INT305 note

(Machine Learning)

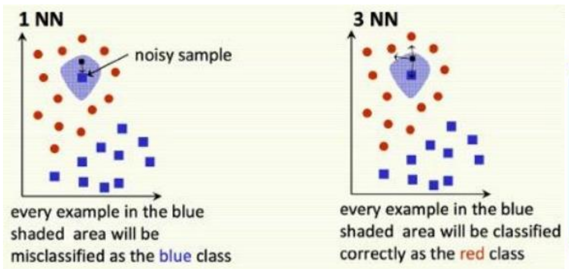
1 Introduction

1.1 Supervised learning (much of this course)

Task	Inputs	Labels
object recognition	image	object category
image captioning	image	caption
document classification	text	document category
speech-to-text	audio waveform	text
⋮	⋮	⋮

1.1.1 KNN

- Nearest neighbours sensitive to noise or mis-labelled data ("class noise").
- Smooth by having k nearest neighbours vote.

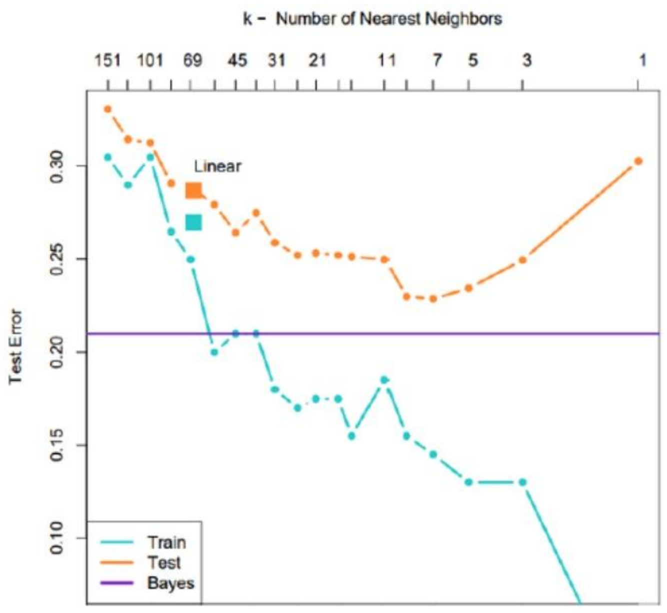

1 NN
noisy sample
every example in the blue shaded area will be misclassified as the blue class
3 NN
every example in the blue shaded area will be classified correctly as the red class

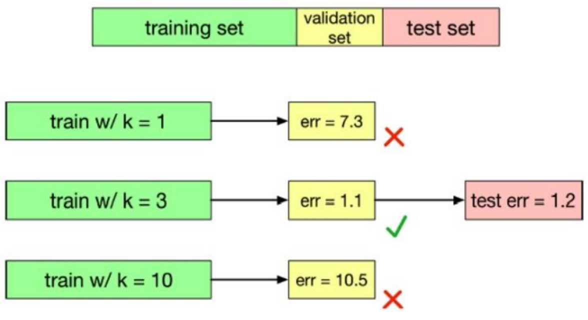
Algorithm (kNN):

1. Find k examples $\{\mathbf{x}^{(i)}, t^{(i)}\}$ closest to the test instance $\mathbf{x}$
2. Classification output is majority class

$$y = arg\max_{t^{(z)}} \sum_{i=1}^{k} \mathbb{I}(t^{(z)} = t^{(i)})$$

- Balancing hyperparameter k
  - ➢ Optimal choice of k depends on number of data points n.
  - ➢ Nice theoretical properties if $k \to \infty$ and $k/n \to 0$.
  - ➢ Rule of thumb: choose $k < \sqrt{n}$.
  - ➢ We can choose k using validation set.





2 Linear Methods for Regression, Optimization

Linear regression exemplifies recurring themes of this course:

- Choose a model and a loss function
- Formulate an optimization problem
- Solve the minimization problem using one of two strategies

- ➢ Direct solution (set derivatives to zero)
- ➢ Gradient descent

- Vectorize the algorithm, i.e. represents in terms of linear algebra
- Make a linear model more powerful using features
- Improve the generalization by adding a regularizer

2.1 Supervised Learning Setup

In supervised learning:

- There is input $\mathbf{x} \in \mathcal{X}$, typically a vector of features (or covariates)
- There is target $t \in \mathcal{T}$, (also called response, outcome, output, class)
- Objective is to learn a function $f: \mathcal{X} \rightarrow \mathcal{T}$ such that $t \approx y = f(\mathbf{x})$ based on some data $\mathcal{D} = \{(\mathbf{x}^{(i)}, t^{(i)})\ for\ i = 1, 2, \cdots, N\}$

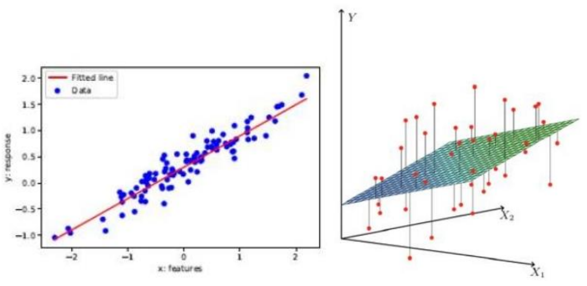
2.2 Linear Regression

2.2.1 Linear Regression Model

Model: In linear regression, we use a linear function of the features $\mathbf{x} = x_1, \cdots, x_D \in \mathbb{R}^D$ 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: $y \approx t$.



- If we have only 1 feature: $y = wx + b$ where $w, x, b \in \mathbb{R}$.
- y is linear in x.
- If we have only D features: $y = \mathbf{w}^\top \mathbf{x} + b$ where $\mathbf{w}, \mathbf{x} \in \mathbb{R}^D$, $b \in \mathbb{R}$
- y is linear in $\mathbf{x}$.

2.2.2 Linear Regression workflow

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

- $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \cdots x_D^{(i)})^\top \in \mathbb{R}^D$ 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 $\mathbf{x}^{(i)}$:
  - $t^{(i)} \approx y^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + b$
  - Different $\mathbf{w}, b$ define different lines.
  - We want the “best” line $\mathbf{w}, b$.

2.2.3 Linear Regression 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:

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

- $y - t$ is the residual, and we want to make this small in magnitude.
- The 1/2 factor is just to make the calculations convenient.

- Cost function: loss function averaged over all training examples.

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

2.2.3 Linear Regression Vectorization

But if we expand $y^{(i)}$, it will get messy:

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

Vectorize algorithms by expressing them in terms of vectors and matrices:

$$\mathbf{w}=(w_1,\cdots w_D)^\top \qquad \mathbf{x}=(x,\cdots x_D)^\top$$
$$y=\mathbf{w}^\top\mathbf{x}+b$$

Python code:

```python
y = b
for j in range(M):
    y += w[j] * x[j]
```

$\equiv$

```python
y = np.dot(w, x) + b
```

Organize all the training examples into a design matrix $\mathbf{X}$ with one row per training example, and all the targets into the target vector $\mathbf{t}$:

$$\mathbf{X}=\begin{pmatrix}\mathbf{x}^{(1)\top}\\ \mathbf{x}^{(2)\top}\\ \mathbf{x}^{(3)\top}\end{pmatrix}=\begin{array}{cccc|l} & \text{one feature across all training examples} & & & \\ 8 & \boxed{0} & 3 & 0 & \\ 6 & -1 & 5 & 3 & \text{one training example (vector)}\\ 2 & 5 & -2 & 8 & \end{array}$$

Computing the predictions for the whole dataset:

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

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:

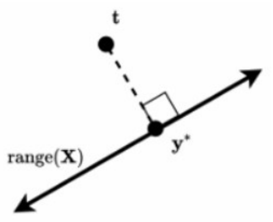
$$\mathbf{X}=\begin{bmatrix}1 & [\mathbf{x}^{(1)}]^\top\\ 1 & [\mathbf{x}^{(2)}]^\top\\ 1 & \vdots\end{bmatrix}\in\mathbb{R}^{N\times(D+1)} \quad \text{and} \quad \mathbf{w}=\begin{bmatrix}b\\ w_1\\ w_2\\ \vdots\end{bmatrix}\in\mathbb{R}^{D+1}$$

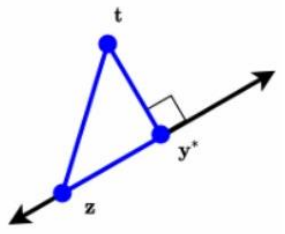
Then, our predictions reduce to $\mathbf{y}=\mathbf{X}\mathbf{w}$.

2.3 Direct Solution •

2.3.1 Linear Algebra •

- We seek $\mathbf{w}$ to minimize $||\mathbf{X}\mathbf{w}-\mathbf{t}||^2$, or equivalently $||\mathbf{X}\mathbf{w}-\mathbf{t}||$
- range $\mathbf{X}=\{\mathbf{X}\mathbf{w}|\mathbf{w}\in\mathbb{R}^D\}$ is a D-dimensional subspace of $\mathbb{R}^N$
- Recall that the closest point $\mathbf{y}^*=\mathbf{X}\mathbf{w}^*$ in subspace range($\mathbf{X}$) of $\mathbb{R}^N$ to arbitrary point $\mathbf{t}\in\mathbb{R}^N$ is found by orthogonal projection.
- We have $(\mathbf{y}^*-\mathbf{t})\perp\mathbf{X}\mathbf{w},\ \forall\mathbf{w}\in\mathbb{R}^D$
- $\mathbf{y}^*$ is the closest point to $\mathbf{t}$





2.3.2 Calculus •

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

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

- To compute, take the single variable derivative, pretending the other arguments are constant.
- Example: partial derivatives of the prediction y.

$$\frac{\partial y}{\partial w_j} = \frac{\partial}{\partial w_j}\left[\sum_{j'} w_{j'} x_{j'} + b\right] \qquad \frac{\partial y}{\partial b} = \frac{\partial}{\partial b}\left[\sum_{j'} w_{j'} x_{j'} + b\right]$$
$$= x_j \qquad\qquad\qquad\qquad = 1$$

- For loss derivatives, apply the chain rule:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_j} &= \frac{d\mathcal{L}}{dy}\frac{\partial y}{\partial w_j} \\ &= \frac{d}{dy}\left[\frac{1}{2}(y-t)^2\right] \cdot x_j \\ &= (y-t)x_j\end{aligned} \qquad \begin{aligned}\frac{\partial \mathcal{L}}{\partial b} &= \frac{d\mathcal{L}}{dy}\frac{\partial y}{\partial b} \\ &= y - t\end{aligned}$$

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

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

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

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

(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: $\mathbb{R}^D \to \mathbb{R}$", denoted $\nabla f(\mathbf{w})$, is:

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

- The gradient points in the direction of the greatest rate of increase.
- Analogue of second derivative (the "Hessian" matrix):

$$\nabla^2 f(\mathbf{w}) \in \mathbb{R}^{D\times D} \text{ is a matrix with} [\nabla^2 f(\mathbf{w})]_{ij} = \frac{\partial^2}{\partial w_i \partial w_j} f(\mathbf{w})$$

- We seek $\mathbf{w}$ to minimize $\mathcal{J}(\mathbf{w}) = ||\mathbf{X}\mathbf{w} - \mathbf{t}||^2 / 2$
- Taking the gradient with respect to $\mathbf{w}$ we get:

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

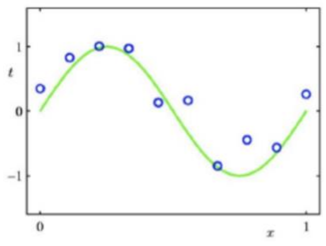
$$\mathbf{w}^* = (\mathbf{X}^{\top}\mathbf{X})^{-1}\mathbf{X}^{\top}\mathbf{t}$$

- Linear regression is one of only a handful of models in this course that permit direct solution.

2.4 Polynomial Feature Mapping

2.4.1 Introduction

The relation between the input and output may not be linear. But we can still use linear regression by mapping the input features to another space using feature mapping (or basis expansion). $\varphi(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^d$ and treat the mapped feature (in $\mathbb{R}^d$) as the input of a linear regression procedure.



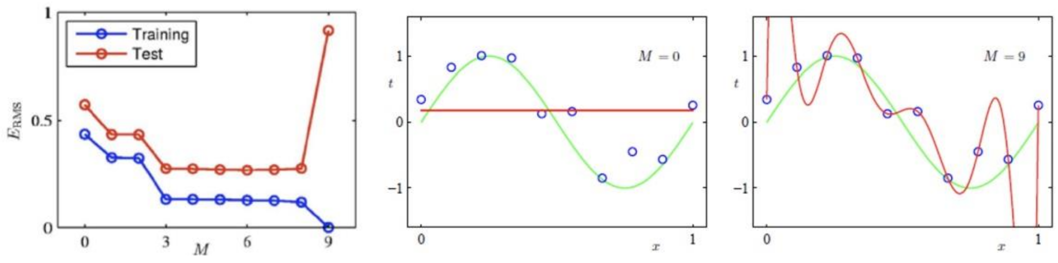
Find the data using a degree-M polynomial function of the form:

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

- Here the feature mapping is $\varphi(x) = [1, x, x^2, \ldots, x^M]^\top$.
- We can use linear regression to find $\mathbf{w}$, since $y = \varphi(x)^\top \mathbf{w}$ is linear with $w_0, w_1, \ldots, w_M$

2.4.2 **Model Complexity and Generalization**

- Underfitting (M=0): model is too simple – does not fit the data.
- Overfitting (M=9): model is too complex – fits perfectly.



2.4.3 **L² Regularization**

Regularizer: a function that quantifies how much we prefer one hypothesis VS another.

- We can encourage the weights to be small by choosing as our regularizer the L^2 penalty.

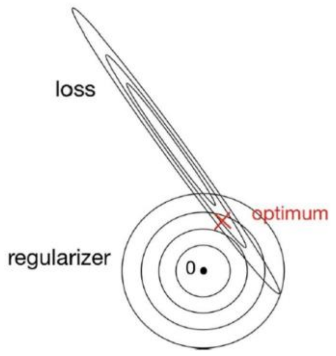
$$\mathcal{R}(\mathbf{w}) = \frac{1}{2}\sum_j w_j^2$$

(To be precise, the L^2 norm is Euclidean distance, we're regularizing the squared L^2 norm)

- The regularized cost function makes a tradeoff between fit to the data and the norm of the weights.

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

- If you fit training data poorly, $\mathcal{J}$ is large. If your optimal weights have high values, $\mathcal{R}$ is large.
- Large λ penalize wight values more.
- Like M, λ is a hyperparameter we can tune with a validation set.



2.4.4 **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:

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

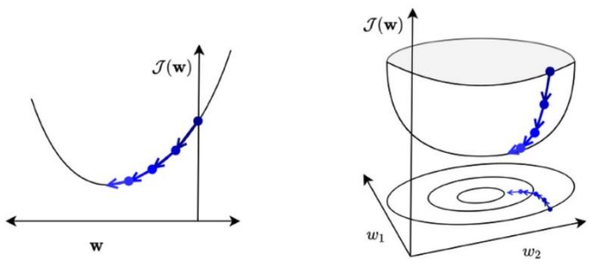
- The case $\lambda = 0$ (no regularization) reduces to least squares solution!

2.5 **Gradient Descent**

2.5.1 **Concepts**

- Many times, we do not have a direct solution: Taking derivatives of $\mathcal{J}$ w.r.t $\mathbf{w}$ and setting them to 0 doesn't have an explicit 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.



(就是等到斜率为 0 即为最优解)

- 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 α (unless $\partial\mathcal{J}/\partial w_j = 0$):
- $\alpha > 0$ is a learning rate (or step size). The larger it is, the faster $\mathbf{w}$ changes (but values are typically small).
- This gets its name from the gradient:

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

- Update rule in vector form:

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

- And for linear regression we have:

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

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

2.5.2 Gradient Descent for Linear Regression

- Why gradient descent, if we can find the optimum directly?
  - ➢ gradient descent can be applied to a much broader set of models
  - ➢ gradient descent can be easier to implement than direct solutions
  - ➢ For regression in high-dimensional space, gradient descent is more efficient than direct solution

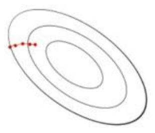
2.5.3 Gradient Descent under the L² Regularization

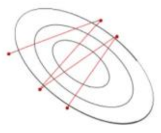
- The gradient descent update to minimize the L^2 regularized cost $\mathcal{J} + \lambda\mathcal{R}$ results in weight decay:

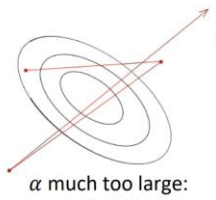
$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \alpha\frac{\partial}{\partial\mathbf{w}}(\mathcal{J} + \lambda\mathcal{R}) \\ &= \mathbf{w} - \alpha\left(\frac{\partial\mathcal{J}}{\partial\mathbf{w}} + \lambda\frac{\partial\mathcal{R}}{\partial\mathbf{w}}\right) \\ &= \mathbf{w} - \alpha\left(\frac{\partial\mathcal{J}}{\partial\mathbf{w}}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

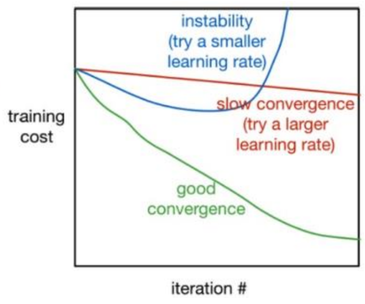
2.5.4 Learning Rate (Step Size)

- In gradient descent, the learning rate α is a hyperparameter we need to tune.
- Good values are typically between 0.001 and 0.1.


α too small: slow progress


α too large: oscillations


α much too large: instability



- To diagnose optimization problems, it's useful to look at training curves: plot the training cost as a function of iteration.
- Warning: in general, it's very hard to tell from the training curves whether an optimizer has converged. They can reveal major problems, but they can't guarantee convergence.

2.5.5 **Stochastic Gradient Descent**

Stochastic gradient descent (SGD): update the parameters based on the gradient for a single training example

1- Choose i uniformly at random

$$2\text{-}\ \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}$$

- Cost of each SGD update is independent of N!
- SGD can make significant progress before even seeing all the data!
- Mathematical justification: if you sample a training example uniformly at random, the stochastic gradient is an unbiased estimate of the batch gradient:

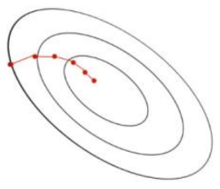
$$\mathbb{E}\left[\frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}\right] = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}}$$

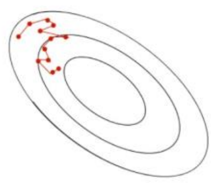
2.5.6 **Mini-batch Stochastic Gradient Descent**

- Problems with using single training example to estimate gradient:
  - ➢ Variance in the estimate may be high
  - ➢ We can't exploit efficient vectorized operations
- Compromise approach:
  - ➢ Compute the gradients on a randomly chosen medium-sized set of training examples $\mathcal{M} \subset \{1, \cdots, N\}$ called a mini-batch.
- Stochastic gradients computed on larger mini-batches have smaller variance.
- The mini-batch size $|\mathcal{M}|$ is a hyperparameter that needs to be set.
  - ➢ Too large: requires more compute: e.g., it takes more memory to store the activations, and longer to compute each gradient update
  - ➢ Too small: can't exploit vectorization, has high variance
  - ➢ A reasonable value might be $|\mathcal{M}| = 100$.

2.5.7 **Comparation**

- 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

▲ Batch Gradient Descent，全批量梯度下降，是最原始的形式，它是指在每一次迭代时使用所有样本来进行梯度的更新。优点是全局最优解，易于并行实现；缺点是当样本数目很大时，训练过程会很慢。

▲ Stochastic Gradient Descent，随机梯度下降，是指在每一次迭代时使用一个样本来进行参数的更新。优点是训

练速度快；缺点是准确度下降，并且可能无法收敛或者在最小值附近震荡。
▲ **Mini-Batch Gradient Descent**，小批量梯度下降，是对上述两种方法的一个折中办法。它是指在每一次迭代时使用一部分样本来进行参数的更新。这种方法兼顾了计算速度和准确度。