

# Linear Methods for Regression, Optimization

---

## Overview

---

Linear Regression:

- Task: predict scalar-valued targets (e.g. stock prices)
- Architecture: linear function of the inputs

**Modular approach:**

1. **choose a model** describing the relationships between variables of interest (选择描述我们感兴趣的元素和变量之间关系的模型)
2. **define a loss function** quantifying how bad the fit to the data is
3. **choose a regularizer** (正则化) saying how much we prefer different candidate models (or explanations of data)
4. fit a model that minimizes the loss function and satisfies the constraint/penalty imposed by the regularizer, possibly using an **optimization algorithm**

## Linear Regression

---

### Model

**Model:** In linear regression, we use a *linear* function of the features  $\mathbf{x} = (x_1, \dots, x_D) \in \mathbb{R}^D$  to make predictions  $y$  of the target value  $t \in \mathbb{R}$ :

$$y = f(\mathbf{x}) = \sum_j w_j x_j + b$$

- ▶  $y$  is the **prediction**
- ▶  $\mathbf{w}$  is the **weights**
- ▶  $b$  is the **bias** (or **intercept**)

$\mathbf{w}$  and  $b$  together are the **parameters**

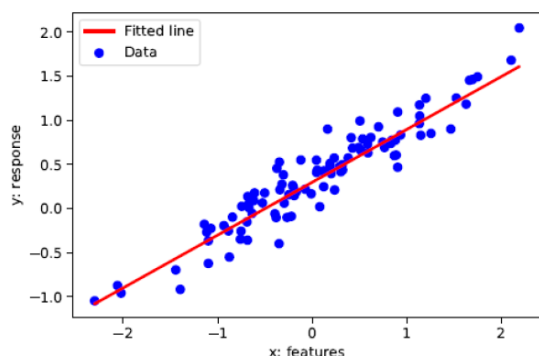
We hope that our prediction is close to the target:  $y \approx t$ .

### Linear Regression

We have a dataset  $\mathcal{D} = \{(\mathbf{x}^{(i)}, t^{(i)}) \text{ for } i = 1, 2, \dots, N\}$  where,

- $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_D^{(i)})^\top \in \mathbb{R}^D$  are the inputs (e.g. age, height)
- $t^{(i)} \in \mathbb{R}$  is the target or response (e.g. income)
- predict  $t^{(i)}$  with a linear function of  $\mathbf{x}^{(i)}$ :

注:  $t^{(i)}$  是对于  $x^{(i)}$  的预测结果



- $t^{(i)} \approx y^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + b$
- Different  $(\mathbf{w}, b)$  define different lines.
- We want the “best” line  $(\mathbf{w}, b)$ .
- How to quantify “best”?

## Loss Function

- A **loss function**  $\mathcal{L}(y, t)$  defines how bad it is if, for some example  $\mathbf{x}$ , the algorithm predicts  $y$ , but the target is actually  $t$ .
- **Squared error loss function**:

$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$$

注:  $y - t$  是 residual (差距), 我们希望它越小越好;

乘以  $\frac{1}{2}$  可以减轻运算量。

- **Cost function**: 所有 training examples 的平均 loss

$$\begin{aligned} \mathcal{J}(\mathbf{w}, b) &= \frac{1}{2N} \sum_{i=1}^N \left( y^{(i)} - t^{(i)} \right)^2 \\ &= \frac{1}{2N} \sum_{i=1}^N \left( \mathbf{w}^\top \mathbf{x}^{(i)} + b - t^{(i)} \right)^2 \end{aligned}$$

- 术语不同: 有些叫 cost 或 average loss
- 符号方面: 将  $\frac{1}{2N} \sum_{i=1}^N \left( y^{(i)} - t^{(i)} \right)^2$  中的  $y^{(i)}$  展开会得到

$$\frac{1}{2N} \sum_{i=1}^N \left( \sum_{j=1}^D \left( w_j x_j^{(i)} + b \right) - t^{(i)} \right)^2$$

注: N 指有 N 个 training examples, D 指 x 有 D 个 features

## Vectorization

- Computing the predictions for the whole dataset:

$$\mathbf{X}\mathbf{w} + b\mathbf{1} = \begin{pmatrix} \mathbf{w}^\top \mathbf{x}^{(1)} + b \\ \vdots \\ \mathbf{w}^\top \mathbf{x}^{(N)} + b \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix} = \mathbf{y}$$

- Computing the squared error cost across the whole dataset:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + b\mathbf{1}$$

$$\mathcal{J} = \frac{1}{2N} \|\mathbf{y} - \mathbf{t}\|^2$$

- We can also add a column of 1's to design matrix, combine the bias and the weights, and conveniently write

$$\mathbf{X} = \begin{bmatrix} 1 & [\mathbf{x}^{(1)}]^\top \\ 1 & [\mathbf{x}^{(2)}]^\top \\ \vdots & \vdots \end{bmatrix} \in \mathbb{R}^{N \times (D+1)} \quad \text{and} \quad \mathbf{w} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \end{bmatrix} \in \mathbb{R}^{D+1}$$

Then, our predictions reduce to  $\mathbf{y} = \mathbf{X}\mathbf{w}$ .

## Minimization loss

Two commonly applied mathematical approaches:

- Algebraic (代数方法), e.g. using inequalities (不等式):
  - ▶ to show  $z^*$  minimizes  $f(z)$ , show that  $\forall z, f(z) \geq f(z^*)$
  - ▶ to show that  $a = b$ , show that  $a \geq b$  and  $b \geq a$
- Calculus: minimum of a smooth function (if it exists) occurs at a critical point (临界点), i.e. point where the derivative is zero (导数为零的点).

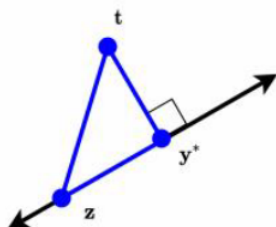
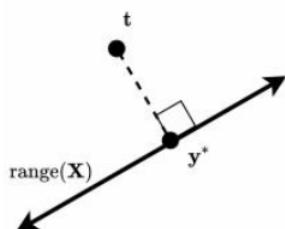
Solutions may be direct or iterative.

## Direct Solution

### Linear Algebra

我们要寻找一个  $\mathbf{w}$  来 minimize  $\|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$ , or equivalently  $\|\mathbf{X}\mathbf{w} - \mathbf{t}\|$ .  
 $\text{range}(\mathbf{X}) = \{\mathbf{X}\mathbf{w} \mid \mathbf{w} \in \mathbb{R}^D\}$  is a  $D$ -dimensional subspace of  $\mathbb{R}^N$ .

The closest point  $\mathbf{y}^* = \mathbf{X}\mathbf{w}^*$  in subspace  $\text{range}(\mathbf{X})$  of  $\mathbb{R}^N$  to arbitrary point  $\mathbf{t} \in \mathbb{R}^N$  is found by orthogonal projection (直角投影).



- We have  $(\mathbf{y}^* - \mathbf{t}) \perp \mathbf{X}\mathbf{w}, \forall \mathbf{w} \in \mathbb{R}^D$

- Why is  $\mathbf{y}^*$  the closest point to  $\mathbf{t}$ ?

- ▶ Consider any  $\mathbf{z} = \mathbf{X}\mathbf{w}$
- ▶ By Pythagorean theorem and the trivial inequality ( $x^2 \geq 0$ ):

$$\begin{aligned} \|\mathbf{z} - \mathbf{t}\|^2 &= \|\mathbf{y}^* - \mathbf{t}\|^2 + \|\mathbf{y}^* - \mathbf{z}\|^2 \\ &\geq \|\mathbf{y}^* - \mathbf{t}\|^2 \end{aligned}$$

注:  $\text{range}(\mathbf{X})$  和它所在的线, 代表的是可能的  $y$  值。

- From the previous slide, we have  $(\mathbf{y}^* - \mathbf{t}) \perp \mathbf{X}\mathbf{w}$ ,  $\forall \mathbf{w} \in \mathbb{R}^D$
- Equivalently, the columns of the design matrix  $\mathbf{X}$  are all orthogonal to  $(\mathbf{y}^* - \mathbf{t})$ , and we have that:

$$\begin{aligned}\mathbf{X}^\top (\mathbf{y}^* - \mathbf{t}) &= \mathbf{0} \\ \mathbf{X}^\top \mathbf{X}\mathbf{w}^* - \mathbf{X}^\top \mathbf{t} &= \mathbf{0} \\ \mathbf{X}^\top \mathbf{X}\mathbf{w}^* &= \mathbf{X}^\top \mathbf{t} \\ \mathbf{w}^* &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t}\end{aligned}$$

## Calculus

- **Partial derivative**: derivative of a multivariate function with respect to one of its arguments.

$$\frac{\partial}{\partial x_1} f(x_1, x_2) = \lim_{h \rightarrow 0} \frac{f(x_1 + h, x_2) - f(x_1, x_2)}{h}$$

- For cost derivatives, use **linearity** and average over data points:

$$\frac{\partial \mathcal{J}}{\partial w_j} = \frac{1}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) x_j^{(i)} \quad \frac{\partial \mathcal{J}}{\partial b} = \frac{1}{N} \sum_{i=1}^N y^{(i)} - t^{(i)}$$

注: 这里是对  $w$ , 和  $b$  的偏导。

- Minimum must occur at a point where partial derivatives are zero.

$$\frac{\partial \mathcal{J}}{\partial w_j} = 0 \quad (\forall j), \quad \frac{\partial \mathcal{J}}{\partial b} = 0.$$

(if  $\partial \mathcal{J} / \partial w_j \neq 0$ , you could reduce the cost by changing  $w_j$ )

- We call the vector of partial derivatives the **gradient**
- Thus, the “gradient of  $f : \mathbb{R}^D \rightarrow \mathbb{R}$ ”, denoted  $\nabla f(\mathbf{w})$ , is:

$$\left( \frac{\partial}{\partial w_1} f(\mathbf{w}), \dots, \frac{\partial}{\partial w_D} f(\mathbf{w}) \right)^\top$$

注: 上式是  $w$  的梯度。

**梯度的方向, 和函数最大增长率的方向一致。**

Analogue (类似情况) of second derivative (the "Hessian" matrix):  $\nabla^2 f(\mathbf{w}) \in \mathbb{R}^{D \times D}$  is a matrix with  $[\nabla^2 f(\mathbf{w})]_{ij} = \frac{\partial^2}{\partial w_i \partial w_j} f(\mathbf{w})$

注: 海森矩阵, 描述了函数的局部曲率, 可判定多元函数的极值问题。用二次导数, 可以确定梯度的变化趋势。

- We seek  $\mathbf{w}$  to minimize  $\mathcal{J}(\mathbf{w}) = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$
- Taking the gradient with respect to  $\mathbf{w}$  (see course notes for additional details) we get:

$$\nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}) = \mathbf{X}^\top \mathbf{X} \mathbf{w} - \mathbf{X}^\top \mathbf{t} = \mathbf{0}$$

- We get the same optimal weights as before:

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

注：矩阵求导不能用链式法则，有特殊的规则，因此：

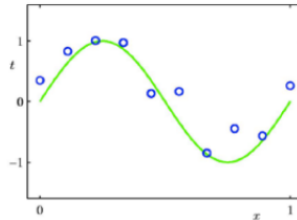
$$\begin{aligned} \nabla_w \mathcal{J}(w) &= \frac{1}{2} (X^2 w^2 + t^2 - 2Xwt)' \\ &= \frac{1}{2} (2X^2 w - 2Xt) \\ &= X^2 w - Xt \\ &= X^\top X w - X^\top t \end{aligned}$$

## Feature Mapping (Basis Expansion)

不是所有输入和输出的关系都是线性的，更多的可能是多项式关系 (polynomial relation)。如果样本量多，回归问题很复杂，而原始特征只有  $x_1, x_2$ 。可以用多项式创建更多的特征  $x_1, x_2, x_1^2, x_2^2, \dots$ 。因为更多的特征进行回归时，得到的分割线可以是任意高阶函数的形状。

**通过一定的映射，把数据映射入高维之后，便于分界。**因为保留了映射前的特征，所以叫特征映射。

If the relationship doesn't look linear, we can fit a polynomial.



Fit the data using a degree- $M$  polynomial function of the form:

$$y = w_0 + w_1 x + w_2 x^2 + \dots + w_M x^M = \sum_{i=0}^M w_i x^i$$

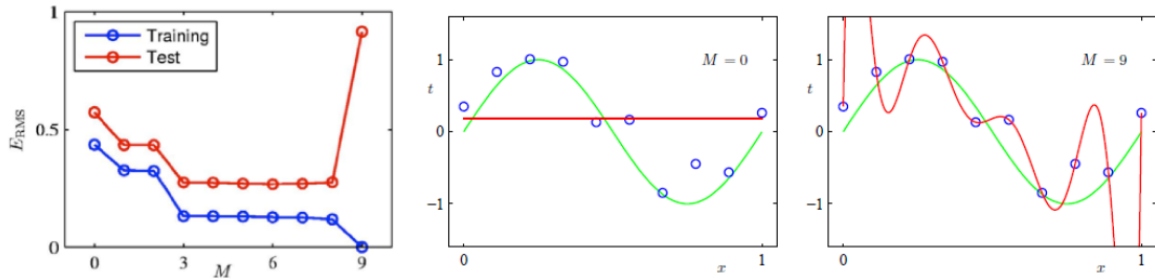
- Here the feature mapping is  $\psi(x) = [1, x, x^2, \dots, x^M]^\top$ .
- We can still use linear regression to find  $\mathbf{w}$  since  $y = \psi(x)^\top \mathbf{w}$  is linear in  $w_0, w_1, \dots$ .
- In general,  $\psi$  can be any function. Another example:  

$$\psi(x) = [1, \sin(2\pi x), \cos(2\pi x), \sin(4\pi x), \dots]^\top.$$

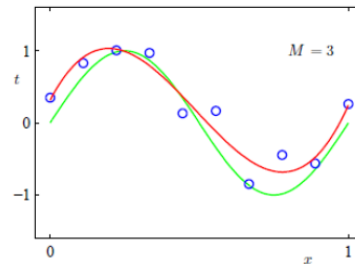
注：在这个问题中，input 只有一个 feature，就是  $x$  轴的值。将这个 feature 映射到  $M$  个维度中，就得到了上式。

Underfitting (M=0): model is too simple — does not fit the data.

Overfitting (M=9): model is too complex — fits perfectly.



Good model (M=3): Achieves small test error (generalizes well).



## Regularization

多项式的 degree  $M$  控制了模型的复杂度。  $M$  是一个超参数，和 KNN 中的  $k$  一样，因此我们同样可以使用 validation set 来进行调整。

不过，我们还有另外的方法：keep the model large, but **regularize** it

- **Regularizer**: a function that quantifies how much we prefer one hypothesis vs. another

### L2 Regularization

我们可以选择使用 **L<sup>2</sup> penalty** 作为 regularizer 来使 weights 变小。

$$\mathcal{R}(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|_2^2 = \frac{1}{2} \sum_j w_j^2$$

正则化的代价函数 (regularized cost function) 在 数据的拟合 和 权重的范数 (norm of the weights) 之间进行权衡。

$$\mathcal{J}_{\text{reg}}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \lambda \mathcal{R}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \frac{\lambda}{2} \sum_j w_j^2$$

- If you fit training data poorly,  $\mathcal{J}$  is large (误差大). If your optimal weights have high values,  $\mathcal{R}$  is large (模型复杂).
- Large  $\lambda$  penalizes weight values more.
- 和  $M$  一样， $\lambda$  也是 hyperparameter，可以通过 validation set 调整

### L2 Regularized Least Squares: Ridge regression

For the least squares problem, we have  $\mathcal{J}(\mathbf{w}) = \frac{1}{2N} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$ .

- When  $\lambda > 0$  (with regularization), regularized cost gives

$$\begin{aligned}\mathbf{w}_{\lambda}^{\text{Ridge}} &= \underset{\mathbf{w}}{\operatorname{argmin}} \mathcal{J}_{\text{reg}}(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2N} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \\ &= (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{t}\end{aligned}$$

- The case  $\lambda = 0$  (no regularization) reduces to least squares solution!
- Note that it is also common to formulate this problem as  $\underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$  in which case the solution is  $\mathbf{w}_{\lambda}^{\text{Ridge}} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{t}$ .

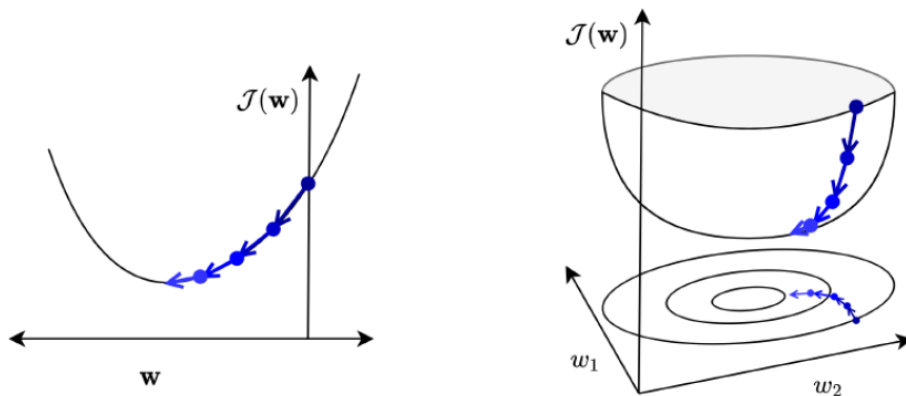
注:  $\mathbf{I}$  表示单位矩阵, 即在主对角线上元素均为 1, 而其他元素都是 0

## Gradient Descent

很多时候, 我们没有 direct solution。

Gradient descent is an [iterative algorithm](#), which means we apply an update repeatedly until some criterion is met.

We [initialize](#) the weights to something reasonable (e.g. all zeros) and repeatedly adjust them in the [direction of steepest descent](#).





- Observe:
  - ▶ if  $\partial\mathcal{J}/\partial w_j > 0$ , then increasing  $w_j$  increases  $\mathcal{J}$ .
  - ▶ if  $\partial\mathcal{J}/\partial w_j < 0$ , then increasing  $w_j$  decreases  $\mathcal{J}$ .
- The following update always decreases the cost function for small enough  $\alpha$  (unless  $\partial\mathcal{J}/\partial w_j = 0$ ):

$$w_j \leftarrow w_j - \alpha \frac{\partial\mathcal{J}}{\partial w_j}$$

- $\alpha > 0$  is a **learning rate** (or step size). The larger it is, the faster  $\mathbf{w}$  changes.
  - ▶ We'll see later how to tune the learning rate, but values are typically small, e.g. 0.01 or 0.0001.
  - ▶ If cost is the sum of  $N$  individual losses rather than their average, smaller learning rate will be needed ( $\alpha' = \alpha/N$ ).
- This gets its name from the **gradient**:

$$\nabla_{\mathbf{w}}\mathcal{J} = \frac{\partial\mathcal{J}}{\partial\mathbf{w}} = \begin{pmatrix} \frac{\partial\mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial\mathcal{J}}{\partial w_D} \end{pmatrix}$$

- ▶ This is the direction of fastest increase in  $\mathcal{J}$ .
- Update rule in vector form:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial\mathcal{J}}{\partial\mathbf{w}}$$

And for linear regression we have:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) \mathbf{x}^{(i)}$$

- So gradient descent updates  $\mathbf{w}$  in the direction of fastest *decrease*.
- Observe that once it converges, we get a critical point, i.e.  $\frac{\partial\mathcal{J}}{\partial\mathbf{w}} = \mathbf{0}$ .

## Gradient Descent under the L2 Regularization



- Gradient descent update to minimize  $\mathcal{J}$ :

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial}{\partial \mathbf{w}} \mathcal{J}$$

- The gradient descent update to minimize the  $L^2$  regularized cost  $\mathcal{J} + \lambda \mathcal{R}$  results in [weight decay](#):

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \alpha \frac{\partial}{\partial \mathbf{w}} (\mathcal{J} + \lambda \mathcal{R}) \\ &= \mathbf{w} - \alpha \left( \frac{\partial \mathcal{J}}{\partial \mathbf{w}} + \lambda \frac{\partial \mathcal{R}}{\partial \mathbf{w}} \right) \\ &= \mathbf{w} - \alpha \left( \frac{\partial \mathcal{J}}{\partial \mathbf{w}} + \lambda \mathbf{w} \right) \\ &= (1 - \alpha \lambda) \mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \end{aligned}$$

## Stochastic Gradient Descent

- So far, the cost function  $\mathcal{J}$  has been the average loss over the training examples:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}^{(i)} = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y(\mathbf{x}^{(i)}, \boldsymbol{\theta}), t^{(i)}).$$

( $\boldsymbol{\theta}$  denotes the parameters; e.g., in linear regression,  $\boldsymbol{\theta} = (\mathbf{w}, b)$ )

- By linearity,

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}} = \frac{1}{N} \sum_{i=1}^N \frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}.$$

计算梯度需要计算所有的 training example, 这称为 **batch training**。

batch training 有时是不切实际的, 如果你有一个大的数据集  $N \gg 1$  (例如数百万个 training examples)。

**随机梯度下降 (SGD)** : 根据单个 training example 的梯度更新参数:

- 均匀随机地选择  $i$  ( $i$  是数据集中的的一个 example)
- $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}$

每个 SGD 更新的成本都与  $N$  无关, SGD 甚至在看到所有数据之前就可以取得重大进展。

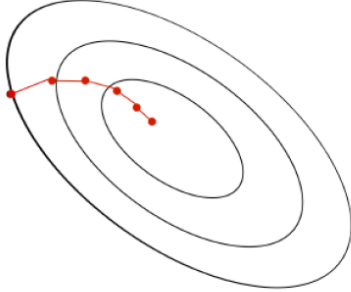
数学理由: 如果随机均匀地采样出一个 training example, 随机梯度是对批次梯度 (batch gradient) 的公正估计 (unbiased estimate) :

$$\mathbb{E} \left[ \frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}} \right] = \frac{1}{N} \sum_{i=1}^N \frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}}$$

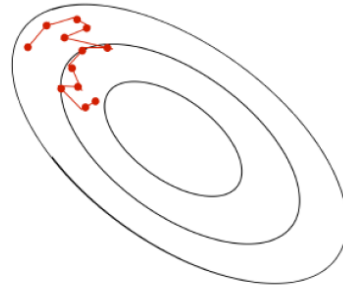
注: 上式是对 batch gradient 的估计, 结果和前面 By linearity 一致, 所以是 unbiased estimate。

使用单一的 training example 来估计梯度可能方差会很高，我们可以随机选择中等大小的数据集  $\mathcal{M} \subset \{1, \dots, N\}$  (称为 mini-batch) ，来进行训练，而这会使方差变小。

Batch gradient descent moves directly downhill (locally speaking).  
SGD takes steps in a noisy direction, but moves downhill on average.



batch gradient descent



stochastic gradient descent

### SGD Learning Rate

在随机训练 (stochastic training) 中，学习率也会由于梯度的随机性而影响波动 (fluctuations) 。

训练策略：

- 在训练的早期使用高学习率，以便快速接最优解
- 逐渐降低学习率，减少 fluctuations