02a.paradigms-knn, Tutorial 01 - The Learning Paradigms, and k-NN

Supervised. In supervised learning, for each instance, we get a correct output.

Unsupervised. In unsupervised learning, we have no outputs (labels) given to us (the task is to explore and make groupings from the data based solely on the input).

Reinforcement. In reinforcement learning, we get some form of output but it does not correspond directly to the correct output for each instance. For example, we could get a single output after a batch of instances is processed (as commonly seen in games, where the eventual game win/loss outcome is the eventual output of a series of instance moves). For example, as a Year 1 student, I would like to choose modules that maximize my starting income upon graduation. If we consider the choice of modules during each of the nominal eight semesters as a problem instance, this would constitute a reinforcement learning problem. In this final task, the input might be the historical enrollment records of past students (and possibly their performance within each module), and the output would be their starting salary. Data is the set of inputs that correspond to the resultant output.

在强化学习中，我们得到某种形式的输出，但它并不直接对应于每个实例的正确输出。例如，我们可以在处理一批实例后获得单个输出（正如在游戏中常见的那样，最终的游戏赢/输结果是一系列实例移动的最终输出）。例如，作为一名一年级学生，我想选择能够最大化我毕业后起始收入的模块。如果我们将标称八个学期中每个学期的模块选择视为一个问题实例，这将构成一个强化学习问题。在这个最终任务中，输入可能是过去学生的历史入学记录（可能还有他们在每个模块中的表现），输出可能是他们的起薪。 数据是对应于结果输出的一组输入。

k-Nearest Neighbors

https://zhuanlan.zhihu.com/p/71646003

https://en.wikipedia.org/wiki/K-d_tree#Complexity

Complexity [edit]

- Building a static k-d tree from n points has the following worst-case complexity:
  - $O(n \log^2 n)$ if an $O(n \log n)$ sort such as Heapsort or Mergesort is used to find the median at each level of the nascent tree;
  - $O(n \log n)$ if an $O(n)$ median of medians algorithm[3][4] is used to select the median at each level of the nascent tree;
  - $O(kn \log n)$ if n points are presorted in each of k dimensions using an $O(n \log n)$ sort such as Heapsort or Mergesort prior to building the k-d tree.[8]
- Inserting a new point into a balanced k-d tree takes $O(\log n)$ time.
- Removing a point from a balanced k-d tree takes $O(\log n)$ time.
- Querying an axis-parallel range in a balanced k-d tree takes $O(n^{1-1/k}+m)$ time, where m is the number of the reported points, and k the dimension of the k-d tree.
- Finding 1 nearest neighbour in a balanced k-d tree with randomly distributed points takes $O(\log n)$ time on average.

04a.linear

Income	Credit History	Debt	Decision
0 – 5K	Bad	Low	Reject
0 – 5K	Good	Low	Approve
0 – 5K	Unknown	High	Reject
0 – 5K	Unknown	Low	Approve
0 – 5K	Unknown	Low	Approve
0 – 5K	Unknown	Low	Reject
5 – 10K	Bad	High	Reject
5 – 10K	Good	High	Approve
5 – 10K	Unknown	High	Approve
5 – 10K	Unknown	Low	Approve
Over 10K	Bad	Low	Reject
Over 10K	Good	Low	Approve

Table 2: Loan processing records

(a)

Entropy : $-\frac{7}{12}\log_2\frac{7}{12}-\frac{5}{12}\log_2\frac{5}{12}=0.98$

Entropy of Income : $\frac{6}{12}(-\frac{3}{6}\log_2\frac{3}{6}-\frac{3}{6}\log_2\frac{3}{6})$ (0-5k) $+\frac{4}{12}(-\frac{3}{4}\log_2\frac{3}{4}-\frac{1}{4}\log_2\frac{1}{4})$ (5-10k) $+\frac{2}{12}(-\frac{1}{2}\log_2\frac{1}{2}-\frac{1}{2}\log_2\frac{1}{2})$ (Over 10k) $=0.937$

Gain Ratio : $(0.98-0.937)/(-\frac{6}{12}\log_2\frac{6}{12}-\frac{4}{12}\log_2\frac{4}{12}-\frac{2}{12}\log_2\frac{2}{12})=0.007$

Entropy of Credit History: 0 (Bad) +0 (Bad) $+\frac{6}{12}(-\frac{4}{6}\log_2\frac{4}{6}-\frac{2}{6}\log_2\frac{2}{6})$ (Unknown) $=0.46$ ✓

Gain Ratio : $(0.98-0.46)/(-\frac{3}{12}\log_2\frac{3}{12}-\frac{3}{12}\log_2\frac{3}{12}-\frac{6}{12}\log_2\frac{6}{12})=0.346$

Entropy of Debt: $\frac{8}{12}(-\frac{5}{8}\log_2\frac{5}{8}-\frac{3}{8}\log_2\frac{3}{8})$ (Low) $+\frac{4}{12}(-\frac{2}{4}\log_2\frac{2}{4}-\frac{2}{4}\log_2\frac{2}{4})$ (High) $=0.97$

Gain Ratio : $(0.98-0.97)/(-\frac{8}{12}\log_2\frac{8}{12}-\frac{4}{12}\log_2\frac{4}{12})=0.01$

Reject Approve $\frac{4}{6}$ Approve, $\frac{2}{6}$ Reject

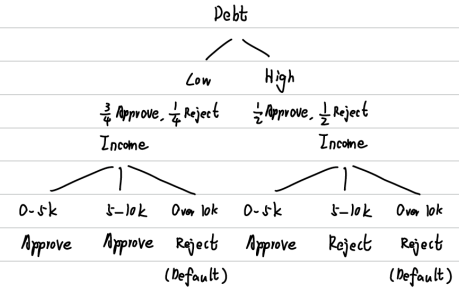
Entropy : $-\frac{4}{6}\log_2\frac{4}{6}-\frac{2}{6}\log_2\frac{2}{6}=0.92$

Entropy of Income : $\frac{4}{6}(-\frac{2}{4}\log_2\frac{2}{4}-\frac{2}{4}\log_2\frac{2}{4})$ (0-5k) $+\frac{2}{6}(-\frac{1}{2}\log_2\frac{1}{2}-\frac{1}{2}\log_2\frac{1}{2})$ (5-10k) $=1$

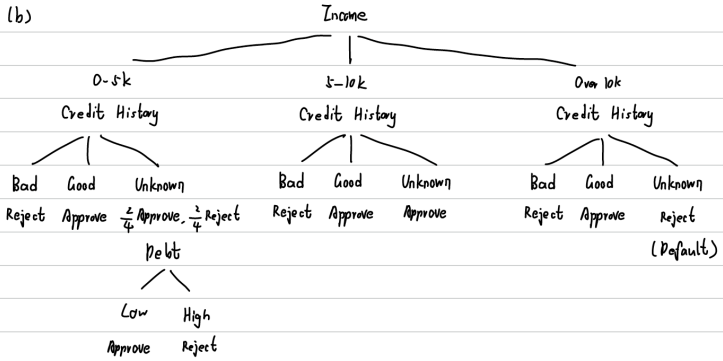
Gain Ratio : $0.92-1<0$

Entropy of Debt: $\frac{4}{6}(-\frac{3}{4}\log_2\frac{3}{4}-\frac{1}{4}\log_2\frac{1}{4})$ (Low) $+\frac{2}{6}(-\frac{1}{2}\log_2\frac{1}{2}-\frac{1}{2}\log_2\frac{1}{2})$ (High) $=0.874$ ✓

Reject Approve $\frac{4}{6}$ Approve, $\frac{2}{6}$ Reject



(b)



07b.evaluations, Tutorial 05 Evaluation Metrics

2. **Micro- and Macro-Averaging**

(For this question, you can try coding out the formulas and verify that the answers match!)

We have a classifier trained to predict images of cats, dogs, and pigs. The confusion matrix for the model is given below.

		Actual		
		Dog	Cat	Pig
Predicted	Dog	10	2	1
	Cat	3	13	2
	Pig	3	4	7

(a) Create the confusion matrix for each of the individual classes, dog, cat, and pig.

To generate the individual confusion matrices, we take the target class as positive examples, and all the other classes as negative examples.

		Actual	
		Positive	Negative
Predicted	Positive	10	3
	Negative	6	26

Dog

		Actual	
		Positive	Negative
Predicted	Positive	13	5
	Negative	6	21

Cat

		Actual	
		Positive	Negative
Predicted	Positive	7	7
	Negative	3	28

Pig

(b) Calculate the micro-average confusion matrix, accuracy, precision, recall, and F_1 score. What do you notice about the precision, recall and F_1 score? Why is this so?

To generate the micro-average confusion matrices, we take the sum of all the individual confusion matrices.

Micro-average	Actual: Positive	Actual: Negative
Predicted: Positive	10	5
Predicted: Negative	5	25

先相加得到 Micro-average

$Accuracy_{Micro}$ = (TP+TN)/(TP+TN+FP+FN) = (10+25)/(10+25+5+5) = 0.778
$Precision_{Micro}$ = TP/(TP+FP) = 10/(10+5) = 0.667
$Recall_{Micro}$ = TP/(TP+FN) = 10/(10+15) = 0.667
$F1_{Micro}$ score = $((P^{-1} + R^{-1})/2)^{-1} = ((0.667^{-1} + 0.667^{-1})/2)^{-1} = 0.667$
We notice that the precision, recall, and F_1 score all has the same value. This occurs because the $FP_{Micro} = FN_{Micro}$. This is true not just for this particular case, but also generalises to all other confusion matrices.
We see that for a general 3 class confusion matrix,

	Actual: Class 1	Actual: Class 2	Actual: Class 3
Predicted: Class 1	a	b	c
Predicted: Class 2	d	e	f
Predicted: Class 3	g	h	i

$FP_{Micro} = (FP_{Class1} + FP_{Class2} + FP_{Class3})/3 = ((b+c) + (d+f) + (g+h))/3$
$FN_{Micro} = (FN_{Class1} + FN_{Class2} + FN_{Class3})/3 = ((d+g) + (b+h) + (c+f))/3$
Rearranging the terms, we can easily see that FP_{Micro} is equal to FN_{Micro}, leading to $Precision_{Micro} = Recall_{Micro}$.
$F1_{Micro}$ score = $((P^{-1} + R^{-1})/2)^{-1} = ((P^{-1} + P^{-1})/2)^{-1} = ((P^{-1})^{-1} = P$
Therefore, for micro-averaging of 3 classes, $Recall_{Micro} = Precision_{Micro} = F1_{Micro}$
This can be extended for confusion matrices with any number of classes.

(c) Calculate the macro-average precision and recall.
$Precision_{Dog}$ = 10/(10+3) = 0.769
$Precision_{Cat}$ = 13/(13+5) = 0.722
$Precision_{Pig}$ = 7/(7+7) = 0.5
$Precision_{Macro} = (Precision_{Dog} + Precision_{Cat} + Precision_{Pig})/3 = 0.664$
$Recall_{Dog}$ = 10/(10+6) = 0.625
$Recall_{Cat}$ = 13/(13+6) = 0.684
$Recall_{Pig}$ = 7/(7+3) = 0.7
$Recall_{Macro} = (Recall_{Dog} + Recall_{Cat} + Recall_{Pig})/3 = 0.603$

(d) Consider the following scenario in table 3 where there is a huge class imbalance.

Class	TP	FP
A	9	1
B	100	900
C	9	1
D	9	1

Table 3: Class imbalance Data

07c.evaluations, 08a.data-processing, 08b.feature-engineering, tutorial-06:

Assume that you are a supply-chain manager. You are using MAPE to judge your regression forecasts about **product demands** for the next month. We list them here. Discuss whether the listed MAPE shortcomings below will or will not affect you.

分母为0不可除，分母接近0会趋向无穷

i. Data with zeroes or close to zeroes. $\left|\frac{1-10000}{1}\right| = 9999$ 无上限

ii. Heavier penalty when predictions are higher than actual data. $\left|\frac{10000-1}{10000}\right| = 0.9999 \approx 1$ 再小也是>0

iii. Assumption that zero in the data's unit of measurement holds meaning.

i. Yes, it will affect. Product demand can reach values of zero. MAPE produces undefined or infinite values when the actual values(y_i) are zero or close to zero. 真实值为0

ii. Yes, it affects as product demand cannot be negative values. MAPE is asymmetric when forecasts are strictly non-negative. It places a heavier penalty when forecasts($\hat{y}_i$) are higher than actuals(y_i). Percentage error cannot exceed 100% for low forecasts, but there are no upper limits for high forecasts as our product demands cannot be a negative value.

iii. No, this is not a shortcoming for us. MAPE assumes that when the value is zero, it is meaningful. In our case, forecasts for product demand being zero would cause MAPE to return 100% if our actual is non-zero. For units of measurements that have arbitrary zero values, using MAPE no longer makes sense.

前提是真实值不为0，那预测值为0是有意义的 MAPE = 100%

(c) Finally, we define an alternative to MAPE, which is SMAPE (Symmetric-MAPE). We define SMAPE as follows.

$$SMAPE = \frac{100\%}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{(|\hat{y}_i| + |y_i|)/2}$$

Analyze how SMAPE fixes MAPE's problems.

As its name suggests, SMAPE overcomes the asymmetry of negative and positive errors of MAPE. Its formulation results in a fixed lower bound of 0% and upper bound of 200%.

Can you think which shortcomings still persist from MAPE? What other shortcomings does SMAPE introduce?

$\frac{|1-1|}{(|1|+|1|)/2} = 0\%$

$\frac{|1-(-1)|}{(|1|+|-1|)/2} = 200\%$

MAE	https://www.statology.org/mean-absolute-error-calculator/
MSE	https://www.statology.org/mse-calculator/
RMSE	https://www.statology.org/rmse-calculator/
MAPE	https://www.statology.org/mape-calculator/
SMAPE	https://rdrr.io/cran/Metrics/man/smape.html

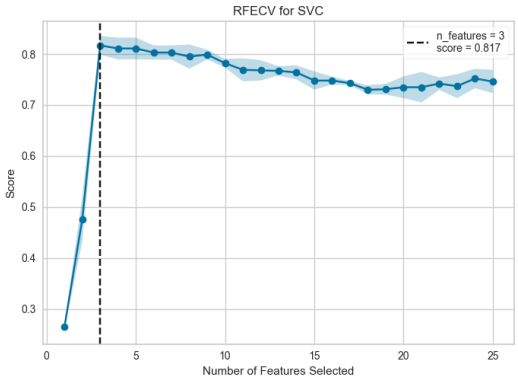
Feature Selection (Wrapper): Recursive Feature Elimination (RFE)

5. 反向特征消除（Backward Feature Elimination）

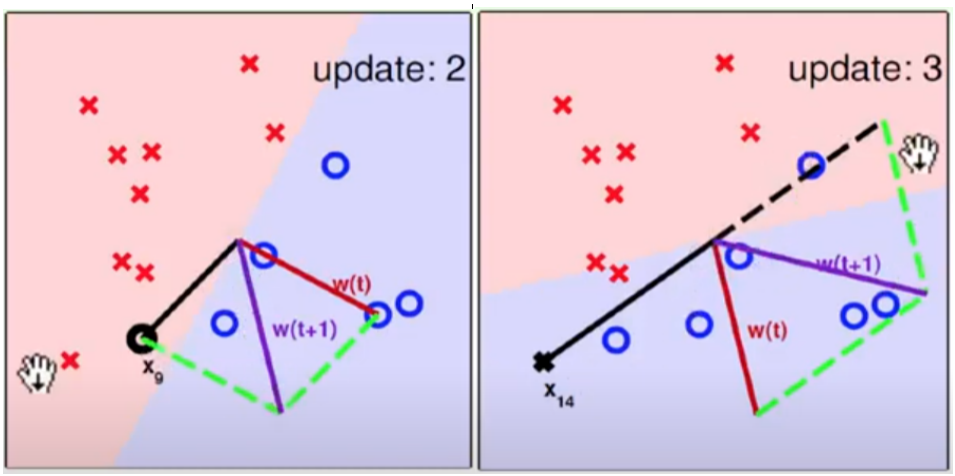
以下是反向特征消除的主要步骤：

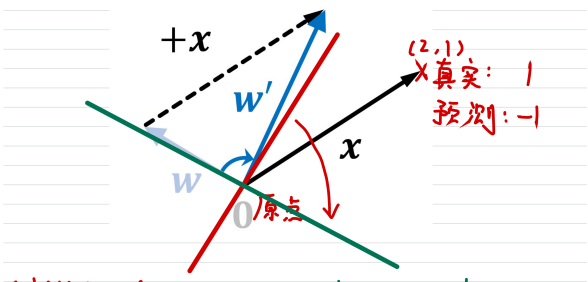
- 先获取数据集中的全部n个变量，然后用它们训练一个模型。
- 计算模型的性能。
- 在删除每个变量（n次）后计算模型的性能，即我们每次都去掉一个变量，用剩余的n-1个变量训练模型。
- 确定对模型性能影响最小的变量，把它删除。
- 重复此过程，直到不再能删除任何变量。

The graph peaked when only the top 3 features are considered. The score slowly decreases as additional features are added. **This is due to the fact that the latter features do not provide any further information. It is possible that they are a source of noise**, explaining the decrease in the score as more features are added.



09a.perceptron-neural-networks, 09b.backprop-ama, tutorial-07:





分割线A → $(w \cdot x) < 0 \Rightarrow -1$ ✗

$x - y = 0$

$w = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

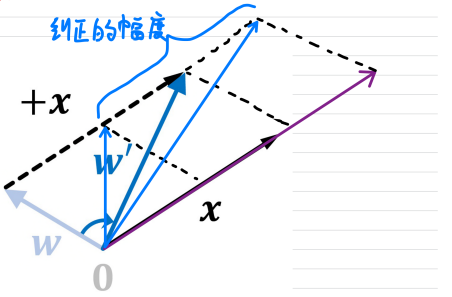
$w \cdot x = \begin{bmatrix} -1 \\ 1 \end{bmatrix} [2,1] = -1 < 0$

分割线B → $(w' \cdot x) > 0 \Rightarrow 1$ ✓

$x + 2y = 0$

$w' = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$w' \cdot x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} [2,1] = 4 > 0$



1. 注意 w_{in} 下标

第一个数：

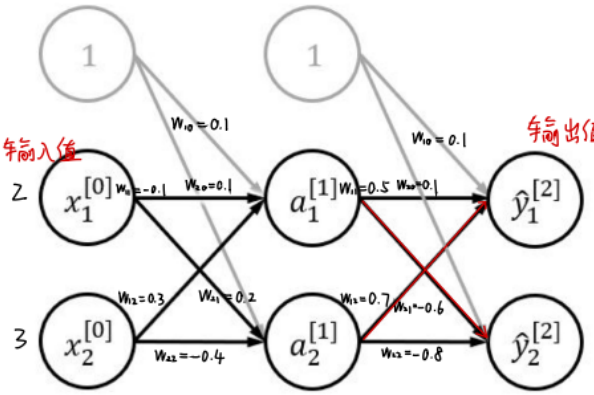
第二个数：

a_1的入边 a_2的入边

$$W^{[1]} = \begin{bmatrix} w_{10}=0.1 & w_{20}=0.1 \\ w_{11}=-0.1 & w_{21}=0.2 \\ w_{12}=0.3 & w_{22}=-0.4 \end{bmatrix}$$

$\hat{y}_1$的入边 $\hat{y}_2$的入边

$$W^{[2]} = \begin{bmatrix} w_{10}=0.1 & w_{20}=0.1 \\ w_{11}=0.5 & w_{21}=-0.6 \\ w_{12}=0.7 & w_{22}=-0.8 \end{bmatrix}$$



$\hat{y}$是预测值，y是实际值

真实值

$y_1 = 0.1$

$y_2 = 0.9$

(a) ReLU：$f(x) = \max(0, x)$，本题每一层都用了激活函数 ReLU

a_1：$1\times0.1 + 2\times(-0.1) + 3\times0.3 = 0.8$ 输入值$(1, X_1, X_2)$分别乘以a_1入边的权重，$ReLU(0.8) = 0.8$

a_2：$1\times0.1 + 2\times0.2 + 3\times(-0.4) = -0.9$ 输入值$(1, X_1, X_2)$分别乘以a_2入边的权重，$ReLU(-0.9) = 0$

$\hat{y}_1$：$1\times0.1 + 0.8\times0.5 + 0\times0.7 = 0.5$ 输入值$(1, a_1, a_2)$分别乘以$\hat{y}_1$入边的权重，$ReLU(0.5) = 0.5$

$\hat{y}_2$：$1\times0.1 + 0.8\times(-0.6) + 0\times(-0.8) = -0.38$ 输入值$(1, a_1, a_2)$分别乘以$\hat{y}_2$入边的权重，$ReLU(-0.38) = 0$

$$L(\hat{y}, y) = \frac{1}{2}\left((\underbrace{0.5}_{\hat{y}_1} - \underbrace{0.1}_{y_1})^2 + (\underbrace{0}_{\hat{y}_2} - \underbrace{0.9}_{y_2})^2\right) = 0.485$$

两个数 $\hat{y}_1, \hat{y}_2$

(b) Suppose we already know that $\frac{\partial L(\hat{y}^{[2]}, y)}{\partial \hat{y}_1^{[2]}} = 0.5$, $\frac{\partial L(\hat{y}^{[2]}, y)}{\partial \hat{y}_2^{[2]}} = 0.3$, $a_1^{[1]} = 0.5, a_2^{[1]} = 0.4, \hat{y}_1^{[2]} > 0, \hat{y}_2^{[2]} > 0$ Calculate the following gradient (partial derivative):
$L(\hat{y}^{[2]}, y)$ with respect to $W_{21}^{[2]}$ and $L(\hat{y}^{[2]}, y)$ with respect to $W_{12}^{[2]}$.

(b) $\frac{\partial L(\hat{y}, y)}{\partial \hat{y}}$：对损失函数求预测值的偏导

$\hat{y}_2 = 1\times w_{20} + 0.5\times w_{21} + 0.4\times w_{22}$

$$\frac{\partial L(\hat{y}, y)}{\partial w_{21}} = \frac{\partial L(\hat{y}, y)}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial w_{21}} = 0.7\times0.5 = 0.15$$

中间少了一项对激活函数求导（ReLU求导=1）

$\hat{y}_1 = 1\times w_{10} + 0.5\times w_{11} + 0.4\times w_{12}$

$$\frac{\partial L(\hat{y}, y)}{\partial w_{21}} = \frac{\partial L(\hat{y}, y)}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial w_{12}} = 0.5\times0.4 = 0.2$$

2. Perceptrons

(a) Model AND, OR, and NOT logic functions using a perceptron. Assume AND, and OR functions take 2 inputs where while the NOT functions takes a single output. Additionally, is it possible to model XOR function using a single Perceptron? Comment on your answer.

(b) Model XOR function(takes 2 inputs) using a number of perceptrons which implement AND, OR, and NOT functions. Show the diagram of the final Perceptron network. Clearly specify the weights of your network.

(c) Can the following function in Figure 1 be expressed with a 3-layer perceptron?

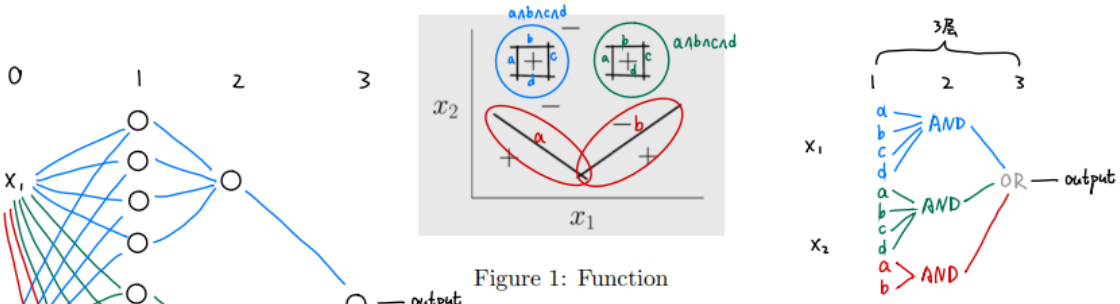
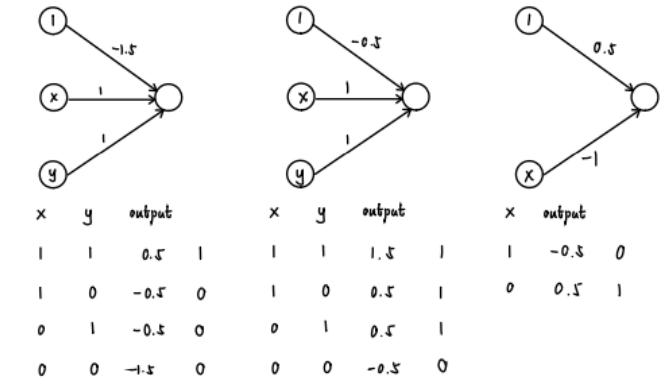


Figure 1: Function

2. XOR: 若 a,b 不同，结果为 1；若 a,b 相同，结果为 0



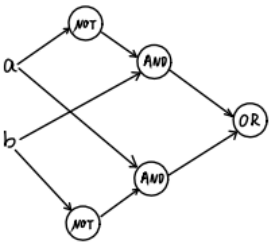
x	y	output	
1	1	0.5	1
1	0	-0.5	0
0	1	-0.5	0
0	0	-1.5	0

x	y	output	
1	1	1.5	1
1	0	0.5	1
0	1	0.5	1
0	0	-0.5	0

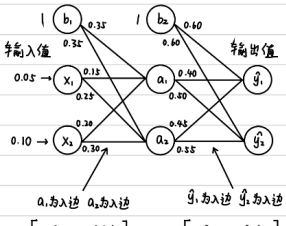
x	output	
1	-0.5	0
0	0.5	1

$a \oplus b = (\neg a \wedge b) \vee (a \wedge \neg b)$

$= OR(AND(NOT(a), b), AND(a, NOT(b)))$



3. Yes



$$W^1 = \begin{bmatrix} 0.35 & 0.35 \\ 0.15 & 0.25 \\ 0.20 & 0.30 \end{bmatrix} \qquad W^2 = \begin{bmatrix} 0.60 & 0.60 \\ 0.40 & 0.50 \\ 0.45 & 0.55 \end{bmatrix}$$

第一个数代表第几个神经元，1开始

第二个数代表当前神经元的第几条入边（权重），0开始

Forward Propagation 向前传播

输入层 → 隐藏层

$net a_1 = b_1 w_{10}^1 + X_1 w_{11}^1 + X_2 w_{12}^1$

$= 1 \times 0.35 + 0.05 \times 0.15 + 0.10 \times 0.20$

$= 0.3775$

$net a_2 = b_1 \times w_{20}^1 + X_1 w_{21}^1 + X_2 w_{22}^1$

$= 1 \times 0.35 + 0.05 \times 0.25 + 0.10 \times 0.30$

$= 0.3925$

这里用 sigmoid 作为激活函数，也可选其它如 ReLU

$out a_1 = \frac{1}{1+e^{-net a_1}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$

$out a_1 = \frac{1}{1+e^{-net a_1}} = \frac{1}{1+e^{-0.3925}} = 0.596884378$

隐藏层 → 输出层

$net \hat{y}_1 = b_2 w_{10}^2 + out a_1 w_{11}^2 + out a_2 w_{12}^2$

$= 1 \times 0.60 + 0.593269992 \times 0.40 + 0.596884378 + 0.45$

$= 1.105905967$

$net \hat{y}_2 = b_2 w_{20}^2 + out a_1 w_{21}^2 + out a_2 w_{22}^2$

$= 1 \times 0.60 + 0.593269992 \times 0.50 + 0.596884378 + 0.55$

$= 1.224921404$

$out \hat{y}_1 = \frac{1}{1+e^{-net \hat{y}_1}} = \frac{1}{1+e^{-1.105905967}} = 0.751365507$

$out \hat{y}_2 = \frac{1}{1+e^{-net \hat{y}_2}} = \frac{1}{1+e^{-1.224921404}} = 0.772928465$

计算误差损失值 $L(y_i, \hat{y}_i)$

$E_{total} = \frac{1}{2} \sum_{i=1} (y_i - out \hat{y}_i)^2$ （$\frac{1}{2}$是公式）

$= \frac{1}{2}[(y_1 - out \hat{y}_1)^2 + (y_2 - out \hat{y}_2)^2]$

求导 $= \frac{1}{2}[(0.01 - 0.751365507)^2 + (0.99 - 0.772928465)^2]$

$= 0.298371109$

求导

sigmoid 求导 $f'(x) = (\frac{1}{1+e^{-x}})'$

$= \frac{e^x}{(1+e^{-x})^2}$

$= \frac{1+e^x-1}{(1+e^{-x})^2}$

$= \frac{1}{(1+e^{-x})}[1 - \frac{1}{(1+e^{-x})}]$

$= f(x)(1-f(x))$

ReLU 求导 $f'(x) = x' = 1$

Back Propagation 反向传播

隐藏层 → 输出层的权值更新

$\frac{\partial E_{total}}{\partial w_{11}^2} = \frac{\partial E_{total}}{\partial out \hat{y}_1} \cdot \frac{\partial out \hat{y}_1}{\partial net \hat{y}_1} \cdot \frac{\partial net \hat{y}_1}{\partial w_{11}^2}$

$\frac{\partial E_{total}}{\partial out \hat{y}_1} = -2 \cdot \frac{1}{2}(y_1 - out \hat{y}_1)$

$= -(0.01 - 0.751365507)$

$= 0.741365507$

$\frac{\partial out \hat{y}_1}{\partial net \hat{y}_1} = out \hat{y}_1 (1 - out \hat{y}_1)$

$= 0.751365507 (1 - 0.751365507)$

$= 0.186815602$

若使用 ReLU

① $\Rightarrow out \hat{y}_1 = 0$ ($net \hat{y}_1 \leq 0$ 时)

$\Rightarrow \frac{\partial E_{total}}{\partial w_{11}} = 0$

② $\Rightarrow out \hat{y}_1 = net \hat{y}_1$ ($net \hat{y}_1 > 0$ 时)

$\Rightarrow$ 意味着这步求导可省略 ($f'(x) = 1$)

$\Rightarrow \frac{\partial E_{total}}{\partial w_{11}} = \frac{\partial E_{total}}{\partial net \hat{y}_1} \cdot \frac{\partial net \hat{y}_1}{\partial w_{11}}$

$\frac{\partial net \hat{y}_1}{\partial w_{11}^2} = out a_1 = 0.593269992$

$\therefore \frac{\partial E_{total}}{\partial w_{11}^2} = 0.741365507 \times 0.186815602 \times 0.593269992 = 0.082167041$

更新 w_{11}^2 的权值：$W_{11}^2 = W_{11}^2 - \eta \frac{\partial E_{total}}{\partial w_{11}^2} = 0.4 - 0.5 \times 0.082167041 = 0.358916480$ （η 是学习率，这里设为 0.5）

输入层 → 隐藏层的权值更新

$\frac{\partial E_{total}}{\partial w_{11}^1} = \frac{\partial E_{total}}{\partial out a_1} \cdot \frac{\partial out a_1}{\partial net a_1} \cdot \frac{\partial net a_1}{\partial w_{11}^1}$

$\frac{\partial E_{total}}{\partial net \hat{y}_1} = \frac{\partial E_{total}}{\partial out \hat{y}_1} \cdot \frac{\partial out \hat{y}_1}{\partial net \hat{y}_1}$

$= 0.741365507 \times 0.186815602$

$= 0.138499858$

$\frac{\partial E_{total}}{\partial net \hat{y}_2} = \frac{\partial E_{total}}{\partial out \hat{y}_2} \cdot \frac{\partial out \hat{y}_2}{\partial net \hat{y}_2}$

（原理同左）$= -0.217071835 \times 0.175510053$

$= -0.380982366$

$\frac{\partial net \hat{y}_1}{\partial out a_1} = W_{11}^2 = 0.40$

$\frac{\partial net \hat{y}_2}{\partial out a_1} = W_{21}^2 = 0.50$

$\frac{\partial E_{total}}{\partial out a_1} = \frac{\partial E_{total}}{\partial out \hat{y}_1} \cdot \frac{\partial out \hat{y}_1}{\partial net \hat{y}_1} \cdot \frac{\partial net \hat{y}_1}{\partial out a_1}$

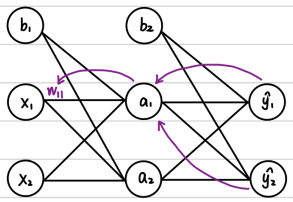
$= 0.138499858 \times 0.40$

$= 0.055399425$

$\frac{\partial E_{total}}{\partial out a_1} = \frac{\partial E_{total}}{\partial out \hat{y}_2} \cdot \frac{\partial out \hat{y}_2}{\partial net \hat{y}_2} \cdot \frac{\partial net \hat{y}_2}{\partial out a_1}$

$= -0.380982366 \times 0.50$

$= -0.190491183$



a_1 会接受从 $\hat{y}_1$ 和 $\hat{y}_2$ 两个地方传来的误差

$\frac{\partial E_{total}}{\partial out a_1} = 0.055399425 + (-0.190491183) = 0.036350306$

$\frac{\partial out a_1}{\partial net a_1} = out a_1 (1 - out a_1)$

$= 0.593269992 (1 - 0.593269992)$

$= 0.241300709$

$\frac{\partial net a_1}{\partial w_{11}^1} = X_1 = 0.05$

$\therefore \frac{\partial E_{total}}{\partial w_{11}^1} = 0.036350306 \times 0.241300709 \times 0.05 = 0.000438568$

更新 w_{11}^1 的权值：$W_{11}^1 = W_{11}^1 - \eta \frac{\partial E_{total}}{\partial w_{11}} = 0.15 - 0.5 \times 0.000438568 = 0.149780716$ （η 是学习率，这里设为 0.5）

最后，更新完所有权值后用新的权值重新计算，E_{total} 会不断下降

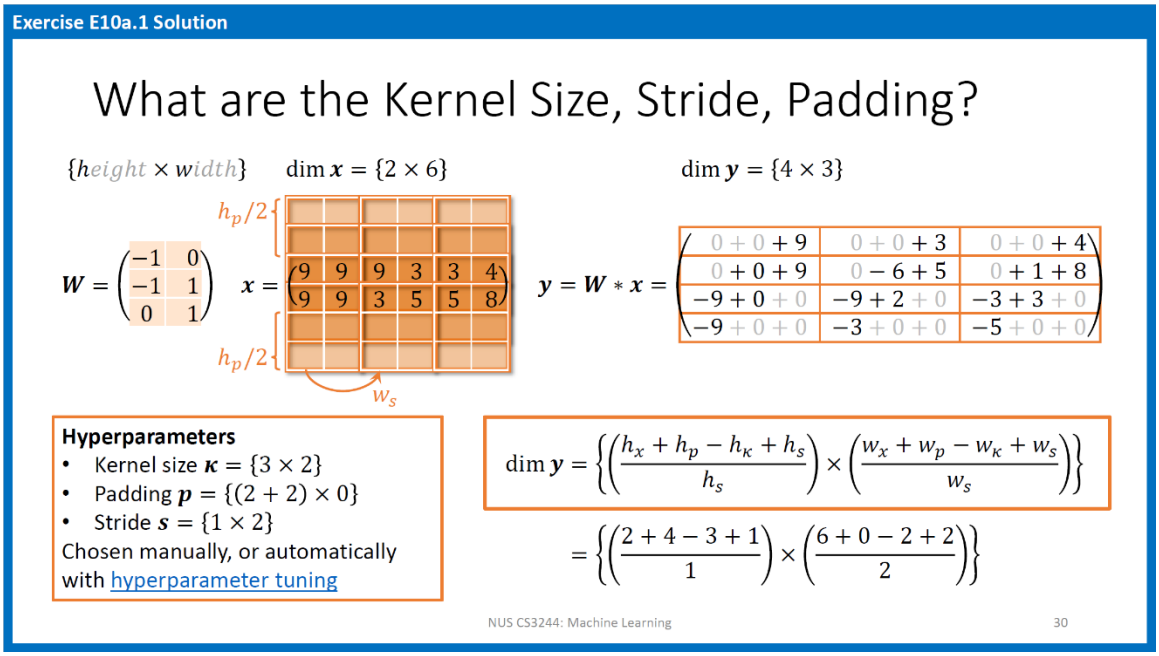
10a.deep-learning-cnn, Tutorial 08 - Deep Learning

feature maps 其实就是一个理论意义上的图片，可能是 512 * 512 * 3，可能是 256 * 256 * 10，就是长，宽，通道数的意思，但是就是每次经过卷积后的一个结果。

然后每次卷积前会把这个 feature maps（这个图）丢进去，然后用 kernel 扫描，每个 kernel 扫描这个整个 feature maps（这个图）就会得到一个 2 维的结果，因此这个 kernel 扫描的过程是扫描全部的通道然后再矩阵相加，所以最后成了 2D 的 256 * 256 * 1 的东西。

然后有 n 个 kernel，例如 10 个，每个 kernel 扫描后的结果叠加一起，就成了 256 * 256 * 10。

不断地重复这个 3d 到 2d 又到 3d 又到 2d 又到 3d 过程。



卷积的原理

卷积尺寸变化

输出尺寸=(输入尺寸-filter尺寸+2*padding)/stride+1

宽和高都是这么计算的；

输入图片大小为200×200，依次经过一层卷积（kernel size 5×5，padding 1，stride 2），pooling（kernel size 3×3，padding 0，stride 1），又一层卷积（kernel size 3×3，padding 1，stride 1）之后，输出特征图大小为：97

计算尺寸不被整除只在GoogLeNet中遇到过。卷积向下取整，池化向上取整。

(200-5+21) /2+1 为99.5，取99

(99-3) /1+1 为97

(97-3+21) /1+1 为97

常见的：

stride为1,kernel为 3 ,padding为1 卷积前后尺寸不变

stride为1,kernel为 5 ,padding为2 卷积前后尺寸不变

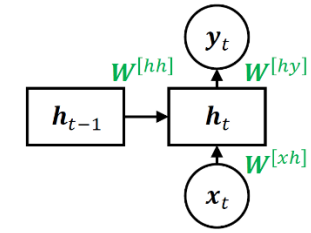
一个 batch(B)的 shape = B(可取 8, 16, 32, 64 等) X H(图像高度) X W(图像宽度) X C(通道数)

在 CNN 中，图像上堆叠两个卷积层，每个卷积层用的 3X3 kernel 扫描，而不是一个卷积层，用 5X5 kernel 扫描？一个 5X5 kernel 有 25 个参数，两个 3X3 kernels 有 18 个参数（一个 3X3 kernel 有 9 个

参数），因此两个 3X3 kernels 的运算速度更快，而且一个 5X5 kernel 和两个 3X3 kernels 的视野（步长为 1）是一样的。

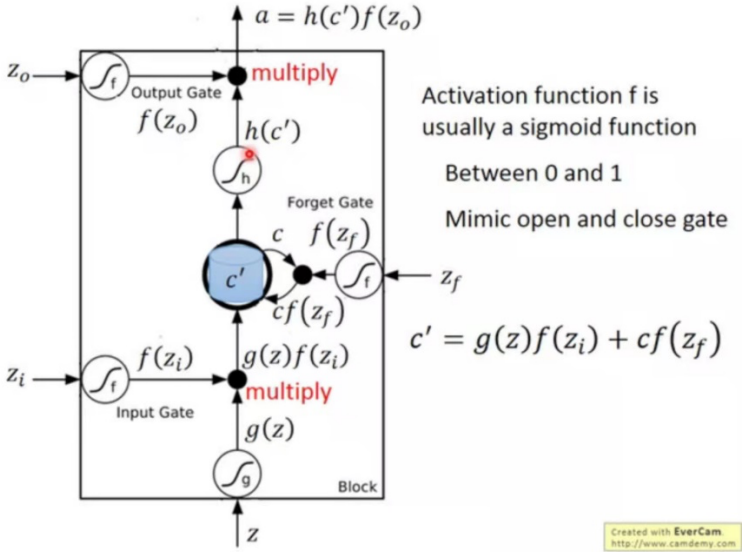
You need a model to determine the sentiment of sentences (RNN, CNN)
You need a model to do translation between two languages (RNN)

Tutorial 09 - RNN and XAI

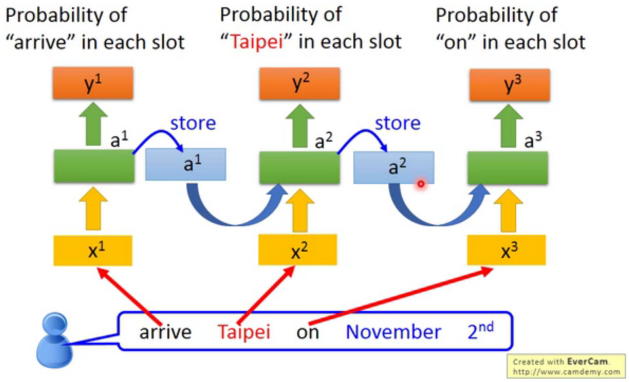


Recurrent Neural Network

$$y_t = g^{[y]}\left((W^{[hy]})^\top h_t\right)$$
$$h_t = g^{[h]}\left((W^{[xh]})^\top x_t + (W^{[hh]})^\top h_{t-1}\right)$$
$$h_t = g^{[h]}\left((W^{[xh]} \oplus W^{[hh]})^\top (x_t \oplus h_{t-1})\right)$$



RNN The same network is used again and again.



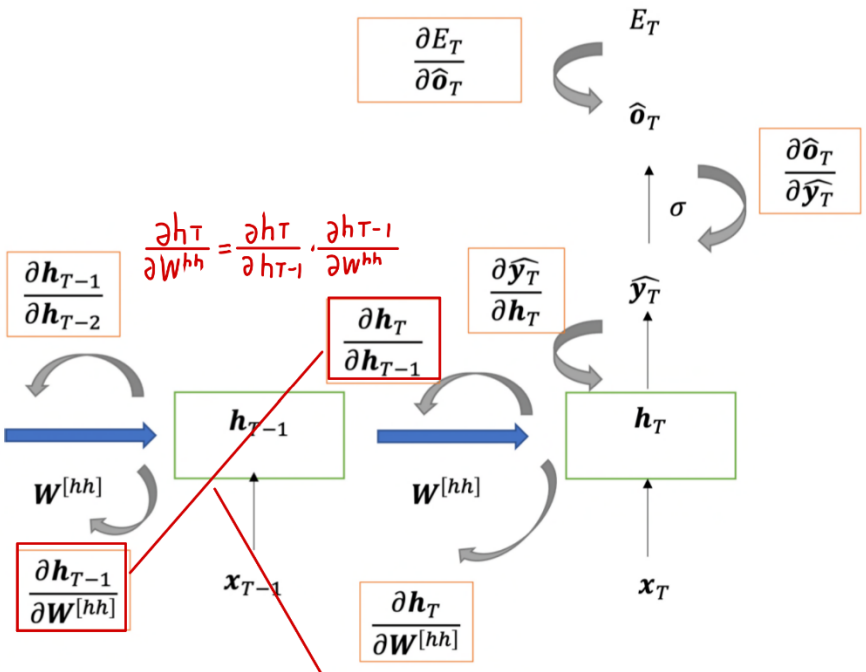


Figure 1: Gradient Flow

1. **RNN and BPTT** Here, we'll be computing gradients via *Backpropagation Through Time* (BPTT). The Forward Pass of a RNN can be characterised as follows(Here σ denotes *softmax* function):

$$\mathbf{h}_t = g^{[h]}((\mathbf{W}^{[xh]})^\top \mathbf{x}_t + (\mathbf{W}^{[hh]})^\top \mathbf{h}_{t-1}) \quad (1)$$

$$\hat{\mathbf{y}}_t = g^{[y]}((\mathbf{W}^{[hy]})^\top \mathbf{h}_t) \quad (2)$$

$$\hat{\mathbf{o}}_t = \sigma(\hat{\mathbf{y}}_t) \quad (3)$$

The loss L is the Cross Entropy Loss:

$$L = -\sum_t^T \mathbf{y}_t \cdot \log(\hat{\mathbf{o}}_t) \quad (4)$$

For simplicity, let's call the final time step loss, $E_T = -\mathbf{y}_T \log(\hat{\mathbf{o}}_T)$. The objective of BPTT is to update the parameters $\mathbf{W}^{[xh]}$, $\mathbf{W}^{[hh]}$, and $\mathbf{W}^{[hy]}$.

(a) Use Chain Rule to find an expression for $\frac{\partial E_T}{\partial \mathbf{W}^{[hh]}}$. (Note, there is no need to expand the term $\frac{\partial \mathbf{h}_{T-1}}{\partial \mathbf{W}^{[hh]}}$ further)

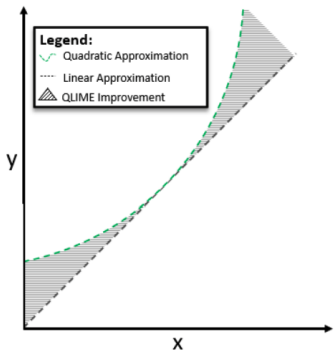
The answer is a multiplication of the terms via Chain Rule:

$$\frac{\partial E_T}{\partial \mathbf{W}^{[hh]}} = \frac{\partial E_T}{\partial \hat{\mathbf{o}}_T} \times \frac{\partial \hat{\mathbf{o}}_T}{\partial \hat{\mathbf{y}}_T} \times \frac{\partial \hat{\mathbf{y}}_T}{\partial \mathbf{h}_T} \times \frac{\partial \mathbf{h}_T}{\partial \mathbf{W}^{[hh]}} \quad (5)$$

$$\frac{\partial E_T}{\partial \mathbf{W}^{[hh]}} = \frac{\partial E_T}{\partial \hat{\mathbf{o}}_T} \times \frac{\partial \hat{\mathbf{o}}_T}{\partial \hat{\mathbf{y}}_T} \times \frac{\partial \hat{\mathbf{y}}_T}{\partial \mathbf{h}_T} \times \left(\frac{\partial g^{[h]}(\mathbf{x}_T, \mathbf{h}_{T-1})}{\partial \mathbf{W}^{[hh]}} + \frac{\partial g^{[h]}(\mathbf{x}_T, \mathbf{h}_{T-1})}{\partial \mathbf{h}_{T-1}} \times \frac{\partial \mathbf{h}_{T-1}}{\partial \mathbf{W}^{[hh]}} \right) \quad (6)$$

Figure 1 shows the gradient flow.

In LIME, we use a predictor which is a quadratic function rather than a line. How do you sample points for the decision boundary? Show how quadratic LIME can improve over a linear LIME.



What are the disadvantages of using LIME for explanation?

- Finding a proper neighborhood is difficult. For every new region, we may have to find new kernel widths. 很难找到合适的邻居。 对于每个新区域，我们可能必须找到新的内核宽度。
- Repeating the sampling process can produce different explanations. 重复抽样过程会产生不同的解释。
- Locally perturbed data may not be realistic. 局部扰动的数据可能不现实。

1. **K-means convergence:** We have learned the iterative algorithm for K-means. Let's look into the algorithm again.

Algorithm 1 K-means Clustering

1: **for** $c = 1$ to k **do**
2: $\quad \mu_c \leftarrow$ some random location
3: **while** Not Converged **do**
4: $\quad$ **for** $j = 1$ to m **do**
5: $\quad\quad y^{(j)} \leftarrow \mathrm{c} = \arg\min_c \|x^{(j)} - \mu_c\|^2$ ▷ A. assign example
6: $\quad\quad x^{(j)} \leftarrow S_c$
7: $\quad$ **for** $c = 1$ to k **do**
8: $\quad\quad \mu_c \leftarrow \frac{1}{|S_c|} \sum_{x \in S_c} x$ ▷ B. re-estimate center
9: **return** y

The algorithm leads to convergence when the clustering quality L_{clust} is minimized (*i.e.*, when cluster centers or assignments stop changing).

$$L_{clust} = \sum_{c=1}^{k} \sum_{x \in S_c} \|x - \mu_c\|^2 \tag{1}$$

(a) Show that each data assignment step (Line 5 in Algorithm 1) minimizes L_{clust}, given fixed cluster centers.

计算每个数据点到固定簇 $\mu_{assigned}$ 的距离，若最近，则 assign

$$L = \arg\min_{S} \sum_{c=1}^{k} \sum_{x \in S_c} \|x - \mu_c\|^2$$

（簇中所有点到中间点的距离和）的和最小

Image credit

$L_{clust} = \sum_{c=1}^{k} \sum_{x^{(j)} \in S} \|x^{(j)} - \mu_c\|^2$

最小化

$= \sum_{j=1}^{m} \|x^{(j)} - \mu_{assigned}\|^2$

$= \sum_{j=1}^{m} L_{x^{(j)}}$（数据点 x 到固定的簇中点的损失值）

目的是对每个数据点 minimize the error（即最小化损失）

=> 每个数据点会被分到最近的簇

(b) Show that each cluster center update step (Line 8 in Algorithm 1) minimizes L_{clust}, given fixed data assignments.

计算当前簇的新的中间点，若中间点有变化，则 assign

$L_{clust} = \sum_{c=1}^{k} \sum_{x^{(j)} \in S} \|x^{(j)} - \mu_c\|^2$

最小化

$= \sum_{c=1}^{k} L_c$（数据点到它的簇中点的损失值）

目的是对每个数据点 minimize the error（即最小化损失）

对于一个簇的损失值 L_c

$L_c = \sum_{x \in S} \|x - \mu_c\|^2$（这是矢量计算）

$= \sum_{x \in S} \left[\sum_{d} (x_d - \mu_{c,d})^2 \right]$（转化为标量计算，d 代表 x 或 μ_c 的维度）

⇓ 使 x 到簇中点的距离最小 => 求导 = 0

$\frac{\partial L_c}{\partial \mu_c} = \begin{bmatrix} \frac{\partial L_c}{\partial \mu_{c,1}} \\ \frac{\partial L_c}{\partial \mu_{c,2}} \\ \vdots \\ \frac{\partial L_c}{\partial \mu_{c,n}} \end{bmatrix}$ 不同维度的导数 => 对于其中一个维度的导数（偏导）=> $\frac{\partial L_c}{\partial \mu_{c,d}} = \sum_{x \in S} \frac{\partial}{\partial \mu_{c,d}} (x_d - \mu_{c,d})^2$

$= \sum_{x \in S} -2(x_d - \mu_{c,d})$

令 $\sum_{x \in S} -2(x_d - \mu_{c,d}) = 0$

$\left(\sum_{x \in S} x_d\right) - m\mu_{c,d} = 0$（m 代表簇中的点的数量）

$\mu_{c,d} = \frac{1}{m} \sum_{x \in S} x_d$ => 即中点