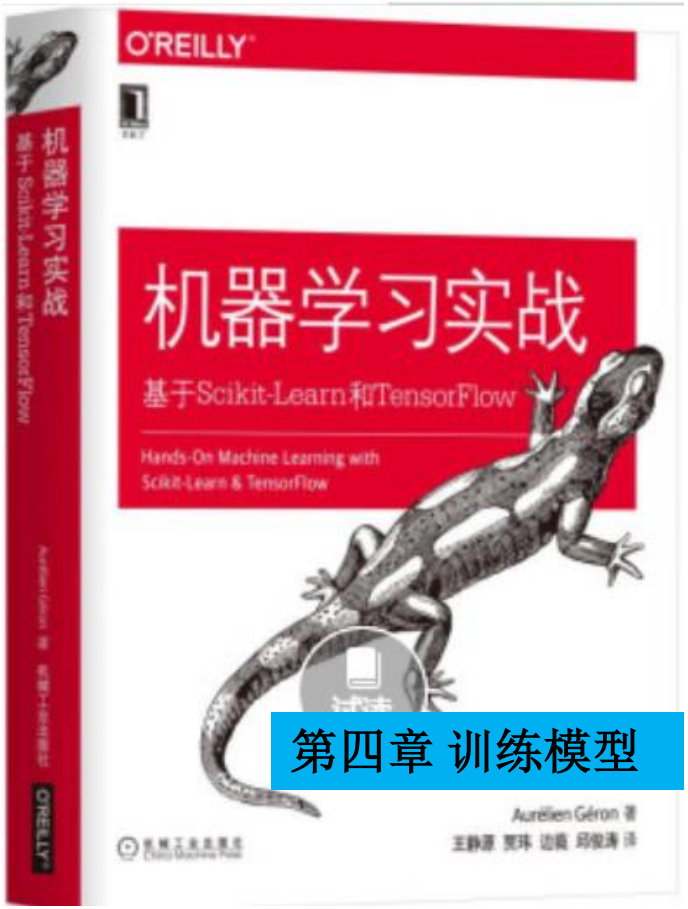
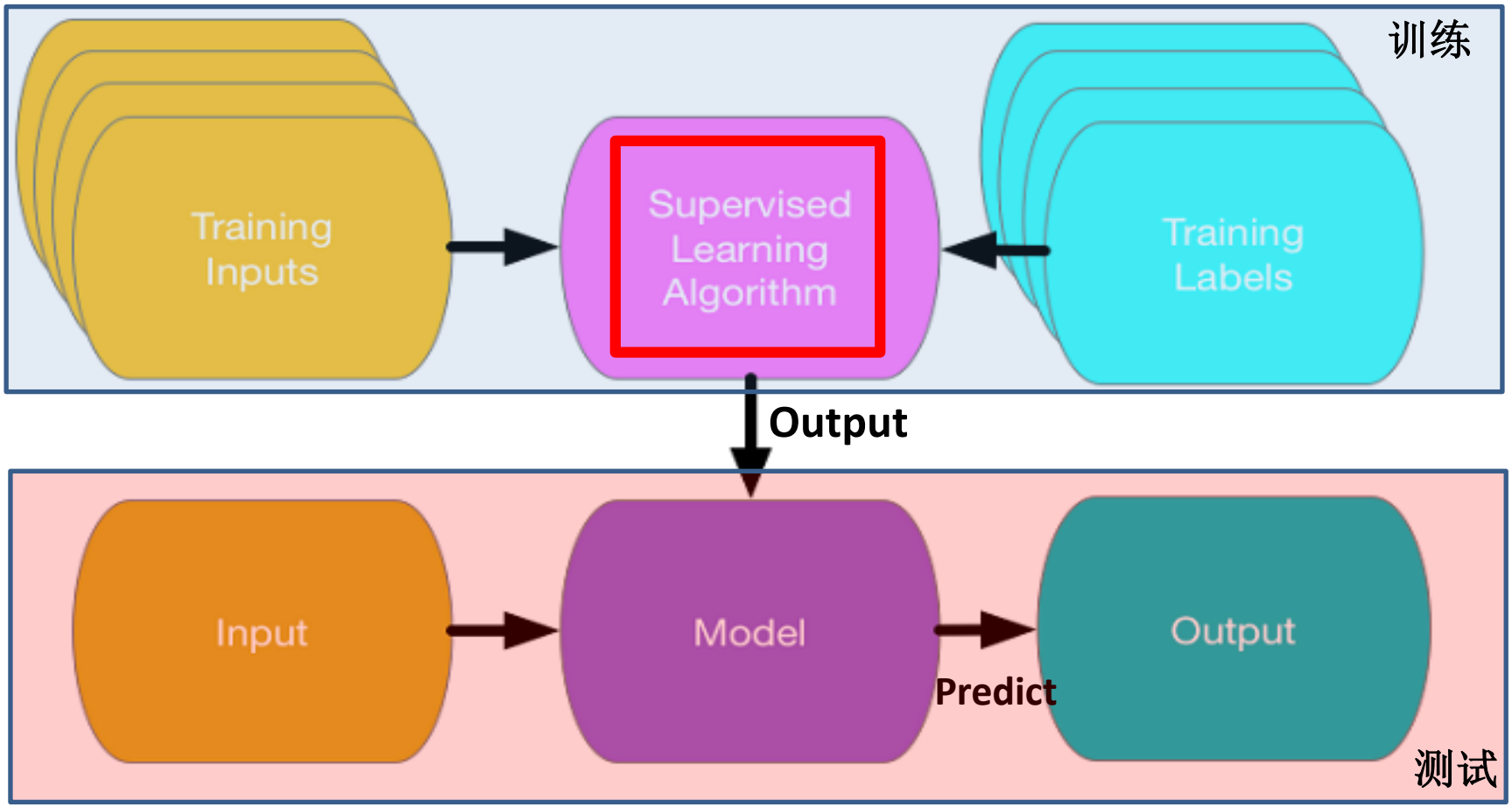
第3章 训练模型

参考资料



教材代码： https://github.com/ageron/handson-ml


训练
Training Inputs
Supervised Learning Algorithm
Training Labels
Output
Input
Model
Output
Predict
测试

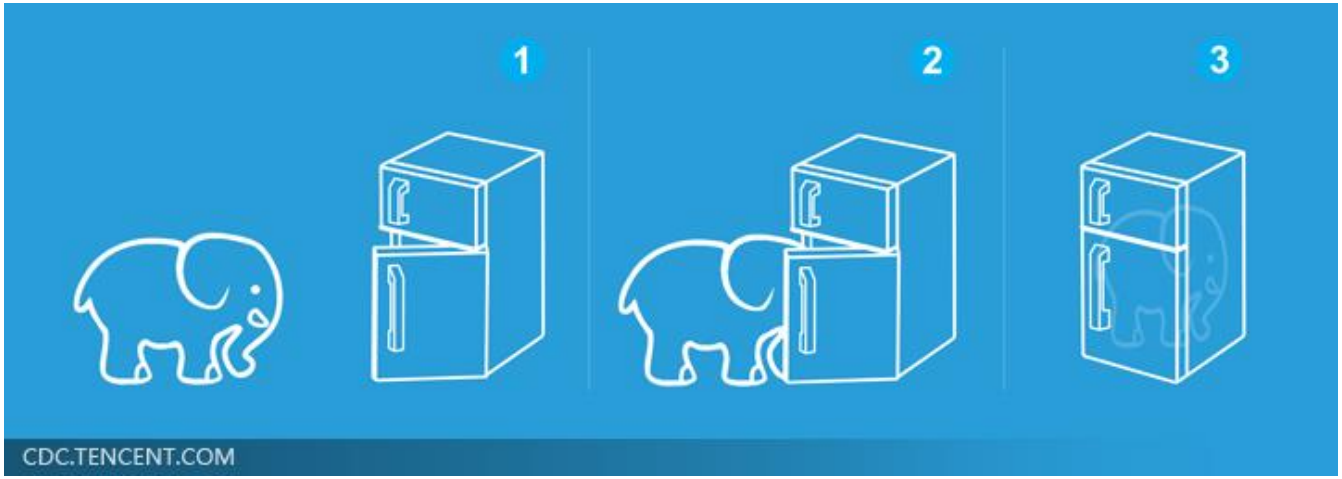
机器学习“三步曲”

Step 1: define a set of function

Step 2: goodness of function (evaluated by Loss Function)

Step 3: pick the best function

就好像把大象放进冰箱 ……



本章内容

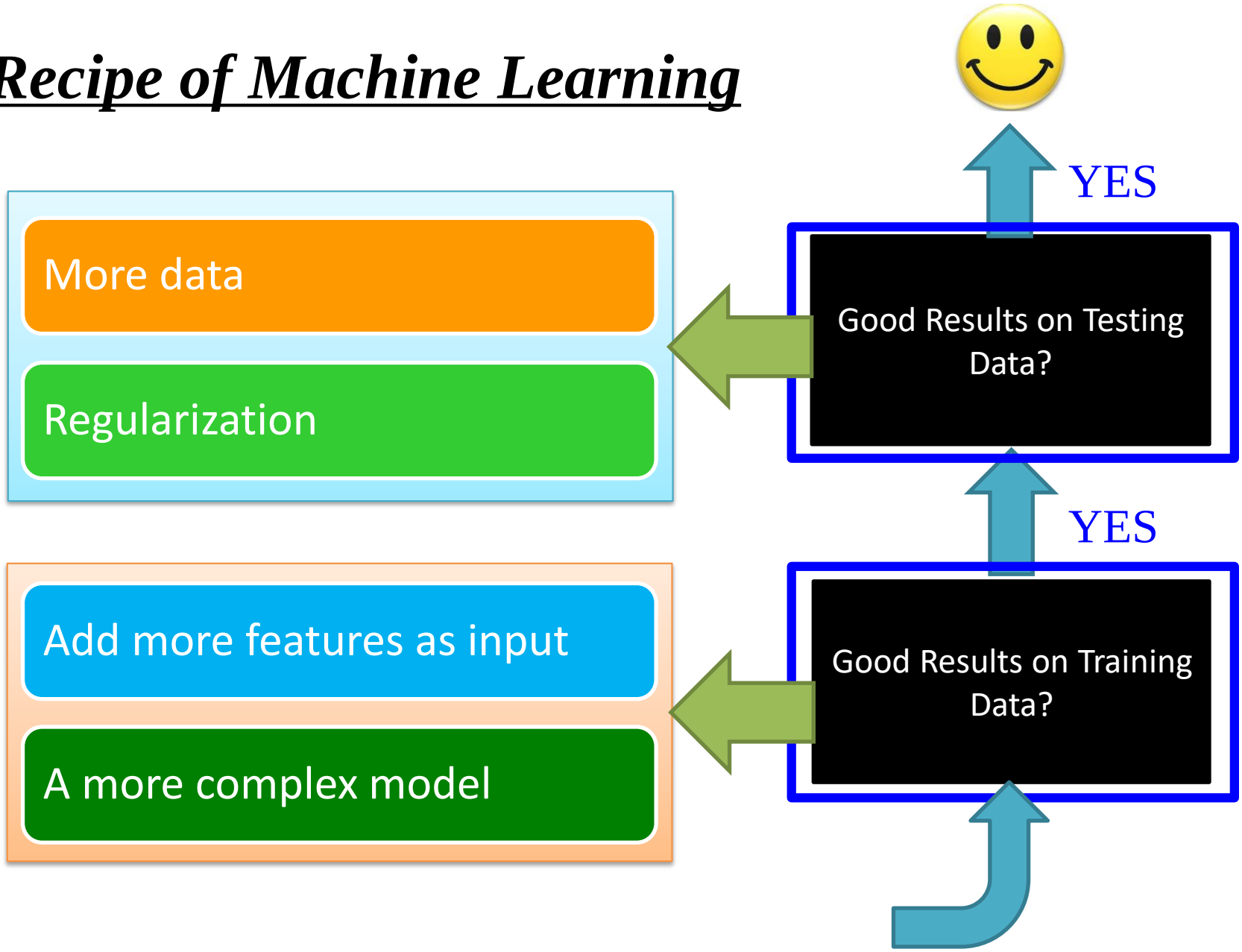
1 如何训练模型：以线性回归为例

- 利用公式，直接使用封闭方程进行求根运算
- 使用迭代优化方法：梯度下降（GD）

2 欠拟合问题和过拟合问题

- 如何判断？利用学习曲线
- 如何解决？

Recipe of Machine Learning



1 如何训练模型？ 即，找到给定模型的最佳函数

- 以一个简单的线性回归模型为例
- 线性回归模型定义：
  - 标量形式

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

  - $\hat{y}$ 表示预测结果
  - n 表示特征的个数
  - x_i 表示第 `i` 个特征的值
  - θ_j 表示第 `j` 个参数（包括偏置项 θ_0 和特征权重值 $\theta_1, \theta_2, \ldots, \theta_n$）

  - 向量形式

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

  - θ 表示模型的参数向量包括偏置项 θ_0 和特征权重值 θ_1 到 θ_n
  - θ^T 表示向量θ的转置（行向量变为了列向量）
  - $\mathbf{x}$ 为每个样本中特征值的向量形式，包括 x_1 到 x_n，而且 x_0 恒为 1
  - $\theta^T \cdot \mathbf{x}$ 表示 θ^T 和$\mathbf{x}$ 的点积
  - h_θ 表示参数为 θ 的假设函数

- 训练模型的目标：找到给定模型的最佳函数
  - 本章中以线性回归模型为例
- 如何衡量函数是“最佳”的？利用损失函数L
  - 回归任务中，常用RMSE来衡量性能

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(\mathbf{x}^{(i)}) - y^{(i)})^2}$$

  - 实践过程中，更多的是用MSE

$$MSE(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

这两个评价函数取得最小值时，对应的参数θ的取值是相同的

- 如何训练模型，找到给定模型的最佳函数？
  - 利用公式，直接使用封闭方程进行求根运算
  - 使用迭代优化方法：梯度下降（GD）

1 如何训练模型？——利用公式求解

- 直接利用公式求解，找到最小化损失函数的**θ值**
  - 通过解正态方程

公式 4-4：正态方程

$$\hat{\theta} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

- $\hat{\theta}$ 指最小化损失 θ 的值
- $\mathbf{y}$ 是一个向量，其包含了 $y^{(1)}$ 到 $y^{(m)}$ 的值

  - 验证**θ值的正确性**

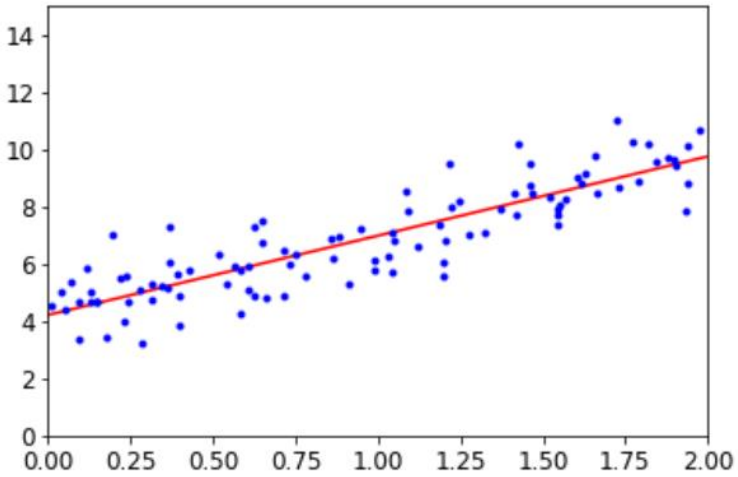
- 如何验证**θ值的正确性？**
  - 首先，生成一些近似线性的数据

```
import numpy as np
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```

生产数据的函数实际上是： **y= 4+3*x +高斯噪声**

  - 利用正态方程来计算最佳参数 $\hat{\theta}$

```
X_b = np.c_[np.ones((100, 1)), X]
theta_best = np.linalg.inv(X_b.T.dot(X_B)).dot(X_b.T).dot(y)

>>> theta_best
array([[4.21509616],[2.77011339]])
```



1. 对比发现：计算出来的最佳参数，非常接近原来的参数
2. 由于存在噪声，计算得到的最佳参数不可能达到原来的值

- 如何验证**θ值的正确性？**
  - 首先，生成一些近似线性的数据

```
import numpy as np
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```

生产数据的函数实际上是： **y= 4+3*x +随机噪声**

  - 利用正态方程来计算最佳参数 $\hat{\theta}$

```
X_b = np.c_[np.ones((100, 1)), X]
theta_best = np.linalg.inv(X_b.T.dot(X_B)).dot(X_b.T).dot(y)
```

```
>>> theta_best
array([[4.21509616],[2.77011339]])
```

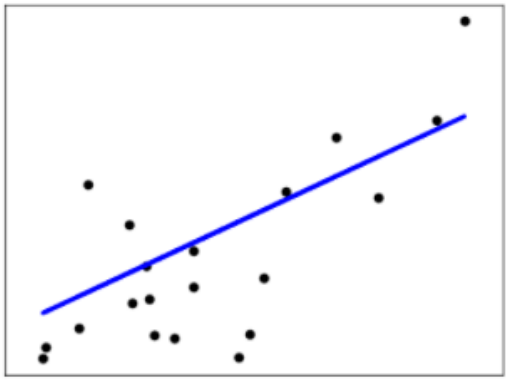
  - 使用Scikit-Learn代码可以达到相同的效果：

```
>>> form sklearn.linear_model import LinearRegression
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X,y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([4.21509616]),array([2.77011339]))
>>> lin_reg.predict(X_new)
array([[4.21509616],[9.75532293]])
```

1.1.1. Ordinary Least Squares

`LinearRegression` fits a linear model with coefficients $w = (w_1, \ldots, w_p)$ to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_w ||Xw - y||_2^2$$



`LinearRegression` will take in its `fit` method arrays X, y and will store the coefficients w of the linear model in its `coef_` member:

```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
...
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                 normalize=False)
>>> reg.coef_
array([0.5, 0.5])
```

是否需要先进行归一化

https://scikit-learn.org/stable/modules/linear_model.html#ordinary-least-squares

1. Supervised learning

1.1. Generalized Linear Models

- 什么时候利用正态方程求解线性回归模型的参数更合适？
  - 劣势：特征的个数较大的时候（例如：特征数量为100000），正态方程求解将会非常慢。

正态方程需要计算矩阵 $\mathbf{X}^T \cdot \mathbf{X}$ 的逆，它是一个 $n * n$ 的矩阵（n 是特征的个数）

矩阵求逆的运算复杂度大约在 $O(n^{2.4})$ 到 $O(n^3)$ 之间

  - 优势：只要内存空间足够大，可以对大规模数据进行训练
    - 在训练集上对于每一个实例来说是线性的，其复杂度为O(m)，此处m表示训练样本数量

1 如何训练模型？——利用梯度下降

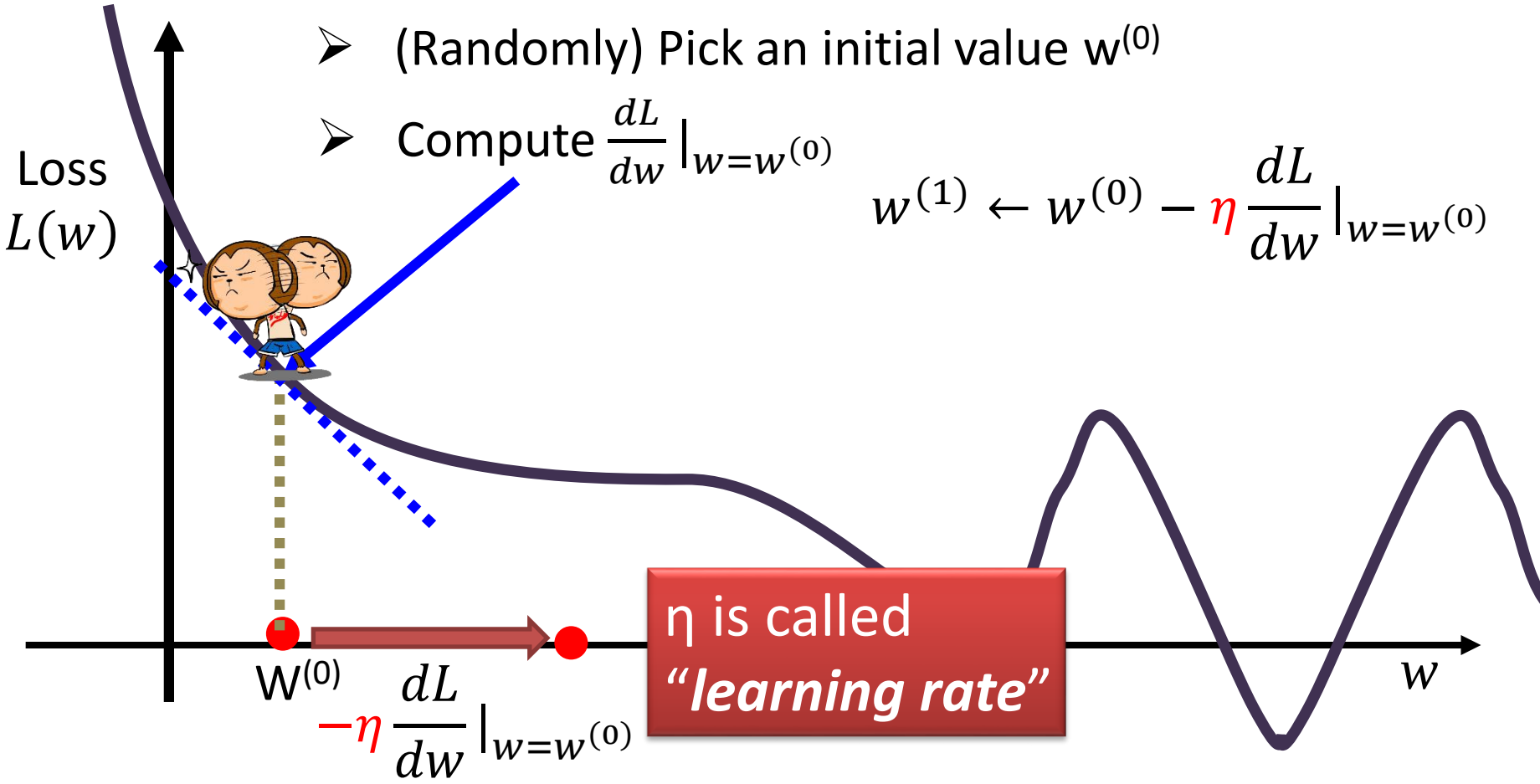
- 梯度下降(Gradient Descent)是一种非常通用的优化算法，适合在特征个数非常多，训练实例非常多，内存无法满足要求的时候使用。
- 如何利用梯度下降来更新参数，以便找到最佳参数？
- 梯度下降的Tips

http://chico386.pixnet.net/album/photo/171572850

Gradient Descent

$$w^* = arg\min_w L(w)$$

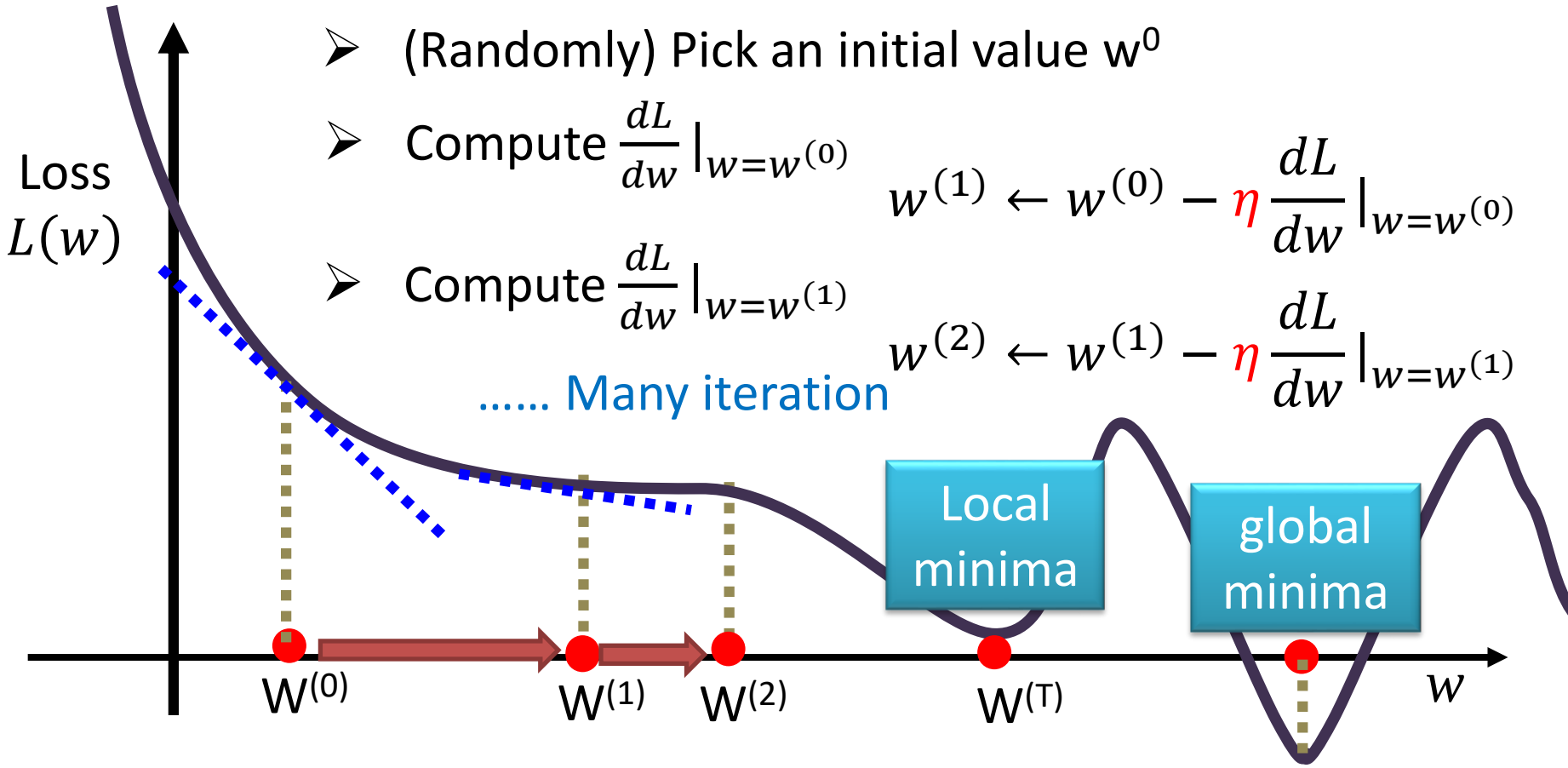
- Consider loss function $L(w)$ with one parameter w:



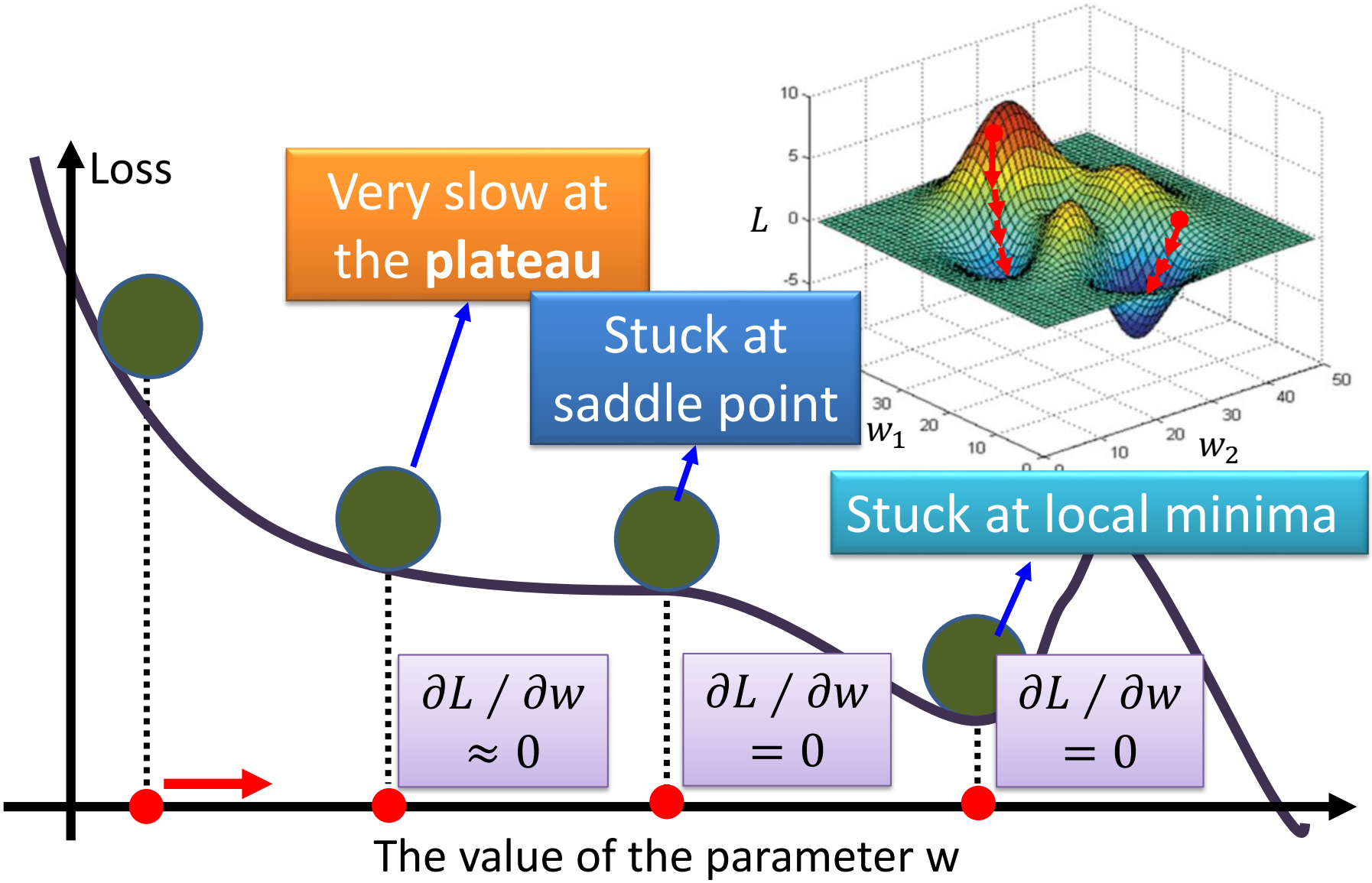
Gradient Descent

- Consider loss function $L(w)$ with one parameter w:

$$w^* = arg\min_w L(w)$$



Gradient Descent



Gradient Descent

$$\begin{bmatrix} \frac{\partial L}{\partial w} \\ \frac{\partial L}{\partial b} \end{bmatrix}$$ gradient

- How about two parameters?

$$w^*, b^* = arg \min_{w,b} L(w, b)$$

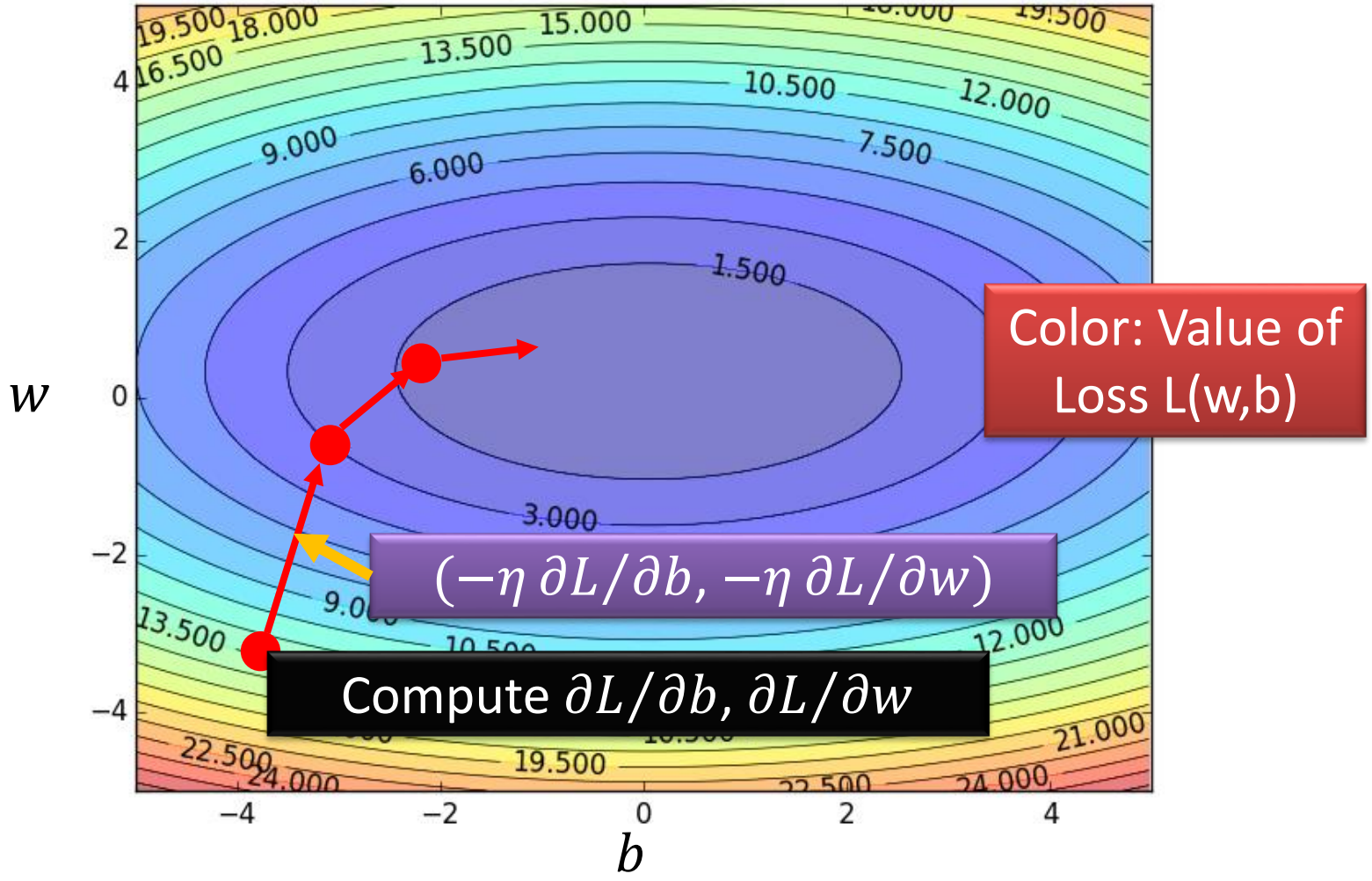
  - (Randomly) Pick an initial value $w^{(0)}$, $b^{(0)}$
  - Compute $\frac{\partial L}{\partial w}|_{w=w^{(0)},b=b^{(0)}}, \frac{\partial L}{\partial b}|_{w=w^{(0)},b=b^{(0)}}$

$$w^{(1)} \leftarrow w^{(0)} - \eta \frac{\partial L}{\partial w}|_{w=w^{(0)},b=b^{(0)}} \qquad b^{(1)} \leftarrow b^{(0)} - \eta \frac{\partial L}{\partial b}|_{w=w^{(0)},b=b^{(0)}}$$

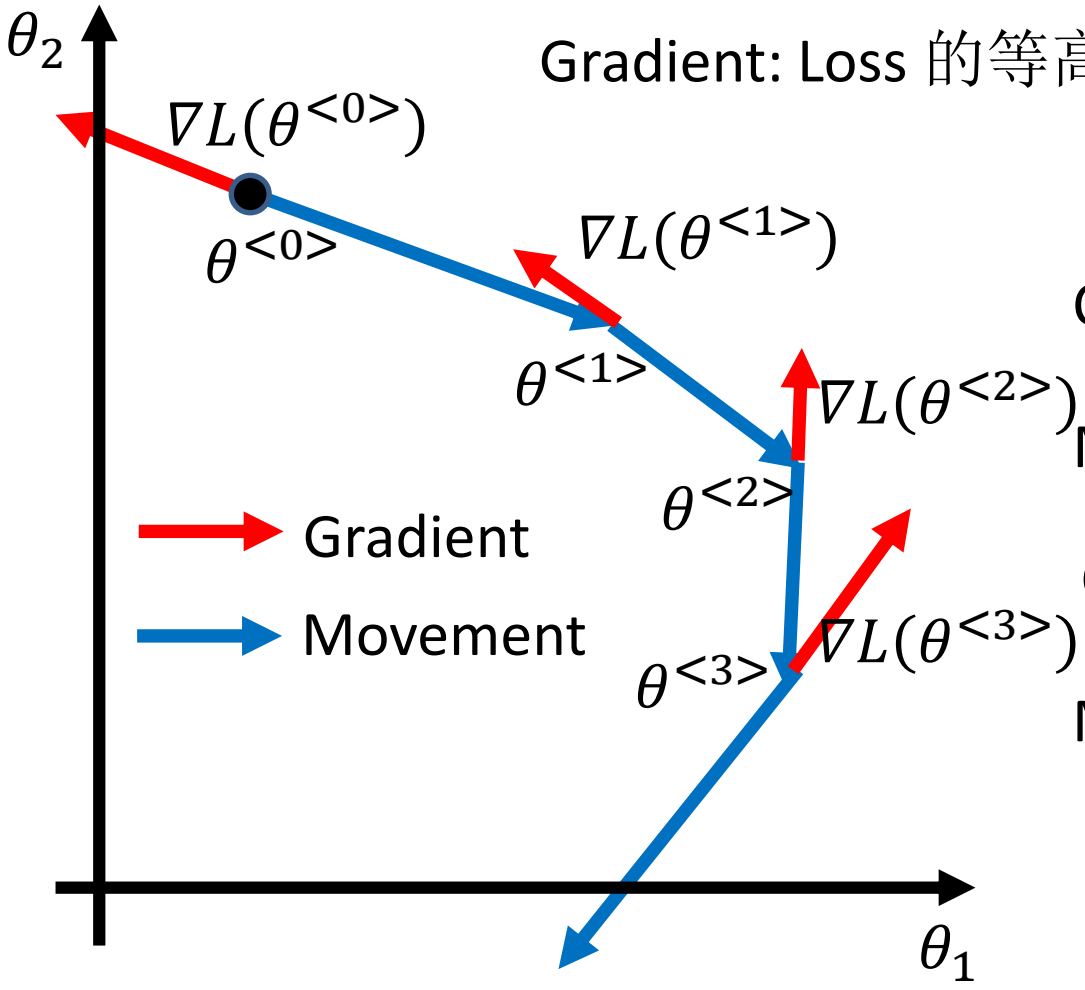
  - Compute $\frac{\partial L}{\partial w}|_{w=w^{(1)},b=b^{(1)}}, \frac{\partial L}{\partial b}|_{w=w^{(1)},b=b^{(1)}}$

$$w^{(2)} \leftarrow w^{(1)} - \eta \frac{\partial L}{\partial w}|_{w=w^{(1)},b=b^{(1)}} \qquad b^{(2)} \leftarrow b^{(1)} - \eta \frac{\partial L}{\partial b}|_{w=w^{(1)},b=b^{(1)}}$$

Gradient Descent



Gradient Descent



Gradient: Loss 的等高线的法线方向

Start at position $\theta^{<0>}$

Compute gradient at $\theta^{<0>}$

Move to $\theta^{<1>} = \theta^{<0>} - \eta\nabla L(\theta^{<0>})$

Compute gradient at $\theta^{<1>}$

Move to $\theta^{<2>} = \theta^{<1>} - \eta\nabla L(\theta^{<1>})$

⋮

Gradient Descent

- Formulation of $\partial L/\partial w$ and $\partial L/\partial b$

$$L(w,b) = \frac{1}{7}\sum_{i=1}^{7}\left(\hat{y}^{(i)} - \left(b + w \cdot x_{cp}^{(i)}\right)\right)^2$$

$$\frac{\partial L}{\partial w} =? \frac{1}{7}\sum_{i=1}^{7} 2\left(\hat{y}^{(i)} - \left(b + w \cdot x_{cp}^{(i)}\right)\right)$$

$$\frac{\partial L}{\partial b} =?$$

Gradient Descent

- Formulation of $\partial L/\partial w$ and $\partial L/\partial b$

$$L(w,b) = \frac{1}{7}\sum_{i=1}^{7}\left(\hat{y}^{(i)} - \left(b + w \cdot x_{cp}^{(i)}\right)\right)^2$$

$$\frac{\partial L}{\partial w} =? \frac{1}{7}\sum_{i=1}^{7} 2\left(\hat{y}^{(i)} - \left(b + w \cdot x_{cp}^{(i)}\right)\right)\left(-x_{cp}^{(i)}\right)$$

$$\frac{\partial L}{\partial b} =? \frac{1}{7}\sum_{i=1}^{7} 2\left(\hat{y}^{(i)} - \left(b + w \cdot x_{cp}^{(i)}\right)\right)$$

Gradient Descent

- When solving:

$$\theta^* = \arg\min_{\theta} L(\theta)$$ by gradient descent

- Each time we update the parameters, we obtain θ that makes $L(\theta)$ smaller.

$$L\left(\theta^{(0)}\right) > L\left(\theta^{(1)}\right) > L\left(\theta^{(2)}\right) > \cdots$$

Is this statement correct?

NO !

$$\theta^{<i+1>} = \theta^{<i>} - \eta \nabla L\left(\theta^{<i>}\right)$$

- 影响梯度下降的优化结果的因素有：
  - 学习率Learning Rate
    - 过大？过小？正合适？
  - 特征是否经过缩放
    - 影响训练速度
  - 梯度下降的三类变体
    - BGD, SGD, MBGD

Gradient Descent Tips

- Tip 1: Tuning your learning rates

- Tip 2: Feature Scaling/Normalization

Make the training faster

- Tip 3: Variants of Gradient Descent
  - Stochastic Gradient Descent, Mini-batch Gradient Descent

Gradient Descent

$$\theta^{<i+1>} = \theta^{<i>} - \boldsymbol{\eta}\nabla L\left(\theta^{<i>}\right)$$

$$\nabla L(\theta) = \begin{bmatrix} \partial L(\theta_1)/\partial\theta_1 \\ \partial L(\theta_2)/\partial\theta_2 \end{bmatrix}$$

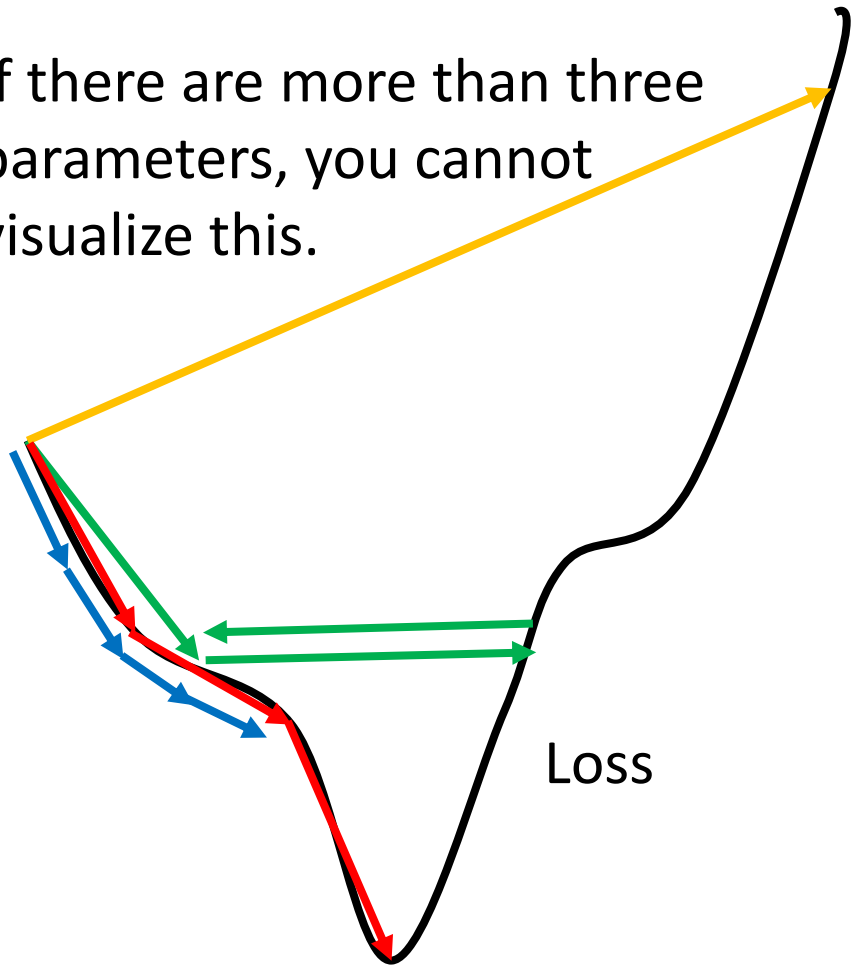
Tip 1: Tuning your learning rates $\boldsymbol{\eta}$

Learning Rate

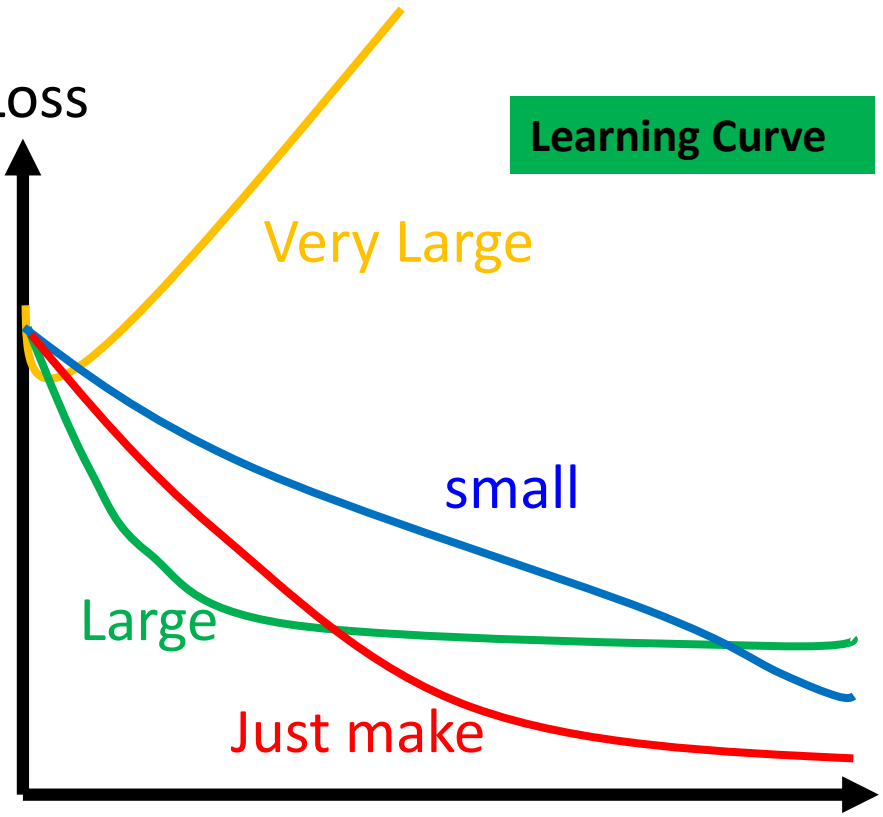
$$\theta^{<i>} = \theta^{<i-1>} - \eta \nabla L\left(\theta^{<i-1>}\right)$$

Set the learning rate η carefully

If there are more than three parameters, you cannot visualize this.



Loss Surface



But you can always visualize this.

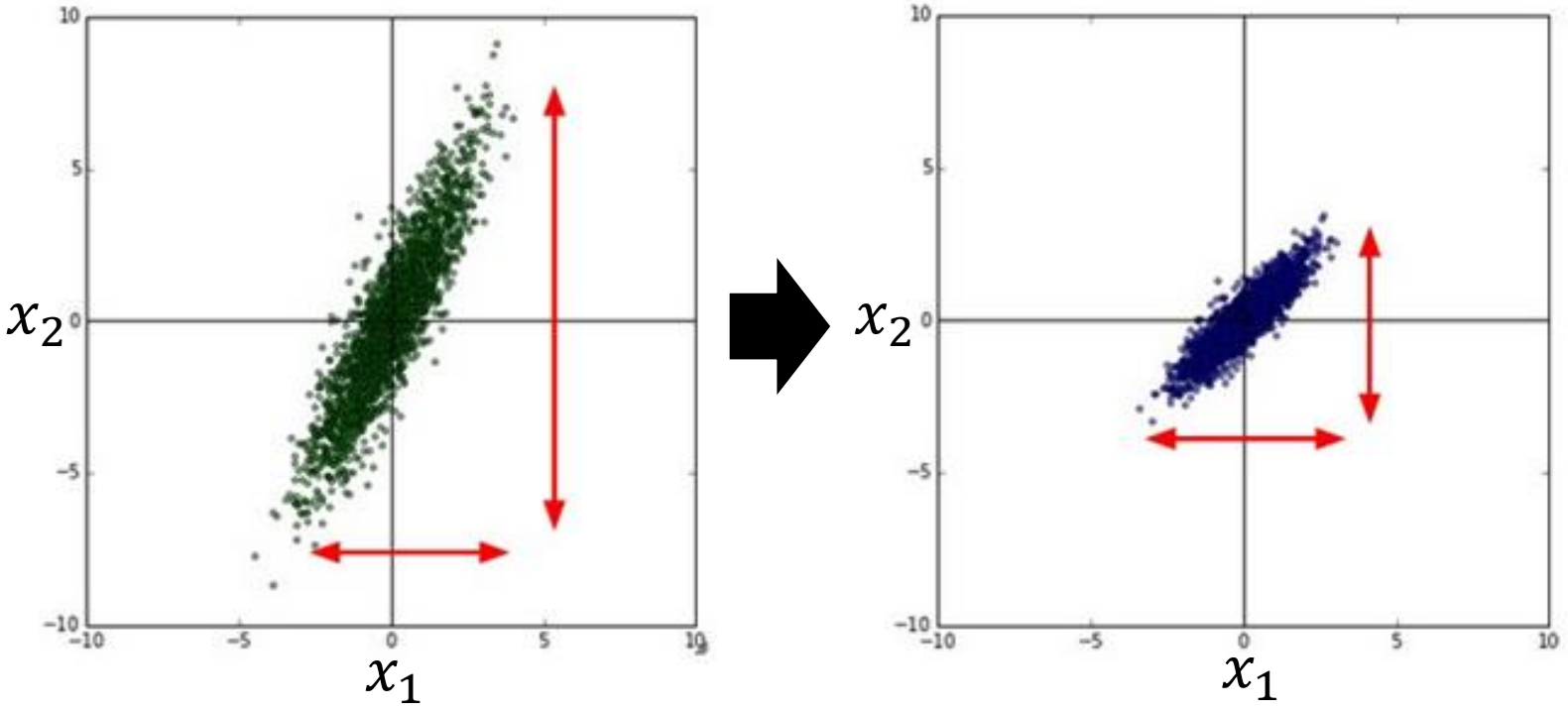
Gradient Descent

Tip 2: Feature Scaling

Feature Scaling

Source of figure: http://cs231n.github.io/neural-networks-2/

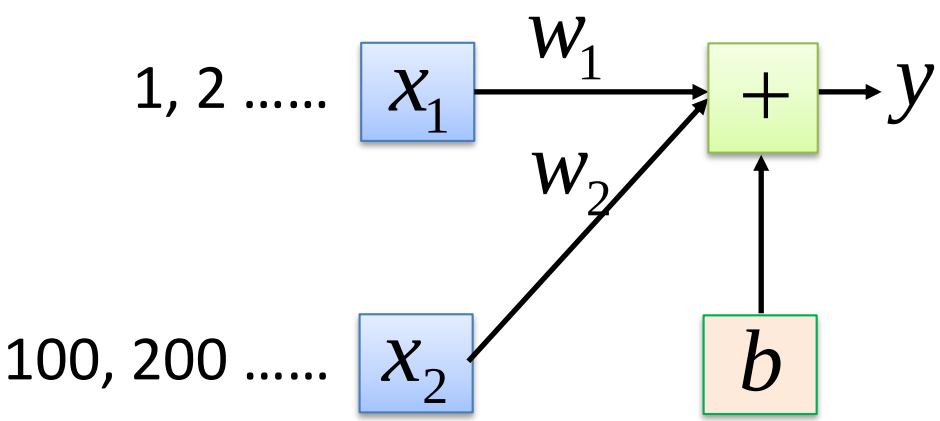
$$y = b + w_1 x_1 + w_2 x_2$$

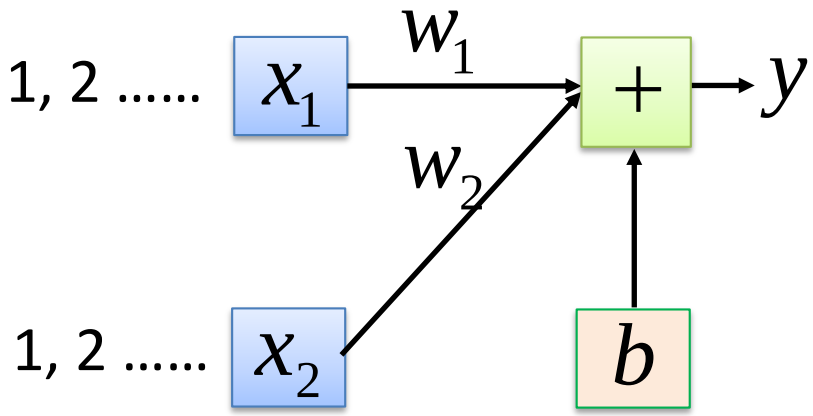


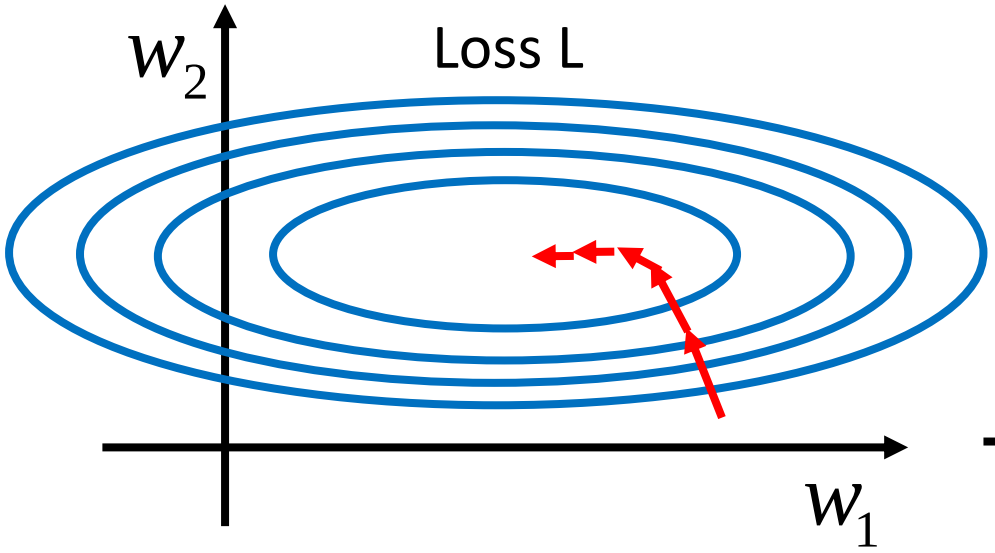
Make different features have the same scaling

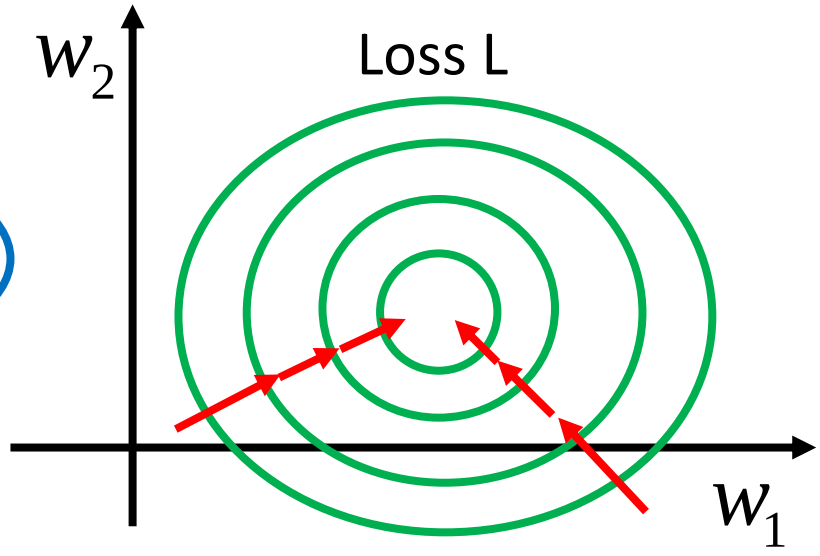
Feature Scaling

$$y = b + w_1 x_1 + w_2 x_2$$









Gradient Descent

Tip 3: Variants of Gradient Descent： BGD, SGD, MBGD

Gradient Descent

◆ ***Gradient Descent***

Batch Gradient Descent (BGD)

Vanilla gradient descent

$$\theta^{<i>} = \theta^{<i-1>} - \eta \nabla L\left(\theta^{<i-1>}\right)$$

$$L = \sum_j \left(\hat{y}^{(j)} - \left(b + \sum w_i x_i^{(j)} \right) \right)^2$$

BGD: Loss is the summation over all training examples

◆ ***Stochastic Gradient Descent*** Faster!

Pick an example $x^{(j)}$

$$L^{(j)} = \left(\hat{y}^{(j)} - \left(b + \sum w_i x_i^{(j)} \right) \right)^2 \qquad \theta^{<i>} = \theta^{<i-1>} - \eta \nabla L^{(j)}\left(\theta^{<i-1>}\right)$$

Loss for only one example

Batch Gradient Descent (BGD)

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

}

gradient descent. Note that, while gradient descent can be susceptible to local minima in general, the optimization problem we have posed here for linear regression has only one global, and no other local, optima; thus gradient descent always converges (assuming the learning rate α is not too large) to the global minimum. Indeed, J is a convex quadratic function.

```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(loss_function, data, params)
  params = params - learning_rate * params_grad
```

Stochastic Gradient Descent (SGD)

Loop {

for i=1 to m, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

}

}

This algorithm is called **stochastic gradient descent** (also **incremental gradient descent**). Whereas batch gradient descent has to scan through the entire training set before taking a single step—a costly operation if m is large—stochastic gradient descent can start making progress right away, and continues to make progress with each example it looks at. Often, stochastic gradient descent gets θ "close" to the minimum much faster than batch gradient descent. (Note however that it may never "converge" to the minimum, and the parameters θ will keep oscillating around the minimum of $J(\theta)$; but in practice most of the values near the minimum will be reasonably good approximations to the true minimum.[2]) For these reasons, particularly when the training set is large, stochastic gradient descent is often preferred over batch gradient descent.

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function, example, params)
    params = params - learning_rate * params_grad
```

Stochastic Gradient Descent (SGD)

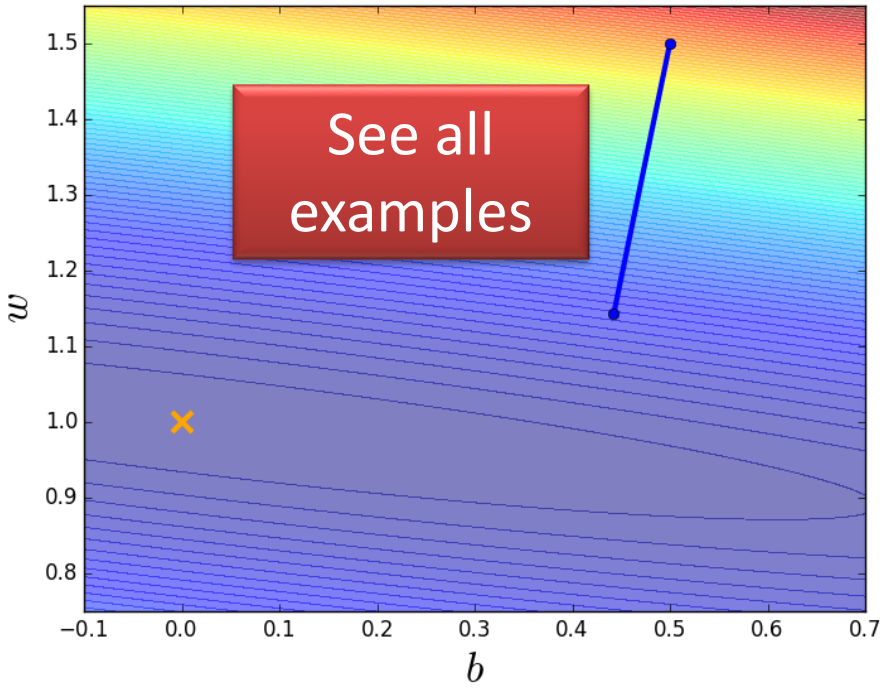
Gradient Descent

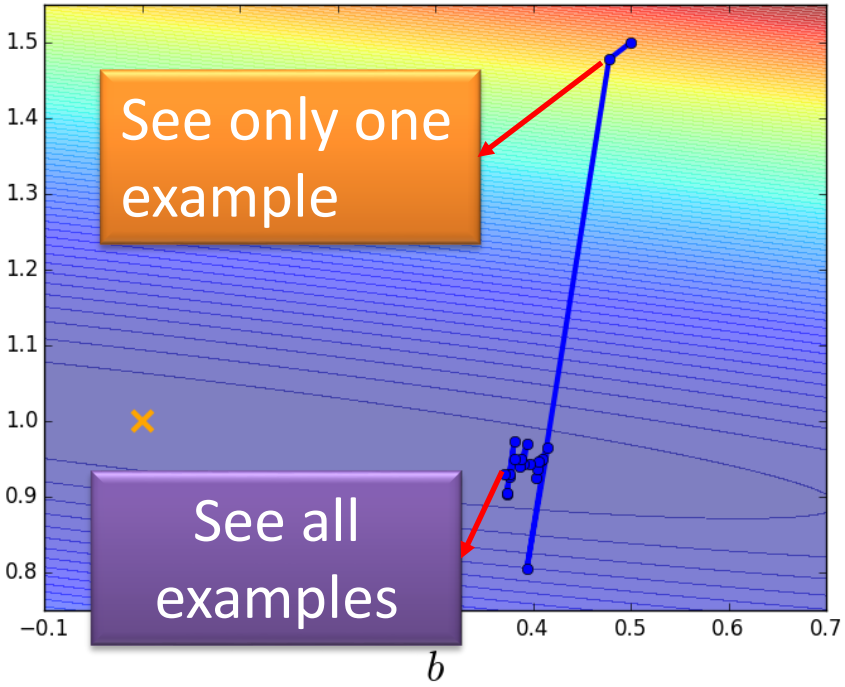
Update after seeing all examples

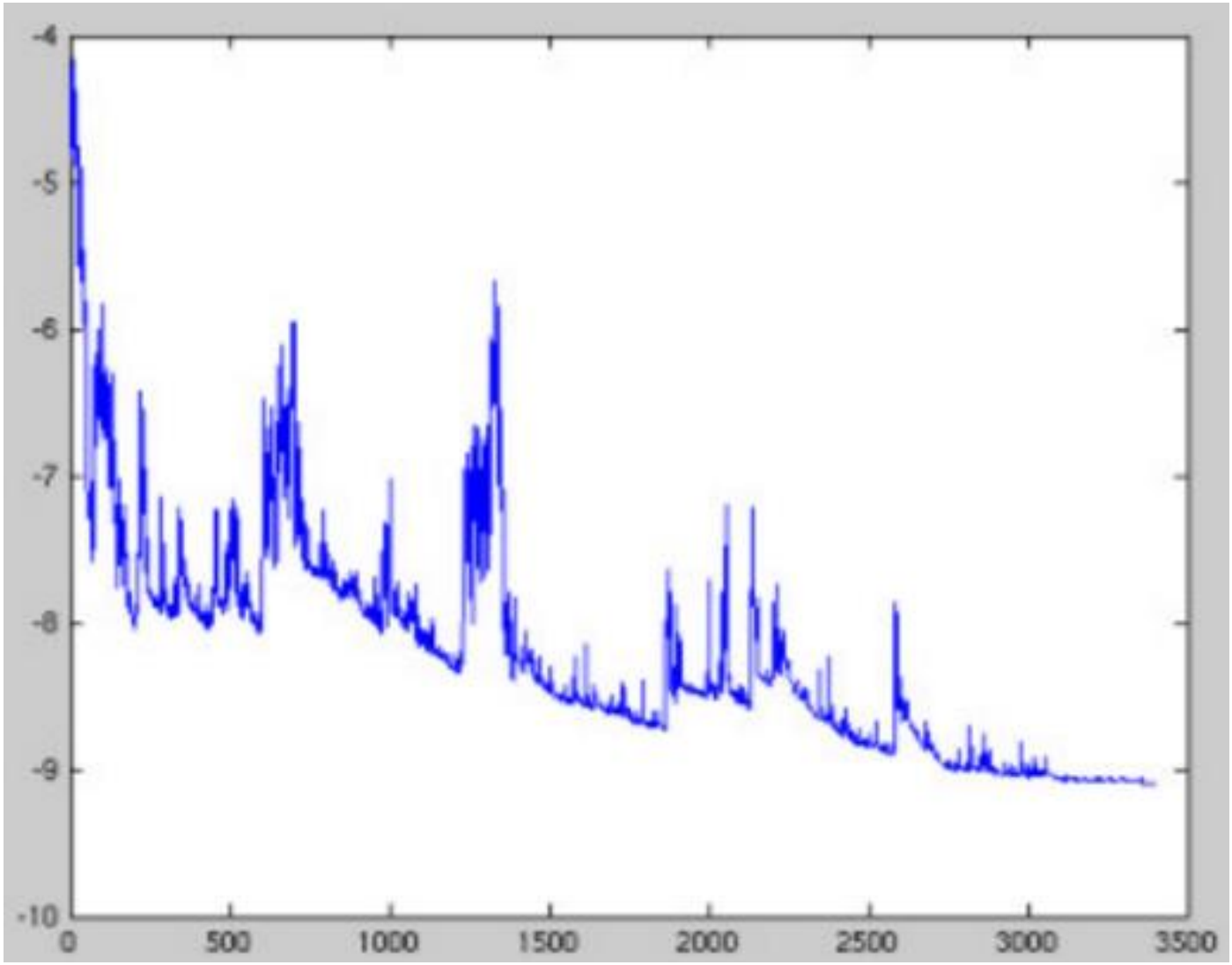
Stochastic Gradient Descent

Update for each example

If there are 20 examples, 20 times faster.






-4
-5
-6
-7
-8
-9
-10
0
500
1000
1500
2000
2500
3000
3500

Mini-batch Gradient Descent (MBGD)

- Mini-batch GD(MBGD)
  - tradeoff between SGD and GD
  - BatchSize: 1 ~ #(training data)
  - Update the parameters once using BatchSize samples

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data, batch_size=50):
    params_grad = evaluate_gradient(loss_function, batch, params)
    params = params - learning_rate * params_grad
```

Review: BGD, MBGD, SGD

- SGD
  - Update the parameters once using one sample
- BGD
  - Update the parameters once using all the training samples
- Mini-batch GD(MBGD)
  - tradeoff between SGD and GD
  - BatchSize: 1 ~ number of training data
  - Update the parameters once using a part of the training samples dependent to BatchSize

Note: Training Error are computed on the entire training data to plot the loss curve.

- Epoch:
  - 1 epoch means that all the samples in the set are used once.
- Iteration:
  - #(Batch)
- Batch-size:
  - The number of data in one iteration/batch

示例：梯度下降的三种变体：BGD,SGD,MBGD

- 首先，生成一些近似线性的数据

```
import numpy as np
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```

生产数据的函数实际上是： y= 4+3*x +随机噪声

- 利用正态方程来计算最佳参数 $\hat{\theta}$

```
X_b = np.c_[np.ones((100, 1)), X]
theta_best = np.linalg.inv(X_b.T.dot(X_B)).dot(X_b.T).dot(y)
```

```
>>> theta_best
array([[4.21509616],[2.77011339]])
```

- 达到相同的效果：

```
>>> form sklearn.linear_model import LinearRegression
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X,y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([4.21509616]),array([2.77011339]))
>>> lin_reg.predict(X_new)
array([[4.21509616],[9.75532293]])
```

利用BGD来计算最佳参数 $\hat{\theta}$

```
>>> theta_best
array([[4.21509616],[2.77011339]])
```

公式 4-7：梯度下降步长

$$\theta^{(next\ step)} = \theta - \eta \nabla_\theta MSE(\theta)$$

$$\nabla_\theta MSE(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} MSE(\theta) \\ \frac{\partial}{\partial \theta_1} MSE(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} MSE(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - y)$$

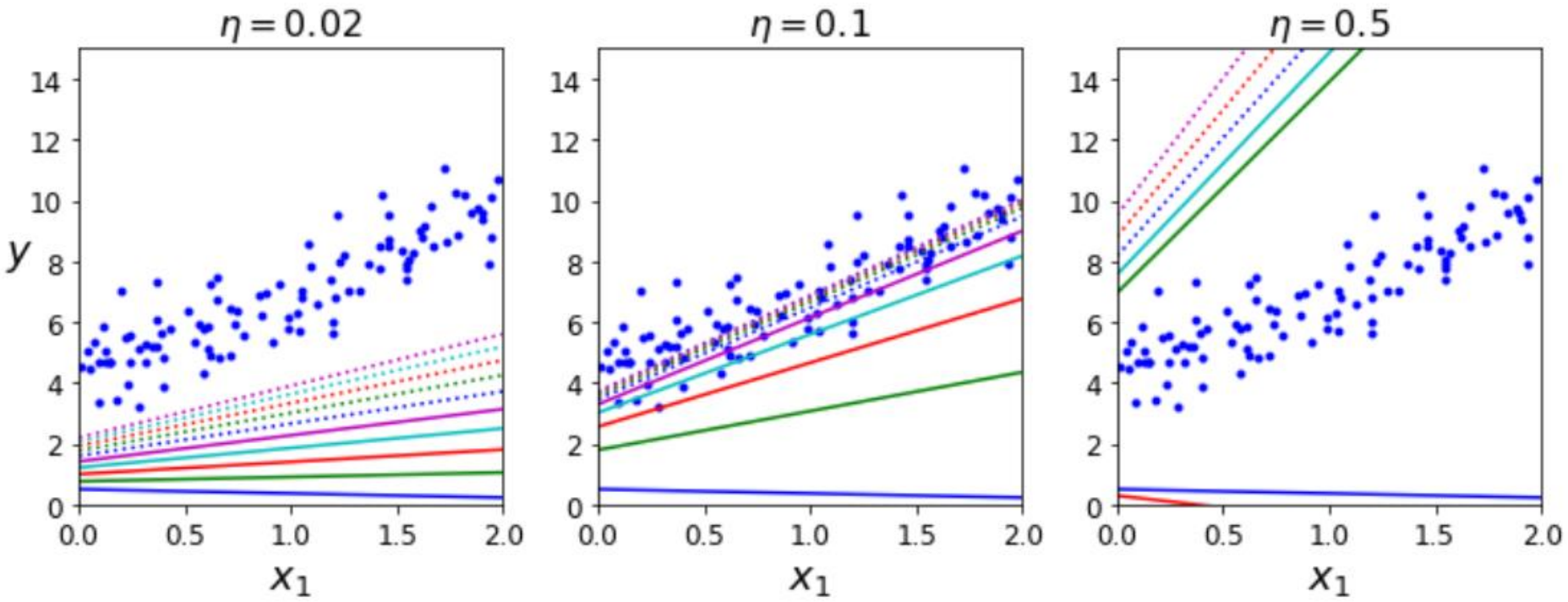
```
eta = 0.1 # 学习率
n_iterations = 1000
m = 100

theta = np.random.randn(2,1) # 随机初始值

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

```
>>> theta
array([[4.21509616],[2.77011339]])
```

超参数： eta, n_iterations

```
if iteration == 0:    style = "b-"
if iteration == 1:    style = "g-"
if iteration == 2:    style = "r-"
if iteration == 3:    style = "c-"
if iteration == 4:    style = "m-"
if iteration == 5:     style = "b:"
if iteration == 6:     style = "g:"
if iteration == 7:     style = "r:"
if iteration == 8:     style = "c:"
if iteration == 9:      style = "m:"
```

- 不同学习率时，利用BGD在不同epoch时得到的参数所对应的函数的拟合情况



利用SGD来计算最佳参数 $\hat{\theta}$

```
>>> theta_best
array([[4.21509616],[2.77011339]])
```

```
n_epochs = 50
t0, t1 = 5, 50  #learning_schedule的超参数
m = 100
def learning_schedule(t):
    return t0 / (t + t1)

theta = np.random.randn(2,1)

for epoch in range(n_epochs):
    for i in range(m):
        random_index = np.random.randint(m)
        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 * gradiens
```

超参数：n_iterations, t0, t1

```
>>> theta
array([[4.21076011],[2.748560791]])
```

利用SGD来计算最佳参数 $\hat{\theta}$

——使用SGDRegressor

```
>>> theta_best
array([[4.21509616],[2.77011339]])
```

```
from sklearn.linear_model import SGDRegressor
sgd_reg = SGDRegressor(n_iter=50, penalty=None, eta0=0.1)
sgd_reg.fit(X, y.ravel())
```

没有使用正则化

```
>>> sgd_reg.intercept_, sgd_reg.coef_
(array([4.18380366]),array([2.74205299]))
```

超参数：eta, n_iterations, penalty

```
>>> theta
array([[4.21076011],[2.748560791]])
```

1. Supervised learning

1.1. Generalized Linear Models

1. Supervised learning

1.1. Generalized Linear Models

1.1.12. Stochastic Gradient Descent - SGD

Stochastic gradient descent is a simple yet very efficient approach to fit linear models. It is particularly useful when the number of samples (and the number of features) is very large. The `partial_fit` method allows online/out-of-core learning.

The classes **`SGDClassifier`** and **`SGDRegressor`** provide functionality to fit linear models for classification and regression using different (convex) loss functions and different penalties. E.g., with `loss="log"`, **`SGDClassifier`** fits a logistic regression model, while with `loss="hinge"` it fits a linear support vector machine (SVM).

利用MBGD来计算最佳参数 $\hat{\theta}$

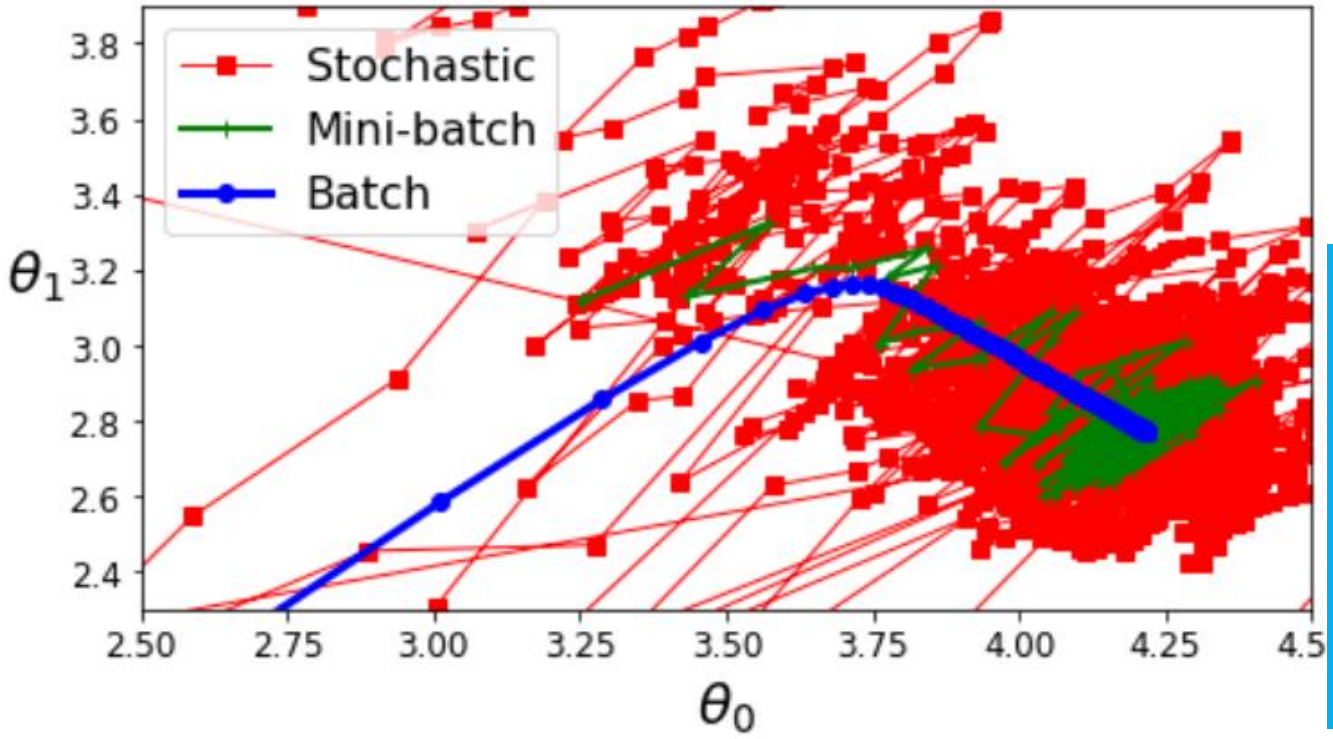
```
n_iterations = 50
minibatch_size = 20
m = 100
np.random.seed(42)
theta = np.random.randn(2,1)  # random initialization

t0, t1 = 200, 1000
def learning_schedule(t):
    return t0 / (t + t1)

t = 0
for epoch in range(n_iterations):
    shuffled_indices = np.random.permutation(m)
    X_b_shuffled = X_b[shuffled_indices]
    y_shuffled = y[shuffled_indices]
    for i in range(0, m, minibatch_size):
        t += 1
        xi = X_b_shuffled[i:i+minibatch_size]
        yi = y_shuffled[i:i+minibatch_size]
        gradients = 2/minibatch_size * xi.T.dot(xi.dot(theta) - yi)
        eta = learning_schedule(t)
        theta = theta - eta * gradients
```

```
>>> theta_best
array([[4.21509616],[2.77011339]])
```

超参数：n_iterations, minibatch_size, t0, t1

```
theta
```

```
array([[4.25214635],
       [2.7896408 ]])
```



结论：

1. BGD的路径最后停在了最小值，而SGD和MBGD最后都在最小值附近摆动
2. SGD和MBGD的学习率须动态变化，学习率的计算须精心设计

图4-11：参数空间的梯度下降路径

表 4-1：比较线性回归的不同梯度下降算法

Algorithm	Large *m*	Out-of-core support	Large *n*	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	Fast	No	Slow	0	No	`LinearRegression`
Batch GD	Slow	No	Fast	2	Yes	n/a
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	n/a

Out-of-core（核外学习）：对于超大数据集，因内存限制，故每次只能读入加载部分数据到内存中进行训练，然后不断重复这个过程，直到完成所有数据的训练。

总结：如何训练模型

- 直接使用封闭方程进行求根运算
  - 不是所有的模型都能使用封闭方程来计算最佳参数
  - 标准方程仅适用于线性回归
- 使用迭代优化方法：梯度下降（GD）
  - 更通用，可以用于训练其他模型

本章内容

1 如何训练模型：以线性回归为例

- 利用公式，直接使用封闭方程进行求根运算
- 使用迭代优化方法：梯度下降（GD）

2 欠拟合问题和过拟合问题

- 如何判断？利用学习曲线
- 如何解决？

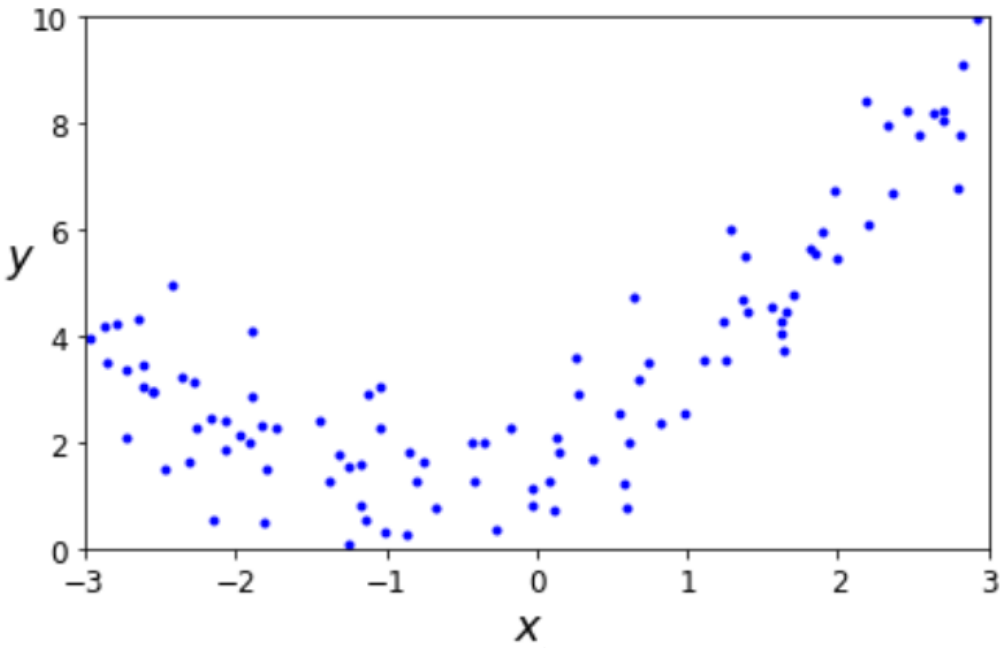
2 欠拟合问题和过拟合问题

- 引入更多特征来增加模型复杂度，从而引发 欠拟合问题

- 利用学习曲线来判断训练结果
- 解决过拟合
  - 利用正则化技术
  - 早期终止法

- 首先，生成一些非线性数据：

```
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)
```



$y = 0.5 * x^2 + x + 2$ +随机噪声

显然，使用 $y = \theta_1 * x + \theta_0$ 这类直线不能恰当的拟合这些数据

- 尝试：使用$y = \theta_2 * x^2 + \theta_1 * x + \theta_0$多项式来拟合
  - $y = \theta_2 * x^2 + \theta_1 * x + \theta_0 = \theta_2 * x_2 + \theta_1 * x_1 + \theta_0$
  - 使用Scikit-Learning 的PolynomialFeatures类 增加新特征，来扩展训练集

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly_features = PolynomialFeatures(degree=2,include_bias=False)
>>> X_poly = poly_features.fit_transform(X)
>>> X[0]
array([-0.75275929])
>>> X_poly[0]
array([-0.75275929, 0.56664654])
```

可视为超参数

$0.56664654 = (-0.75275929)^2$

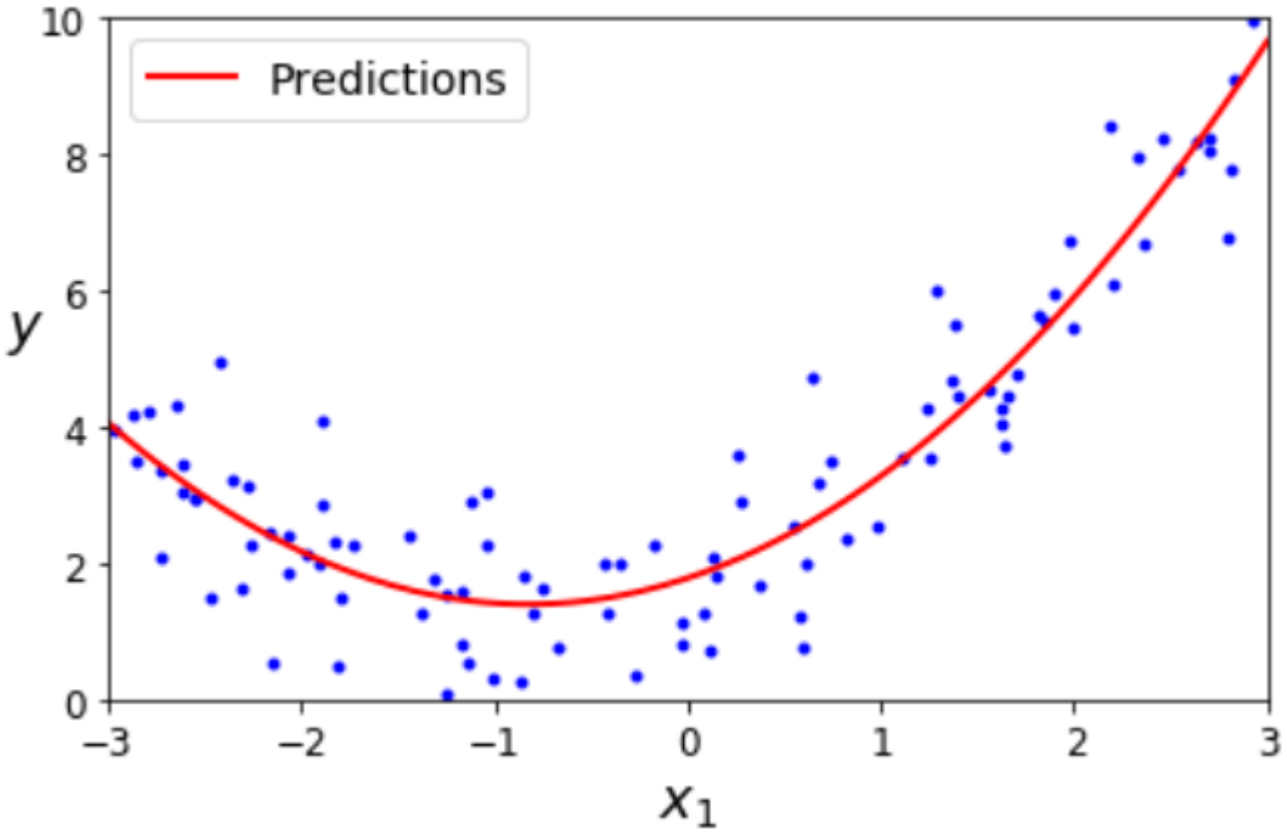
  - 使用LinearRegression模型进行拟合

```
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X_poly, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([ 1.78134581]), array([[ 0.93366893, 0.56456263]]))
```

预估的函数： $\hat{y} = 0.56x_1^2 + 0.93x_1 + 1.78$

真实的函数： **y = 0.5* x² + x + 2 +随机噪声**

比较接近



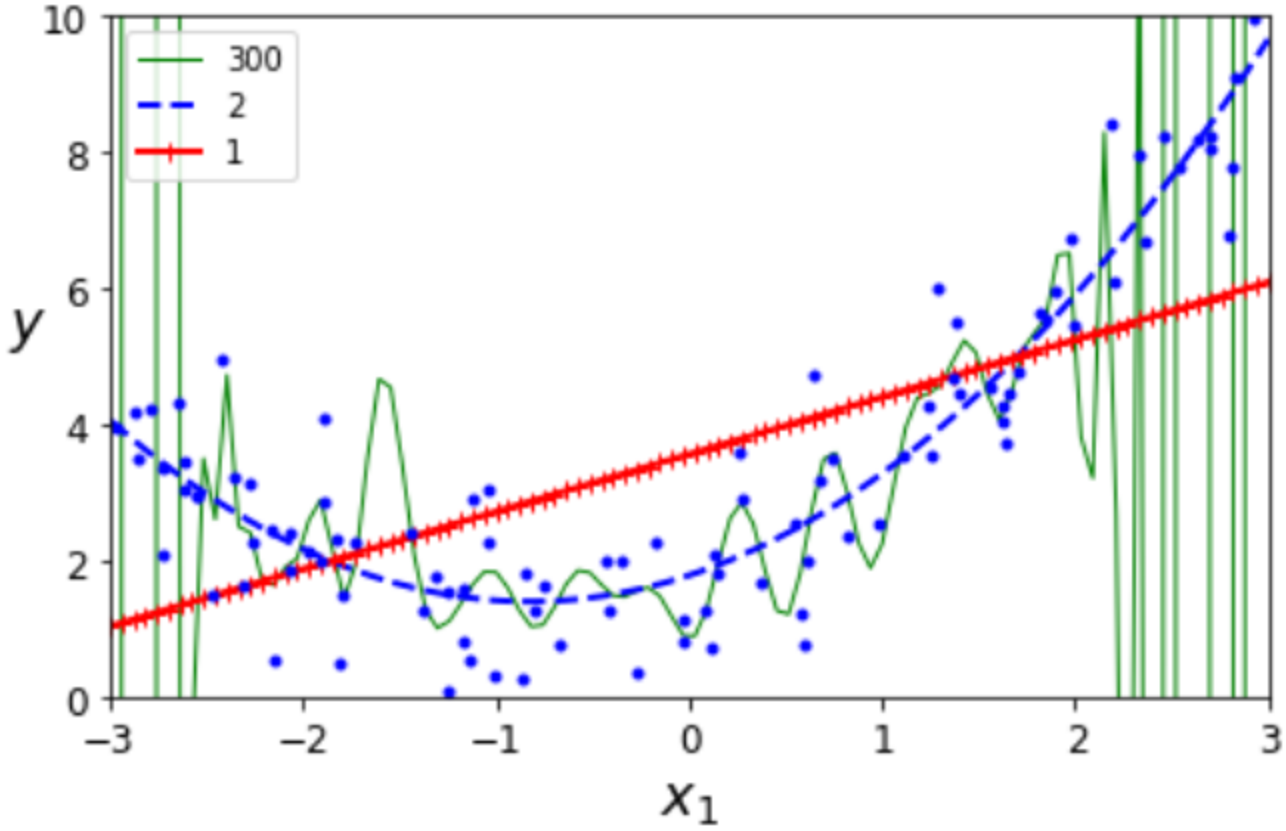
预估的函数： $\hat{y} = 0.56x_1^2 + 0.93x_1 + 1.78$

- 请注意：当存在多个特征时，多项式回归能够找出特征之间的所有关系（这是普通单变量线性回归（只有一个特征）模型无法做到的）。
  - 因为LinearRegression会自动添加当前阶数下特征的所有组合。例如，如果有两个特征a,b，使用3阶（degree=3）的LinearRegression时，将新增加7个特征：a^2, a^3,b^2, b^3, ab, a^2b, ab^2

`PolynomialFeatures(degree=d)` 把一个包含 n 个特征的数组转换为一个包含 $\frac{(n+d)!}{d!n!}$ 特征的数组，$n!$ 表示 n 的阶乘，等于 $1*2*3\cdots*n$。小心大量特征的组合爆炸！

提问：如果训练样本有两个特征a, b，那么，使用4阶（degree=4）的LinearRegression时，将新增加哪些特征？

- 尝试更复杂的多项式来拟合：使用一个300阶的多项式模型去拟合之前的数据集



观察：300阶的多项式模型如何摆动以尽可能接近训练实例

300阶的多项式模型是欠拟合？过拟合？正好？

1. Supervised learning

1.1. Generalized Linear Models

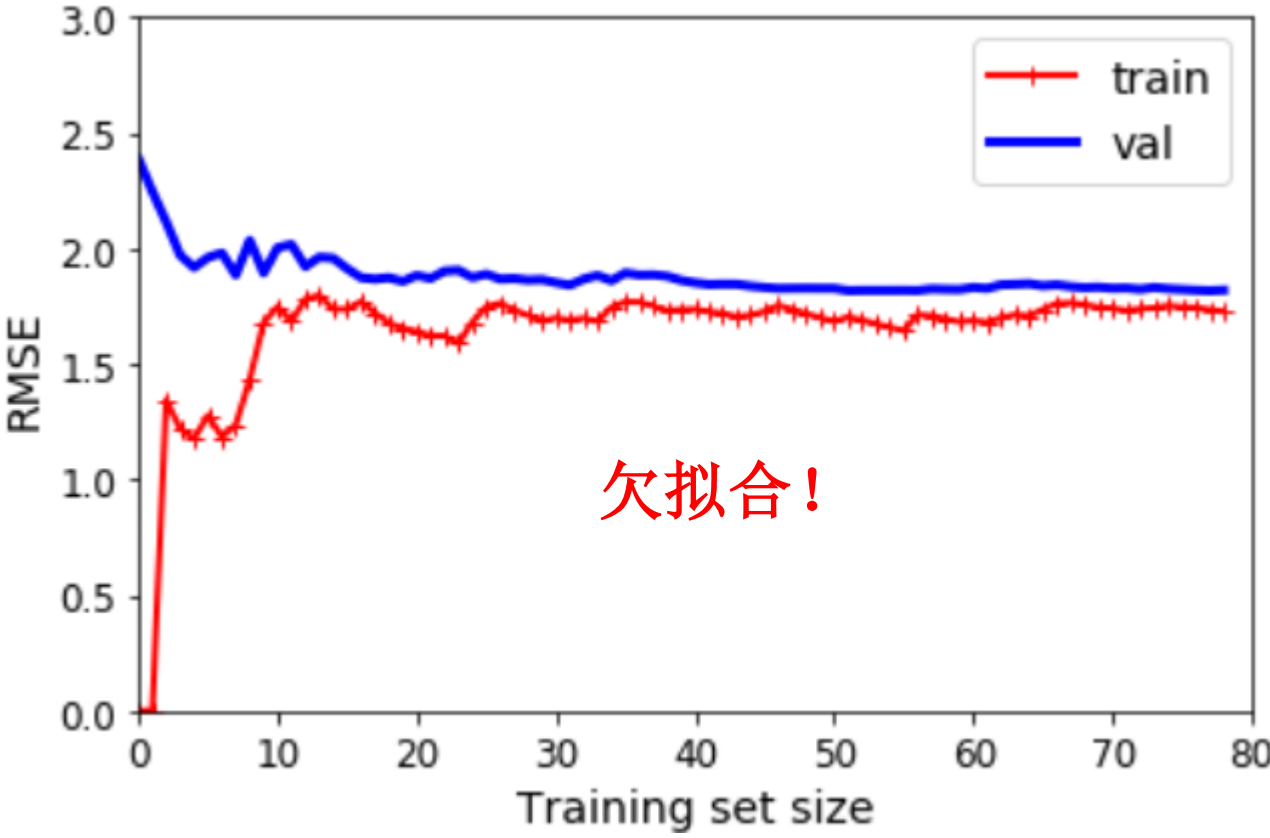
- 判断模型的状态：**是欠拟合？过拟合？正好？**
  - 利用交叉验证（前一章中介绍过）
    - 过拟合：该模型在训练集上表现良好，通过交叉验证指标却得出其在验证集上的表现（泛化能力）很差。
    - 欠拟合：在训练集和验证集 这两方面都表现不好。
  - 利用学习曲线 （本章重难点）
    - 在训练集的不同规模子集上进行多次训练，将得到多个模型
    - 画出这些模型在相应的不同规模的训练集上的表现，即以训练集规模为自变量的误差（分为：在对应规模训练子集上的误差 和 验证集上的误差）
      - 训练集上的学习曲线
      - 验证集上的学习曲线

- 定义一个函数：画出训练集和验证集上的学习曲线

```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

def plot_learning_curves(model, X, y):
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)
    train_errors, val_errors = [], []
    for m in range(1, len(X_train)):
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        y_val_predict = model.predict(X_val)
        train_errors.append(mean_squared_error(y_train_predict, y_train[:m]))
        val_errors.append(mean_squared_error(y_val_predict, y_val))
plt.plot(np.sqrt(train_errors), "r-+", linewidth=2, label="train")
plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="val")
```

- 观察：简单线性回归模型的学习曲线

```
lin_reg = LinearRegression()
plot_learning_curves(lin_reg, X, y)
```



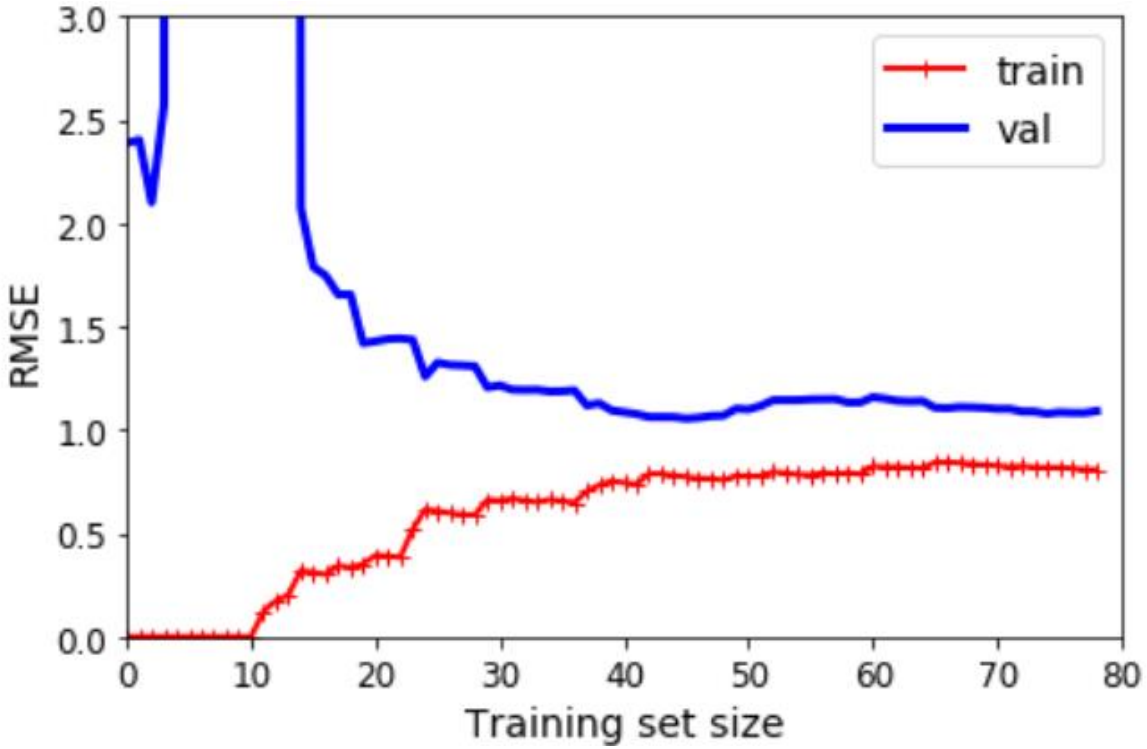
1. 任何时刻，训练集上的表现 都优于 验证集上的表现
2. 训练样本少时，训练集上的拟合效果好，但在验证集上的泛化能力差
3. 训练样本增加时，数据的非线性特点愈加突出，利用线性模型拟合非线性数据的效果差

- 如何解决欠拟合？
  - 添加更多的训练数据？ ×
  - 增加更好的特征？ √
  - 使用更复杂的模型？ √
  - 利用正则化技术？ ×

- 为何会出现欠拟合？
  - 训练数据的规模 与 模型复杂度 出现失衡
  - 训练数据太多， 或者 模型不够复杂

- 观察：同样的数据集上的10阶多项式模型拟合的学习曲线

```
from sklearn.pipeline import Pipeline

polynomial_regression = Pipeline((
    ("poly_features", PolynomialFeatures(degree=10, include_bias=False)),
    ("sgd_reg", LinearRegression()),
))

plot_learning_curves(polynomial_regression, X, y)
```



1. 在同一个训练集上，10阶多项式模型的误差要比线性回归模型低的多。
2. 图中的两条曲线之间有间隔，这意味模型在训练集上的表现要比验证集上好得多，这也是模型过拟合的显著特点。

- 如何解决过拟合？
  - 添加更多的训练数据？ √
  - 增加更好的特征？ ×
  - 使用更复杂的模型？ ×
  - 利用正则化技术？ √

- 为何会出现欠拟合？
  - 训练数据的规模 与 模型复杂度 出现失衡
  - 训练数据太少， 或者 模型太复杂

正则化(Regularization)

- 正则化技术
  - 是解决过拟合问题的通用技术。
  - 正则化一个模型，是指对该模型增加限制，使得该模型自由度减少
  - 比如，正则化一个多项式模型，一个简单的方法就是减少多项式的阶数。
- 如何实现正则化？
  - 在损失函数中，增加正则项，以约束模型中参数的权重。
    - L’(θ) = L(θ) + 正则项

$$\|\theta\|_1 = |w_1| + |w_2| + \dots$$

$$\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \dots$$

正则化(Regularization)

- L’(θ) = L(θ) + 正则项

$$\|\theta\|_1 = |w_1| + |w_2| + \dots$$

$$\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \dots$$

- 寻找最佳参数θ，使得新的损失函数L’(θ) 最小化
  - Find **a set of weight** not only minimizing original cost L(θ) but also close to zero

- 注意：
  - 正则项只有在训练过程中才会被加到损失函数，即训练过程中使用L’(θ) 来计算梯度，以更新参数。
  - 一旦训练完成，应该使用没有正则化的损失函数L(θ)去测量评价模型的表现
  - 偏置项（即常数项）没有正则化

正则化(Regularization)

- 三种正则化方法：
  - 岭（Ridge）回归（ **L2-norm** ）

$$L'(\theta) = L(\theta) + \lambda \cdot \|\theta\|_2 \quad , \quad \|\theta\|_2 = (w_1)^2 + (w_2)^2 + \dots$$

  - Lasso回归（ **L1-norm** ）

$$L'(\theta) = L(\theta) + \lambda \cdot \|\theta\|_1 \quad , \|\theta\|_1 = |w_1| + |w_2| + \dots$$

  - ElasticNet（ **L1 + L2** ）

$$L'(\theta) = L(\theta) + \lambda \cdot [\rho \cdot \|\theta\|_1 + (1 - \rho) \cdot \|\theta\|_2]$$

超参数：正则化因子λ（ λ ϵ [0,1]）

- 以线性回归模型为例，分别介绍三类正则化的线性模型

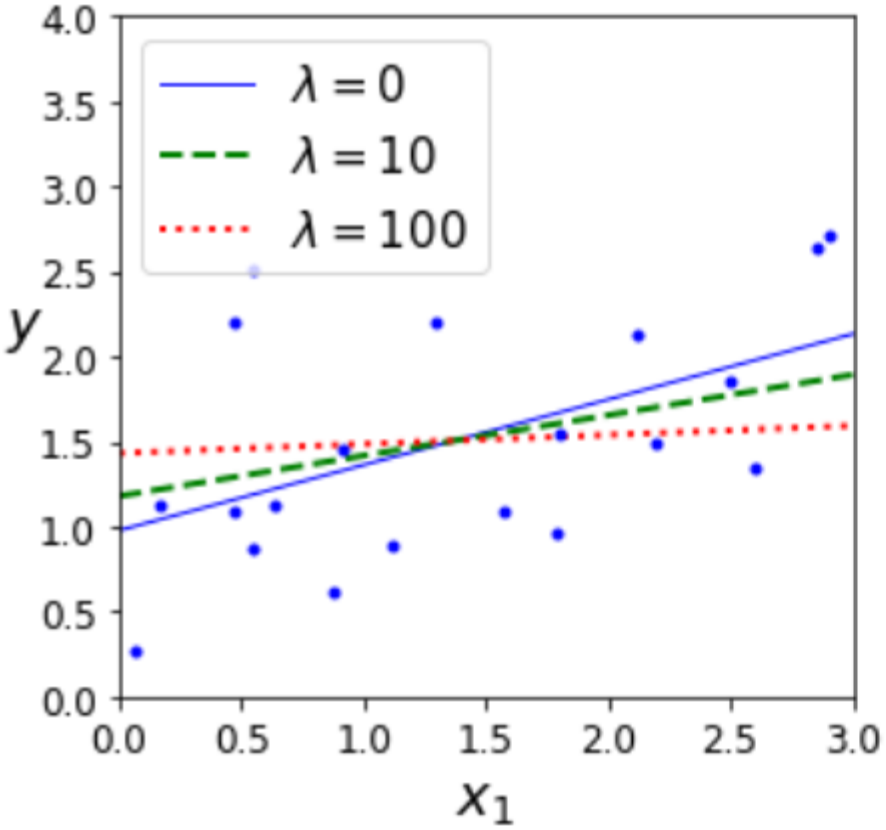
1. Supervised learning

1.1. Generalized Linear Models

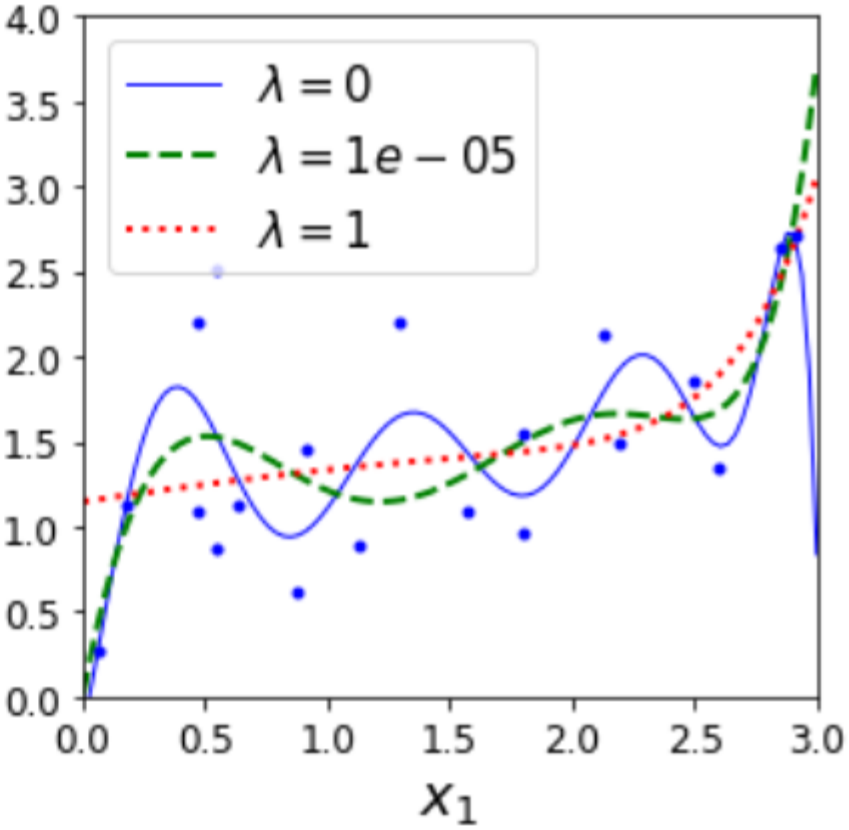
超参数：正则化强度λ

岭（Ridge）回归

- 损失/成本函数： $J(\theta) = MSE(\theta) + \lambda \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$



正则化的线性模型

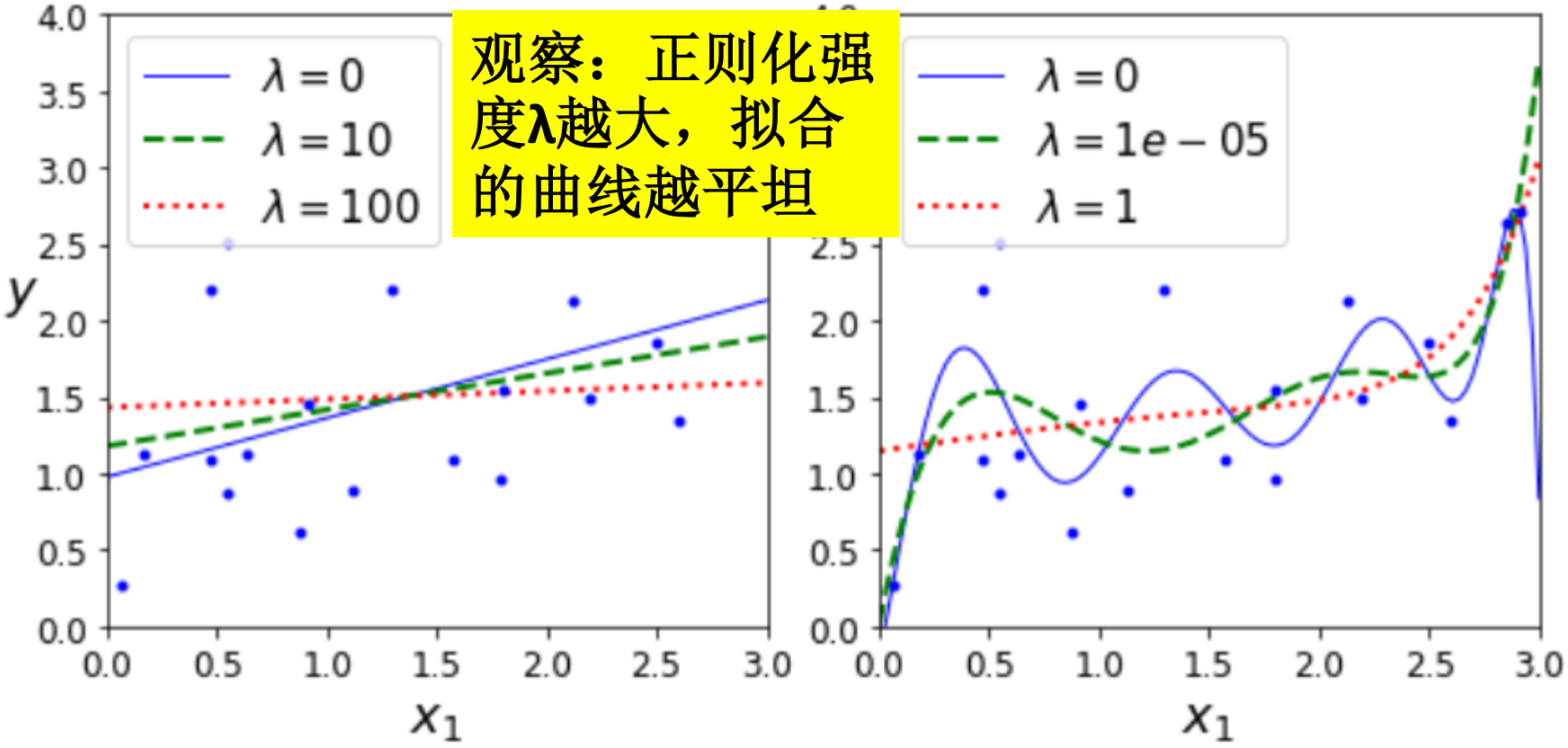


正则化的10阶段多项式回归模型

超参数：正则化强度λ

岭（Ridge）回归

- 损失/成本函数： $J(\theta) = MSE(\theta) + \lambda \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$



正则化的线性模型

正则化的10阶段多项式回归模型

```
lambda = 0 线性回归模型的系数和常数项分别为：
[[0.3852145]]
[0.97573667]
lambda = 10 线性回归模型的系数和常数项分别为：
[[0.23812306]]
[1.17770894]
lambda = 100 线性回归模型的系数和常数项分别为：
[[0.05367258]]
[1.43097916]
lambda = 0 多项式回归模型的系数和常数项分别为：
[[ 6.43580701e+00  2.57281711e+01 -2.75008054e+02 -1.90585036e+03
   2.11658121e+04 -7.55141492e+04  1.37974410e+05 -1.39291200e+05
   7.39888221e+04 -1.61744622e+04]]
[1.50467735]
lambda = 1e-05 多项式回归模型的系数和常数项分别为：
[[  6.8001882  -34.39584308  57.08350671 -19.63712861 -23.18607954
    3.55062464  11.96268146   1.85226073  -6.63277117   3.09958912]]
[1.50467735]
lambda = 1 多项式回归模型的系数和常数项分别为：
[[ 0.19242496 -0.05219979 -0.05187216 -0.02809326 -0.00486859  0.01898063
   0.04493381  0.07311     0.10293252  0.13360102]]
[1.50467735]
```

```
from sklearn.linear_model import Ridge
```

```
def plot_model(model_class, polynomial, alphas, **model_kargs):
    for alpha, style in zip(alphas, ("b-", "g--", "r:")):
        model = model_class(alpha, **model_kargs) if alpha > 0 else LinearRegression()
        if polynomial:
            model = Pipeline([
                    ("poly_features", PolynomialFeatures(degree=10, include_bias=False)),
                    ("std_scaler", StandardScaler()),
                    ("regul_reg", model),
                ])
        model.fit(X, y)
        y_new_regul = model.predict(X_new)
        lw = 2 if alpha > 0 else 1
        plt.plot(X_new, y_new_regul, style, linewidth=lw, label=r"$\lambda = {}$".format(alpha))
    plt.plot(X, y, "b.", linewidth=3)
    plt.legend(loc="upper left", fontsize=15)
    plt.xlabel("$x_1$", fontsize=18)
    plt.axis([0, 3, 0, 4])
```

```
plt.figure(figsize=(8,4))
plt.subplot(121)
plot_model(Ridge, polynomial=False, alphas=(0, 10, 100), random_state=42)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.subplot(122)
plot_model(Ridge, polynomial=True, alphas=(0, 10**-5, 1), random_state=42)

save_fig("ridge_regression_plot")
plt.show()
```

```
from sklearn.linear_model import Ridge
```

```
def plot_model(model_class, polynomial, alphas, **model_kargs):
    for alpha, style in zip(alphas, ("b-", "g--", "r:")):
        model = model_class(alpha, **model_kargs) if alpha > 0 else LinearRegression()
        if polynomial:
            model = Pipeline([
                    ("poly_features", PolynomialFeatures(degree=10, include_bias=False)),
                    ("std_scaler", StandardScaler()),
                    ("regul_reg", model),
                ])
        model.fit(X, y)
        y_new_regul = model.predict(X_new)
        lw = 2 if alpha > 0 else 1
        plt.plot(X_new, y_new_regul, style, linewidth=lw, label=r"$\lambda = {}$".format(alpha))
    plt.plot(X, y, "b.", linewidth=3)
    plt.legend(loc="upper left", fontsize=15)
    plt.xlabel("$x_1$", fontsize=18)
    plt.axis([0, 3, 0, 4])
```

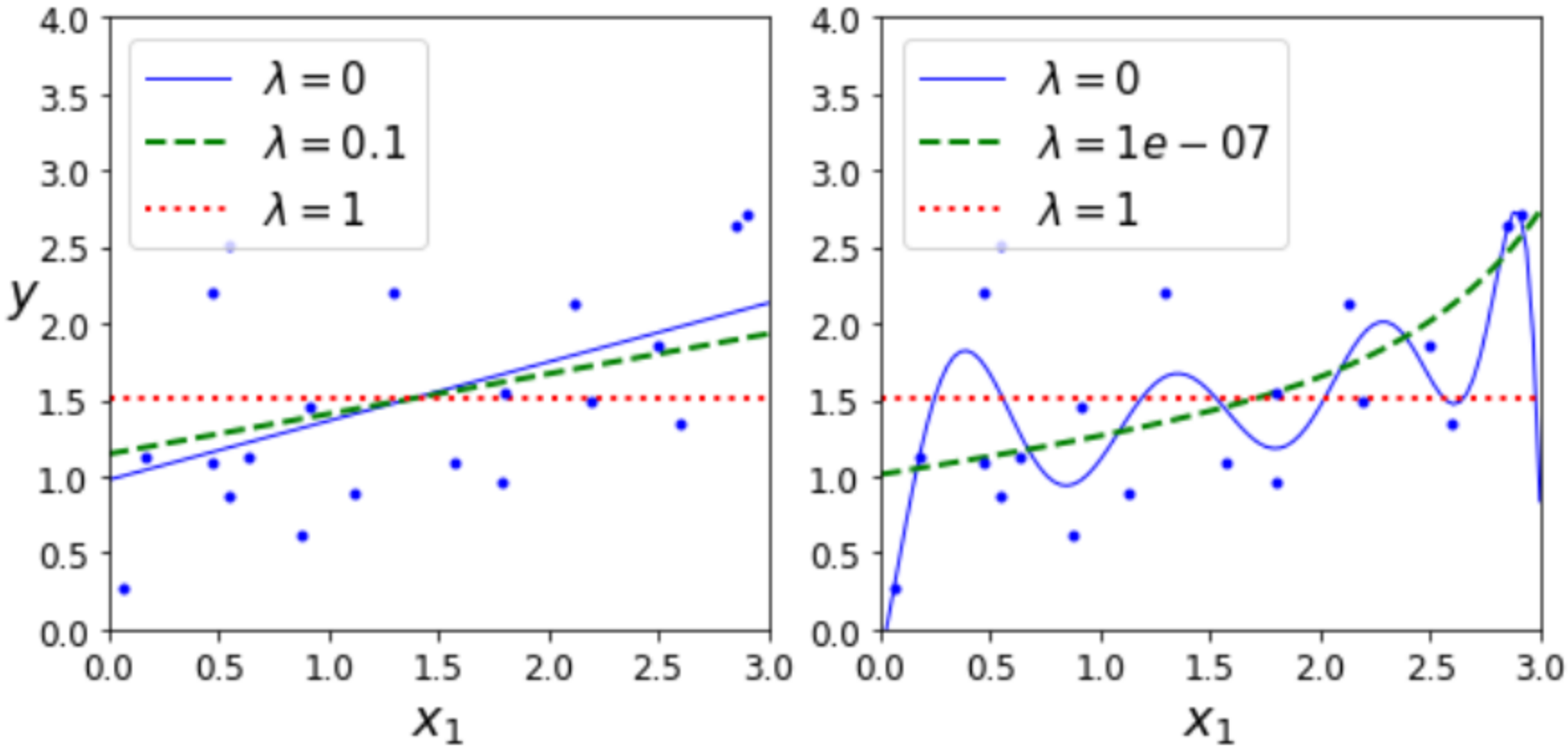
Q：超参数-正则化强度λ 在代码哪里体现？

注意：在正则化前，须先对数据进行放缩（可以使用StandardScaler）

```
plt.figure(figsize=(8,4))
plt.subplot(121)
plot_model(Ridge, polynomial=False, alphas=(0, 10, 100), random_state=42)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.subplot(122)
plot_model(Ridge, polynomial=True, alphas=(0, 10**-5, 1), random_state=42)

save_fig("ridge_regression_plot")
plt.show()
```

套索（ Lasso ）回归

- 损失/成本函数：$J(\theta) = MSE(\theta) + \lambda \sum_{i=1}^{n} |\theta_i|$



正则化的线性模型

正则化的10阶段多项式回归模型

```
lambda = 0 线性回归模型的系数和常数项分别为：
[[0.3852145]]
[0.97573667]
lambda = 0.1 线性回归模型的系数和常数项分别为：
[0.26167212]
[1.14537356]
lambda = 1 线性回归模型的系数和常数项分别为：
[0.]
[1.50467735]
lambda = 0 多项式回归模型的系数和常数项分别为：
[[ 6.43580701e+00  2.57281711e+01 -2.75008054e+02 -1.90585036e+03
   2.11658121e+04 -7.55141492e+04  1.37974410e+05 -1.39291200e+05
   7.39888221e+04 -1.61744622e+04]]
[1.50467735]
lambda = 1e-07 多项式回归模型的系数和常数项分别为：
[0.20758606 0.03784502 0.05125049 0.04279452 0.03407145 0.02752684
 0.02275008 0.01918325 0.0164334  0.01424524]
[1.50467735]
lambda = 1 多项式回归模型的系数和常数项分别为：
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[1.50467735]
```

1. Lasso回归可以将无用的特征的权重降为0

```
from sklearn.linear_model import Lasso

plt.figure(figsize=(8,4))
plt.subplot(121)
plot_model(Lasso, polynomial=False, alphas=(0, 0.1, 1), random_state=42)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.subplot(122)
plot_model(Lasso, polynomial=True, alphas=(0, 10**-7, 1), tol=1, random_state=42)

save_fig("lasso_regression_plot")
plt.show()
```

弹性网络（ElasticNet）

- 损失/成本函数：

$$J(\theta) = MSE(\theta) + r\lambda\sum_{i=1}^{n}|\theta_i| + \frac{1-r}{2}\lambda\sum_{i=1}^{n}\theta_i^2$$

```
>>> from sklearn.linear_model import ElasticNet
>>> elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
>>> elastic_net.fit(X, y)
>>> elastic_net.predict([[1.5]])
array([ 1.54333232])
```

扩展：SGD + 正则化参数**penalty**

```
>>> sgd_reg = SGDRegressor(penalty="l2")
>>> sgd_reg.fit(X, y.ravel())
>>> sgd_reg.predict([[1.5]])
array([[ 1.13500145]])
```

penalty : ***str, 'none', 'l2', 'l1', or 'elasticnet'***

The penalty (aka regularization term) to be used. Defaults to 'l2' which is the standard regularizer for linear SVM models. 'l1' and 'elasticnet' might bring sparsity to the model (feature selection) not achievable with 'l2'.

总结——正则化技术

- 该如何选择线性回归，岭回归，Lasso回归，弹性网络呢？
- 应该避免使用简单的线性回归：模型太简单，无法学习到复杂数据内部的规律
- 一般来说有一点正则项的表现更好
- 岭回归是一个很好的首选项
- 如果你的特征仅有少数是真正有用的，就应该选择Lasso和弹性网络，它能够将无用特征的权重降为零。
- 一般来说，弹性网络的表现要比Lasso好，因为当特征数量比样本的数量大的时候，或者特征之间有很强的相关性时，Lasso可能会表现的不规律

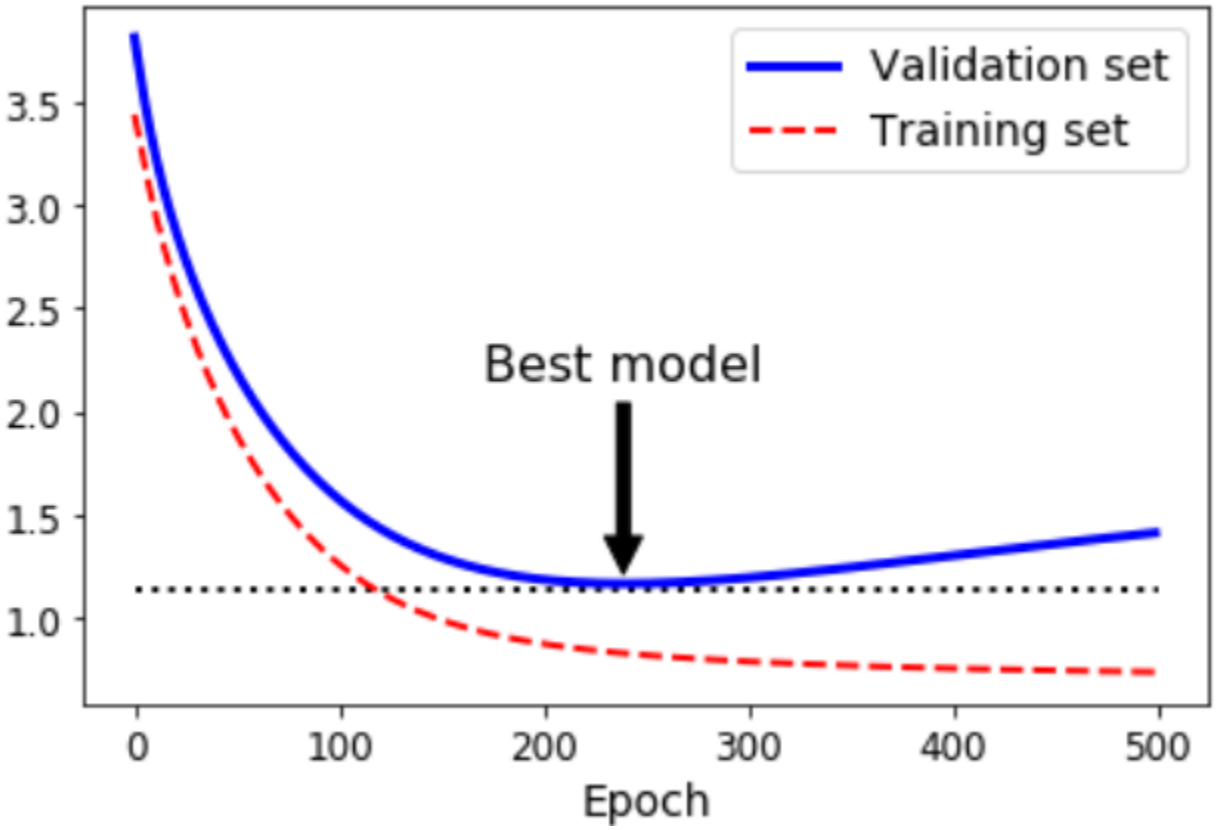
早期停止法(Early Stopping)

- 是一种非常特殊的正则化方法

1. 什么是早期停止法?

当梯度下降在验证错误达到最小值时立即停止训练。

2. 如何判断某一个时刻的验证误差为最小值?

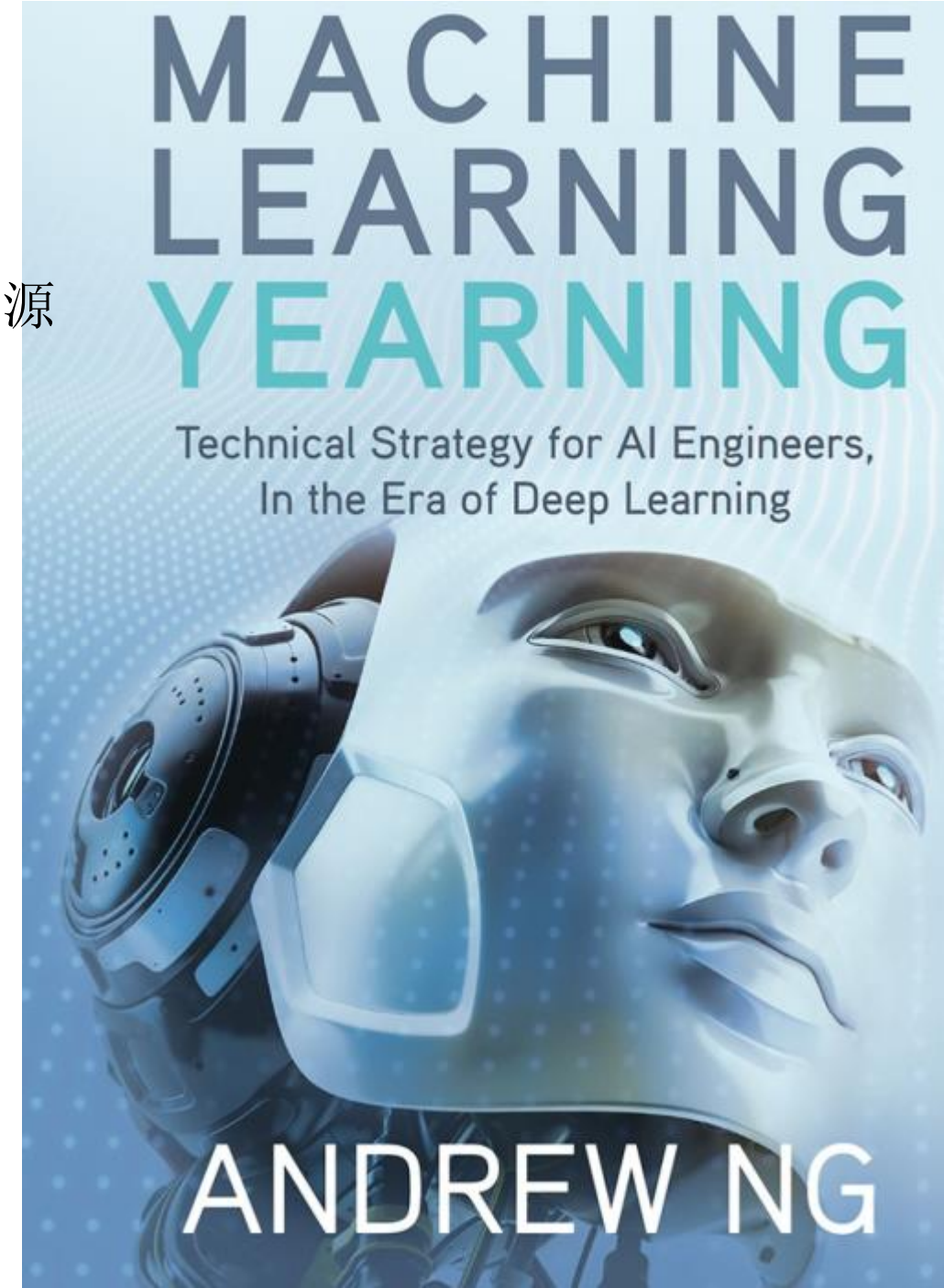


- 如何判断某一个时刻的验证误差为最小值？
  - 随机梯度和小批量梯度下降不是平滑曲线，你可能很难知道它是否达到最小值。
  - 一种解决方案是，只有在验证误差高于最小值一段时间后（你确信该模型不会变得更好了），才停止，之后将模型参数回滚到验证误差最小值。

```
from sklearn.base import clone
sgd_reg = SGDRegressor(n_iter=1, warm_start=True, penalty=None,learning_rate="constant"
, eta0=0.0005)

minimum_val_error = float("inf")
best_epoch = None
best_model = None
for epoch in range(1000):
    sgd_reg.fit(X_train_poly_scaled, y_train)
    y_val_predict = sgd_reg.predict(X_val_poly_scaled)
    val_error = mean_squared_error(y_val_predict, y_val)
    if val_error < minimum_val_error:
        minimum_val_error = val_error
        best_epoch = epoch
        best_model = clone(sgd_reg)
```

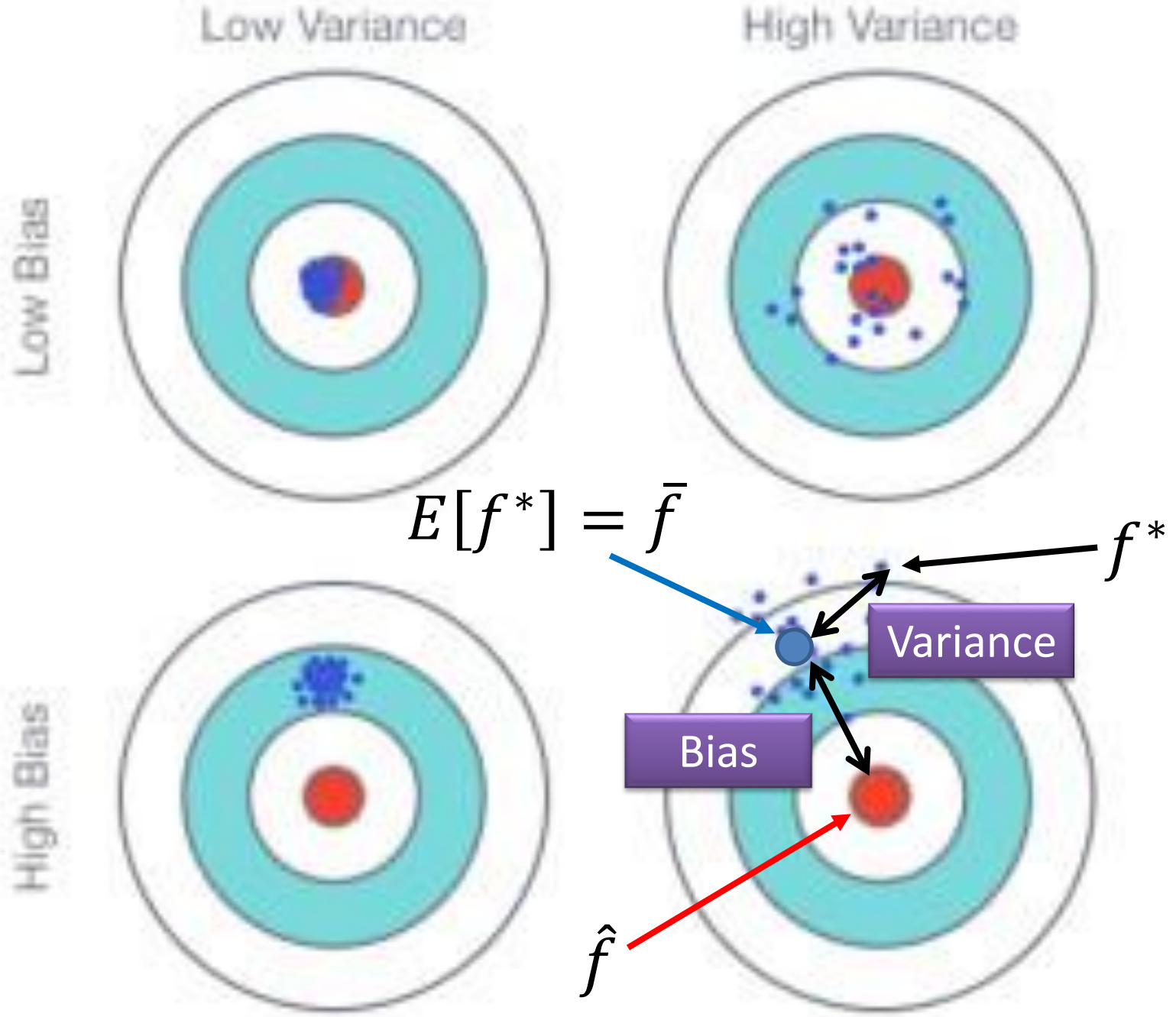
关于误差

20 偏差和方差：误差的两大来源
21 偏差和方差举例
22 与最优错误率比较
23 处理偏差和方差
24 偏差和方差间的权衡
25 减少可避免偏差的技术
26 训练集误差分析
27 减少方差的技术
28 诊断偏差与方差：学习曲线
29 绘制训练误差曲线
30 解读学习曲线：高偏差
31 解读学习曲线：其它情况
32 绘制学习曲线



误差来源分析

- Testing Error = Bias error + Variance Error.
  - 偏差（Bias error ）≈ 训练集上的错误率 （非正式地, By吴恩达）
    - 训练集上的错误率 = avoidable error + unavoidable error
  - 方差（Variance error ）≈ 开发集（或测试集）上的表现比训练集上差多少 （非正式地, By 吴恩达）
    - 用“开发/验证误差 - 训练误差”来衡量


Low Variance
High Variance
Low Bias
High Bias
$E[f^*] = \bar{f}$
f^*
Variance
Bias
$\hat{f}$

- 假设你的算法表现如下：
  - 训练错误率= 1%
  - 开发错误率= 11%

- 偏差和方差分别是多少？
  - 估计偏差为1%，方差为10%（=11%-1%）
  - 低偏差，高方差

- 判断结论：过拟合

- 假设你的算法表现如下：
  - 训练错误率= 15%
  - 开发错误率= 16%
- 偏差和方差分别是多少？
  - 估计偏差为15%，方差为1%（=16%-15%）
  - 高偏差，低方差
- 判断结论：欠拟合

- 假设你的算法表现如下：
  - 训练错误率= 15%
  - 开发错误率= 30%

- 偏差和方差分别是多少？
  - 估计偏差为15%，方差为15%（=30%-15%）
  - 高偏差，高方差

- 判断结论：同时过拟合和欠拟合，过拟合/欠拟合术语很难准确应用于此。
  - 高偏差 + 数据不匹配（训练集和验证集的分布规律不同）

- 假设你的算法表现如下：
  - 训练错误率= 0.5%
  - 开发错误率= 1%

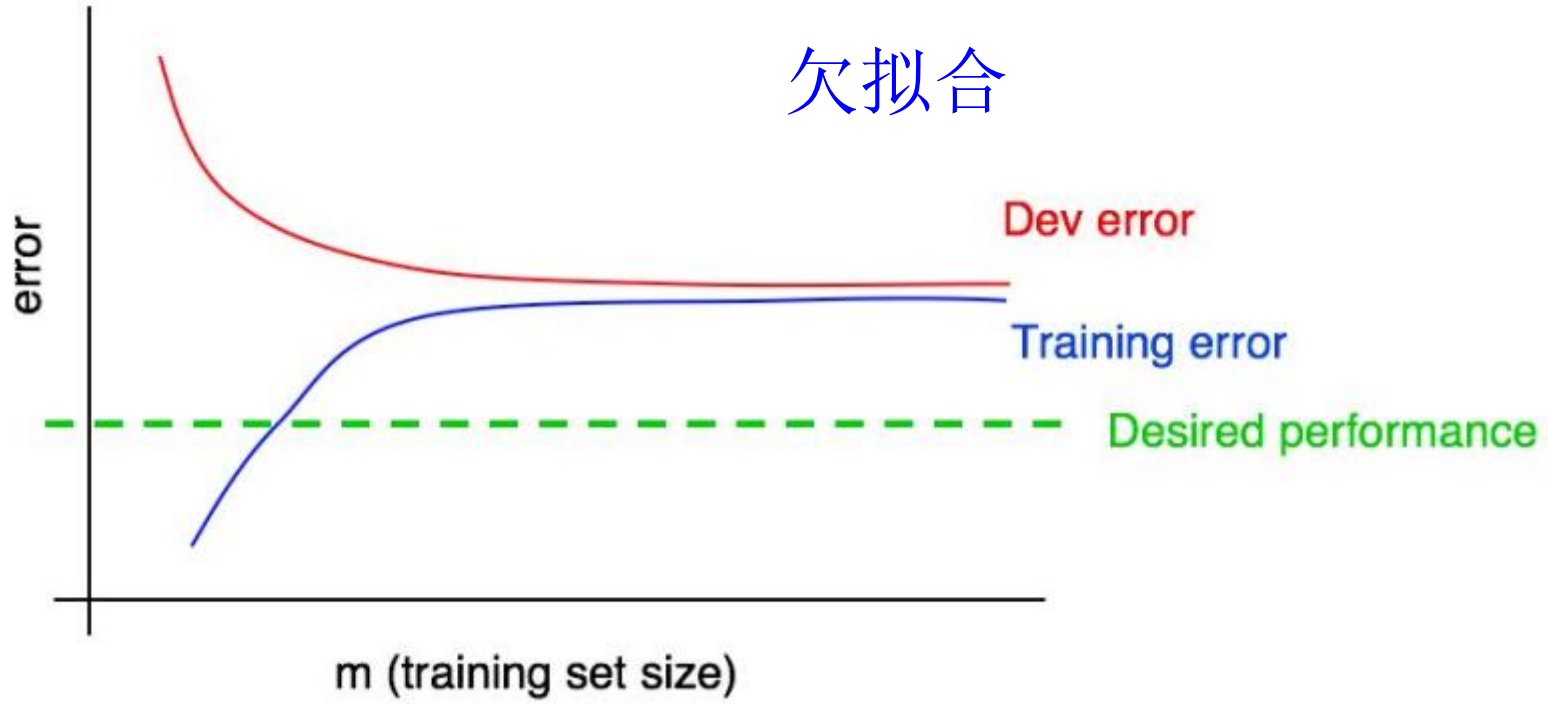
- 偏差和方差分别是多少？
  - 估计偏差为0.5%，方差为0.5%（=1%-0.5%）
  - 低偏差，低方差

- 判断结论：表现很好！ ☺

- 欠拟合： 高bias error, 低 variance error
- 过拟合： 低bias error, 高variance error
- Good： 低bias error, 低variance error

- 利用学习曲线来诊断误差
  - 如何绘制学习曲线？
  - 如何解读学习曲线？
  - 诊断误差之后的进一步判断：欠拟合，过拟合，还是其他？

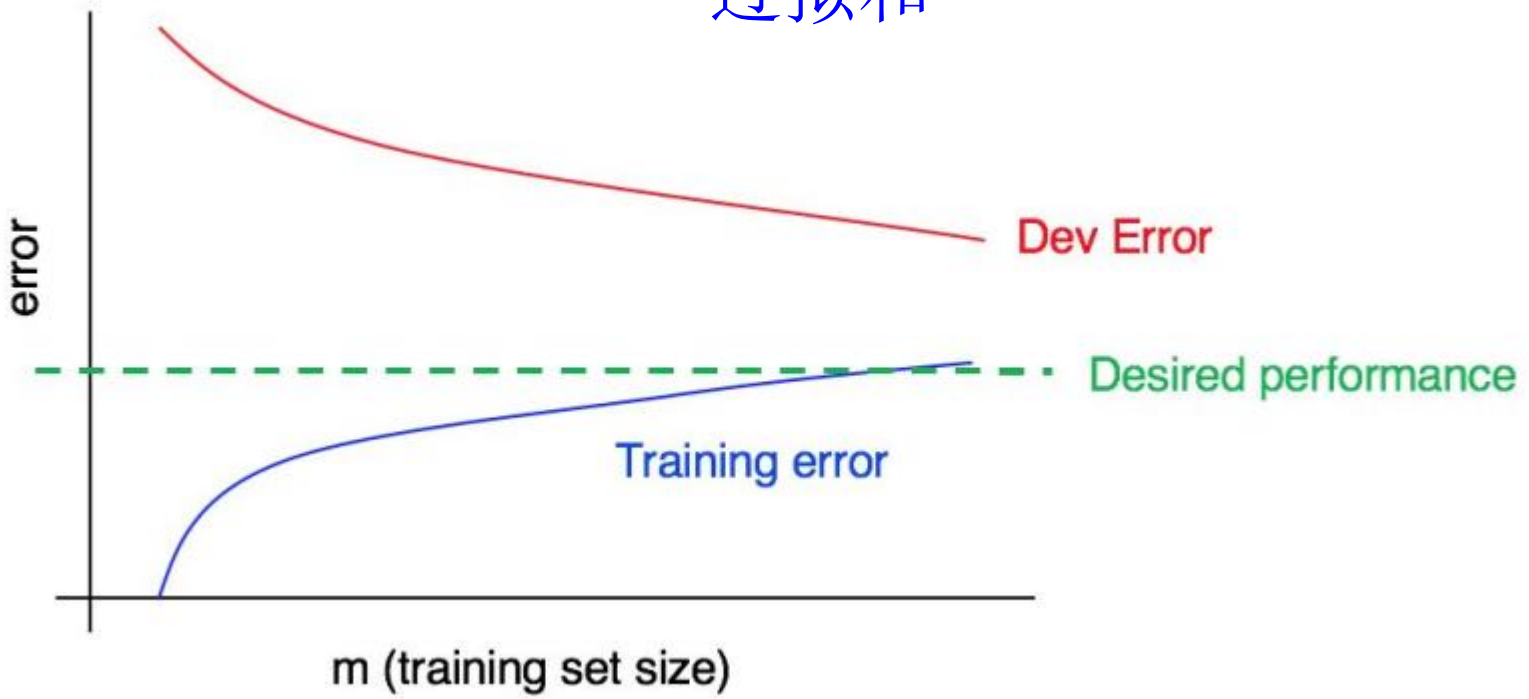
解读学习曲线（1）



1. 高偏差？高方差？还是两者都有？
2. 模型训练结果是？

解读学习曲线（2）

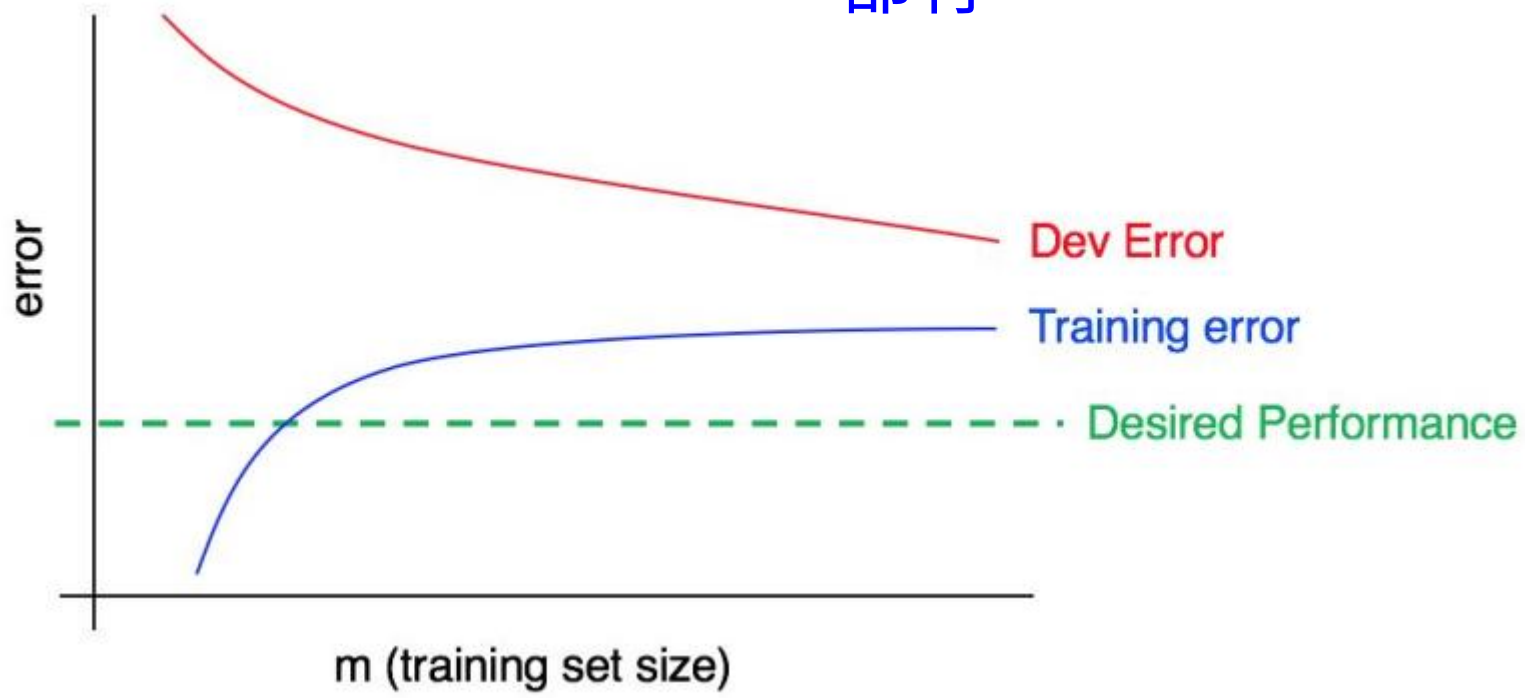
过拟和



1. 高偏差？高方差？还是两者都有？
2. 模型训练结果是？

解读学习曲线（3）

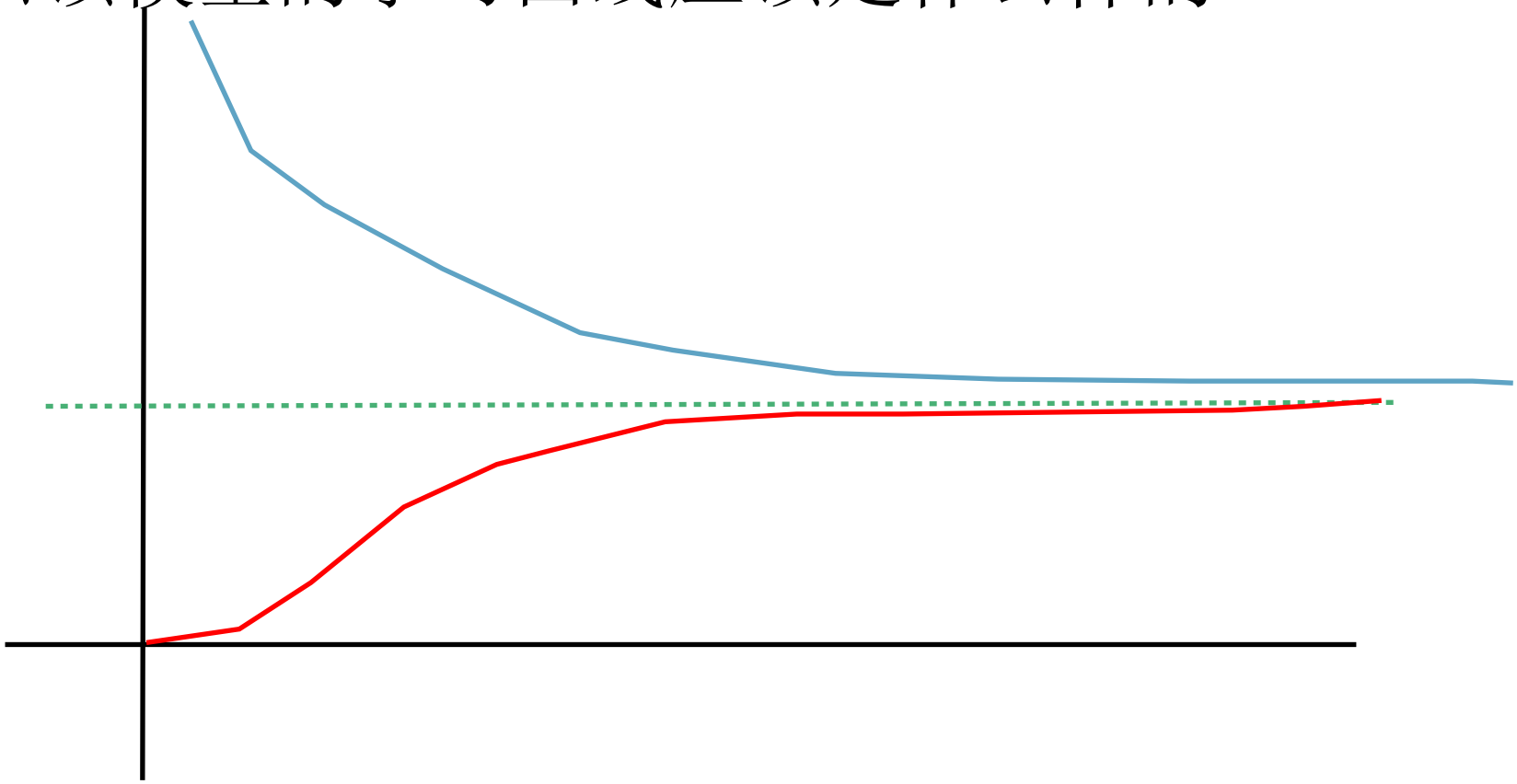
都有



1. 高偏差？高方差？还是两者都有？
2. 模型训练结果是？

解读学习曲线（4）

- 模型在训练集和验证集上表现都很好，那么该模型的学习曲线应该是什么样的？



谢谢！