.random.randint(m)     # m random index
        xi = X_b[random_index:random_index+1]
        yi = y[random_index:random_index+1]
        gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
        eta = learning_schedule(epoch * m + i)
        theta = theta - eta * gradients
```

# Stochastic gradient descent



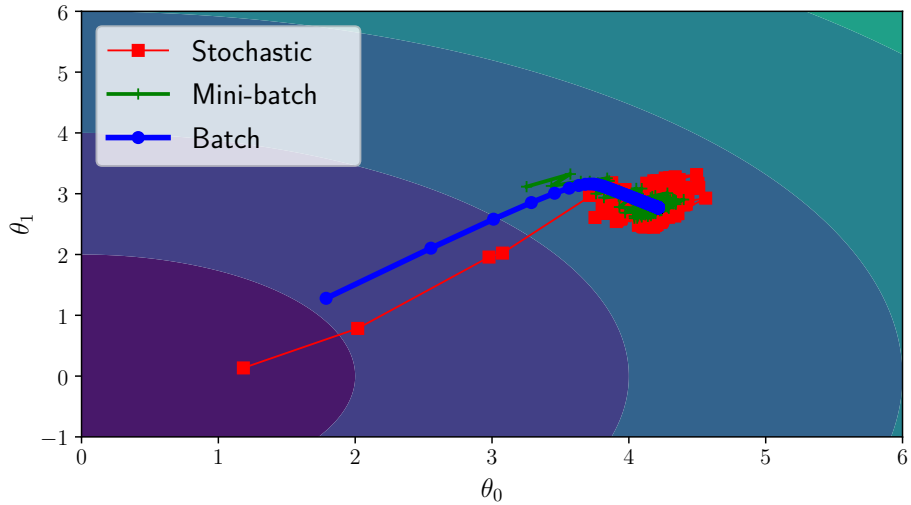



# Mini-batch gradient descent

*Mini-batch gradient descent* takes the best of both worlds and performs and updata for every mini-batch of $n$ training examples:

$$\theta_{(k+1)} = \theta_{(k)} - \eta \nabla_\theta V(\theta, x_{(i:i+n)}, y_{(i:i+n)})$$

- It can reduces the variance of the parameter updates, which can lead to more stable convergence.
- The common mini-batch sizes range between 50 and 256, but can vary of different applications.



# Reference


1. Sebatian Ruder, "*An Overview of Gradient Descent Optimization Algorithms*", arXiv:1609.04747v2, 2017
2. Aurélien, Géron, "*Hands-On Machine Learning with Scikit-Learn & TensorFlow*", O'reilly, 2017
3. S.P.K. Spielberg, R. B. Gopaluni, P. D. Loewen, "Deep Reinforcement Learning Approaches for Process Control", 2017 6th International Symposium on Advanced control of Industrial Processes (AdCONIP), May 28-31, 2017, Taipei, Taiwan