# CAL Algorithm in Mastermind

**Lanston Hau Man Chu***
Department of Electrical and Computer Engineering
University of Wisconsin-Madison
Madison, WI 53706
hchu34@wisc.edu

## Abstract

This article is a descriptive account of the setting of popular code breaking board game *Mastermind* in machine learning language. It also analyzes the performance of CAL algorithm as a strategy for the game. The results, including disagreement coefficient and total number of queries required for code breaking, will be discussed.

## 1 Introduction

*Mastermind* is a popular code-breaking board game invented by Mordecai Meirowitz in the 1970s, originating from a paper and pen game called "Bulls and Cows" in the $19^{th}$ Century.

Two players play the roles of codemaker and codebreaker respectively. The codemaker pre-selects a code and keeps it hidden to the codebreaker. The codebreaker makes guesses on the hidden code, and the codemaker offers feedback to the codebreaker on how close the guess is.

Since the codebreaker actively makes guesses and the codemaker offers feedback on the guess, the game can be categorized as a online learning problem. As the hidden code always exist, it is the case that the realizable assumption can be fulfilled.

The Cohn, Atlas, and Ladner (CAL) algorithm is a well-known algorithm for realizable online learning problems since 1994. It considers the current version space and updates it based on some certain criteria. This paper studies the CAL algorithm and discusses its performance relative to the existing best algorithms specially-designed for *Mastermind*.

### 1.1 Game Design

The basic setting is the game is codemaker (i.e. the one who decides the "code") vs. codebreaker (i.e. the one who makes "guesses" on the code).

The codemaker can pre-select a code $(\eta_1, ..., \eta_d)$ with code length $d$ among $m$ colors. The code is never revealed to the codebreaker until he guesses it correctly.

In each round, the codebreaker makes a guess $(x_1, ..., x_d)$ on the hidden code. The codemaker gives feedback to the codebreaker on how close the guess is to the hidden code by placing flags $(\texttt{Red}, \texttt{White})$ on each guess. For example, if one of the guessed element is correctly placed and is of the correct color, the codebreaker receives a red flag (i.e. $1R$); if two elements are correctly placed and are of the correct color, the codebreaker receives two red flags (i.e. $2R$). If an element is of the correct color but is placed in an incorrect position, the codebreaker receives a white flag (i.e. $1W$).

For example, given the hidden code $\eta = (1, 2, 3, 4)$, a guess $(1, 4, 5, 3)$ would give $flag_{(1,2,3,4)}((1, 4, 5, 3)) = (1R, 2W)$.

*Source Code: https://github.com/lanstonchu/mastermind

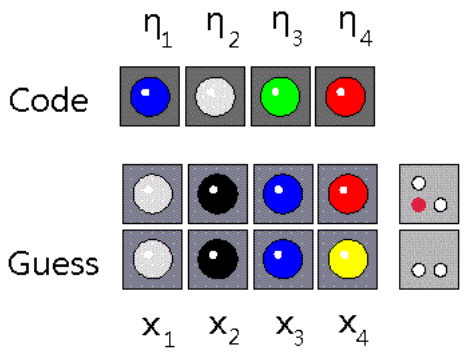


Figure 1: *Mastermind*. $(\eta_1, \eta_2, \eta_3, \eta_4)$ is the code and $(x_1, x_2, x_3, x_4)$ refers to the guess. After each guess, the codemaker places red and white flag(s) (or no flag) for the codebreaker.

In the original release of *Mastermind* in 1972, the code length is 4 and there are 6 colors to choose from. Many previous Mathematical analysis are also based on this setting, e.g. Kunth (1977) and Koyama and Lai (1993). For the purposes of analysis, this paper would also use $g = 4, m = 6$. Results under other settings will be discussed in the conclusion section. Therefore the code and the guess are $(\eta_1, \eta_2, \eta_3, \eta_4)$ and $(x_1, x_2, x_3, x_4)$ respectively.

### 1.2 CAL Algorithm

CAL makes a good guess of $h$ within the hypothesis class $\mathcal{H}$ under the realizable assumption, i.e. $\exists h^* \in \mathcal{H}, y_t = h^*(x_t), \ \forall t$. Since it is realizable, we would keep updating the version space $V_i$ by trimming it. The procedures for CAL would be as below:-

1. Initialize $V_1 = \mathcal{H}$
2. for epochs $i = 1, ..., n$
3. Collect $k$ items $x_1, ..., x_k \overset{i.i.d.}{\sim} P_{x|V_i}$ s.t. they belongs to $DIS(V_i) := \{x \in X : \exists h, h' \in V_i, h(x) \neq h'(x)\}$
4. Query $\{x_i\}_{i=1}^k$. Oracles gives their labels $\{y_i\}_{i=1}^k$
5. Version space trimming: $V_{i+1} = \{h \in V_i : h(x_i) = y_i, \forall i \in [\,k]\,\}$
6. After n epochs, return any $h \in V_{n+1}$

Under the CAL framework, the disagreement coefficient $\theta$ reveals the "topology" of the tasks. The disagreement coefficient plays an important role in our analysis:-

$$\theta = \sup_{\tau \in (0,1)} \frac{P_x(DIS(Ball(h^*, \tau)))}{\tau}$$

As discussed in the following sections, $\theta$ will determine the total number of queries under CAL in the following manner:-

$$\texttt{Total number of queries by CAL} = 2\theta\Big[log(log_2\frac{1}{\epsilon}) + log\frac{|\mathcal{H}|}{\delta}\Big]log_2\frac{1}{\epsilon}$$

## 2 Problem Set Up

### 2.1 The Feedback Flags $R_x$ and $W_x$ [2]

In machine learning framework, we have hypothesis $h \in \mathcal{H}$, where $\mathcal{H}$ is the hypothesis class. We want to understand what $h$ and $\mathcal{H}$ should look like in the *Mastermind* problem. To answer this, we

[2]Program: give_flag()

first formulate the red and white flags. For each guess $x$ from the codebreaker, $x = (x_1, x_2, x_3, x_4)$ and thus $X = \{(x_1, x_2, x_3, x_4)\}$ would be the set of codes combinations. The flags would be written as $R_x$ and $W_x$ for a specific $x$, and we have:-

$$R_x = \#(\texttt{red flag(s) for } x) = |\{(\eta_i, x_i) : \eta_i = x_i\}|$$

Thus $R_x$ reflects the number of correct color and correct position. To formulate $W_x$, we first define the color histogram $C_x$ and $G_x$ for the code and guess respectively. For color $j$,

$$\begin{cases} C_{x,j} = |\{\eta_i : \eta_i = j, \texttt{ for } i \in [\,g\,]\}| \texttt{ ( i.e.Color Histogram for Code)} \\ G_{x,j} = |\{x_i : x_i = j, \texttt{ for } i \in [\,g\,]\}| \texttt{ (i.e.Color Histogram for Guess)} \end{cases}$$

$$W_x = \#(\texttt{white flag(s) for } x) = \sum_{j=1}^{m} min(C_{x,j}, G_{x,j}) - R_x$$

For example, given the code $\eta = (1, 2, 3, 4)$, a guess $x = (3, 2, 5, 3)$ would have $R_x = |\{(2, 2)\}| = 1$. $C_x = (1, 1, 1, 1, 0, 0)$, $G_x = (0, 1, 2, 0, 1, 0)$, and $W_x = (1 + 1) - R_x = 2 - 1 = 1$. Therefore

$$flag^*((3, 2, 5, 3)) = flag_{(1,2,3,4)}((3, 2, 5, 3)) = (1R, 1W)$$

### 2.2 Hypothesis class $\mathcal{H}$

Since the codebreaker needs to find out the code $(\eta_1, \eta_2, \eta_3, \eta_4)$ (i.e. the case of $(g = 4)$), such code $\eta$ would uniquely determine what flag $(r_x, w_x)$ should be given whenever a $x$ is queried. Thus each possible $\eta = \{\eta_i\}_{i=1}^{g}$ refers to one and only one $h \in \mathcal{H}$. We call the hidden code $h^*$, which is the ground truth code $(\eta_1, \eta_2, \eta_3, \eta_4)$ selected by the codemaker.

As the codemaker knows $h^*$, he will be able to tell the actual $(r_x, w_x)$ flags for whatever $x$ that the codebreaker is guessing. Thus the label $y$ is equivalent to all of the followings:-

$$y = (r_x, w_x) = flag_\eta(x) = h_\eta(x) = h_{\{\eta_i\}_{i=1}^{g}}(x) = h^*(x)$$

Therefore the paradigm of *Mastermind* is equivalent to finding $h^*$ in a finite number of queries within $\mathcal{H}$, with each query $x$ referring to each guess in the game.

Note that since each $h$ refers to one possible code, $|\mathcal{H}| = |X|$ and there is a one-one correspondence between each $h$ and $x$. Thus for $g = 4, m = 6$, we have $|H| = m^g = 6^4 = 1296$.

We can also write $h^* = (\eta_1, \eta_2, \eta_3, \eta_4)$, which is an abuse of notation, but should not cause confusion since the there is a one-one correspondence between $X$ and $\mathcal{H}$.

## 3 CAL Algorithm

### 3.1 The distance $d$ [3]

The distance between $h_1$ and $h_2$ would be defined by the probability of assigning different labels: $d(h_1, h_2) = \mathbb{E}_{x \sim P_x} \mathbb{1}(h_1(x) \neq h_2(x)) = \mathbb{P}_{x \sim P_x}(h_1(x) \neq h_2(x))$. Note that $d$ is symmetric, i.e. $d(h_1, h_2) = d(h_2, h_1)$.

Therefore $d(h, h^*) = R(h)$, which is the risk of $h$.

Since $h$ and $x$ have one-one correspondence, we can also write $d(h_1, h_2) = d(x_1, x_2) = d(x_1, h_2)$, i.e. $d(h, h^*) = d((x_1, x_2, x_3, x_4), (\eta_1, \eta_2, \eta_3, \eta_4))$. Again, it is an abuse of notation but should not cause confusion.

For example, if $h^* = (1, 2, 3, 4)$ and $h = (1, 2, 3, 5)$, then

[3] Program: distance()

Table 1: Relationship and statistics between flags and the distance $d$ under $Strategy_{diff}$

| $(R_x,$ | $W_x)$ | $\#(cases)$ | $mean(d)$ | $std(d)$ | $max(d)$ | $min(d)$ |
|---|---|---|---|---|---|---|
| (4, | 0) | 1 | 0 | 0 | 0 | 0 |
| (2, | 2) | 6 | 0.5 | 0 | 0.5 | 0.5 |
| (1, | 3) | 8 | 0.59 | 0 | 0.59 | 0.59 |
| (0, | 4) | 9 | 0.65 | 0 | 0.65 | 0.65 |
| (3, | 0) | 20 | 0.67 | 0.03 | 0.71 | 0.65 |
| (2, | 1) | 48 | 0.75 | 0 | 0.75 | 0.75 |
| (1, | 2) | 132 | 0.82 | 0.01 | 0.83 | 0.82 |
| (2, | 0) | 96 | 0.84 | 0.03 | 0.88 | 0.82 |
| (0, | 3) | 136 | 0.85 | 0 | 0.85 | 0.84 |
| (1, | 1) | 252 | 0.85 | 0.03 | 0.92 | 0.84 |
| (0, | 2) | 312 | 0.88 | 0.02 | 0.94 | 0.85 |
| (1, | 0) | 108 | 0.9 | 0.02 | 0.97 | 0.88 |
| (0, | 1) | 152 | 0.91 | 0.01 | 0.92 | 0.9 |
| (0, | 0) | 16 | 0.92 | 0.01 | 0.96 | 0.91 |
| (3, | 1) | 0 | NaN | NaN | NaN | NaN |

$$d(h, h^*) = d((1,2,3,5),(1,2,3,4)) = \mathbb{P}_x(h(x) \neq h^*(x)) = \frac{\sum_x \mathbb{1}(flag(x) \neq flag^*(x))}{|X|} = 0.7099$$

While $flag^*(h) = flag_{(1,2,3,4)}((1,2,3,5)) = (3R, 0W)$. Also note that $0.7099 = \mathbb{P}_x(h(x) \neq h^*(x)) \leq \mathbb{P}_x(\texttt{x contains either 4 or 5 in any position}) = 1 - (\frac{6-2}{6})^4 = 0.8025$, while the upper bound is a good approximation.

Same flags do not guarantee same distance. For example, consider another $h' = (1,2,3,3)$, while $h^*$ is still $(1,2,3,4)$, we have $\mathbb{P}_x(h'(x) \neq h^*(x)) = 0.6451 \neq 0.7099$, but both $h$ and $h'$ have $(3R, 0W)$. (Also worth to note that $0.6451 \leq P_x(\text{x contains either two 3 or one 4}) = 1 - \frac{3^5 1}{6^4} = 0.7292$.)

One of the most common strategies for the codemaker is that $\eta_1, \eta_2, \eta_3, \eta_4$ are all set in different colors. We called this strategy $Strategy_{diff}$. Under $Strategy_{diff}$, without loss of generality we can simply use $h^* = (1,2,3,4)$. Without further specification, we would use $h^* = (1,2,3,4)$ henceforth. The relationship between the flags and the distance $d$ is shown in Table 1[4].

It is counter-intuitive that better flags do not guarantee smaller distance $d$. For example, $flag^*((2,5,5,5)) = (0R, 1W)$ and $flag^*((6,6,5,5)) = (0R, 0W)$, but $d((2,5,5,5), h^*) = 0.9205$ while $d((2,5,5,5), h^*) = 0.9136 < 0.9205$. In this case, $h$ with better flag has larger distance with $h^*$.

### 3.2 $\tau$-Ball

For a specific $h^*$, after getting the distance $d(h, h^*)$ for all $h \in \mathcal{H}$, we can easily determine whether a specific $h$ is inside the $\tau$-Ball $B(h^*, \tau) := \{h \in \mathcal{H} : d(h, h^*) \leq r\}$. For example, as $d((1,2,3,5), h^*) = 0.7099$, we know that $(1,2,3,5) \in B(h^*, 0.8)$.

Since $|\mathcal{H}|$ is finite, we can also see that the possible number of effective radius of balls is also finite. In our setting $g = 4, m = 6$, after removing the symmetric/duplicated cases, we have 49 different unique values of effective $\tau$.

### 3.3 Version space $V_i$

The CAL algorithm requires realizable assumption, i.e. $\exists h^* \in \mathcal{H}, y_t = h^*(x_t)\,,\ \forall t$. Since $h^*$ refers to the hidden code pre-selected by the codemaker, such $h^*$ exists and thus the realizable assumption

[4]Program: relationship_flag_vs_dist()

Table 2: Boundary values of $P_x(DIS(Ball(h^*, \tau))$. Note that for $\tau \in (\tau_{bdy1}, \tau_{bdy2})$, we have $P_x(DIS(Ball(h^*, \tau)) = P_x(DIS(Ball(h^*, \tau_{bdy1}))$

| $\tau$ | 0 | 0.5 | 0.593 | 0.645 | 0.65 | 0.71+ |
|---|---|---|---|---|---|---|
| $P_x(DIS(Ball(h^*, \tau))$ | 0 | 0.985 | 0.985 | 0.988 | 0.988 | 1 |
| $\lvert DIS(Ball(h^*, \tau)\rvert$ | 0 | 1276 | 1276 | 1280 | 1280 | 1296 |

is fulfilled. For the version space update $V_{i+1} = \{h \in V_i, h(x_i) = y_i\,,\ \forall i \in [\,k]\,\}$, we would then trim the version space $V_i$ in each round of guess, i.e. to remove those $x$ that are inconsistent to the received flags. Here, note that $k = 1$, since in *Mastermind* we can only query one $x$ in each guess.

For example, if the first guess is $(5, 5, 6, 6)$ and the flags that the codebreaker receives is $(0R, 0W)$, then it is obvious that 5 and 6 are not contained by $h^*$, and therefore all $h$ containing either 5 or 6 in any position can be removed from the version space.

### 3.4 Disagreement region $DIS(\cdot)$ [5]

For disagreement region $DIS(V_i)$, that is defined as $DIS(V_i) := \{x \in X : \exists h, h' \in V_i, h(x) \neq h'(x)\}$. That is, if $(1, 2, 3, 4)$ and $(1, 2, 3, 6)$ are inside $V_i$, then $(6, 6, 6, 6) \in DIS(V_i)$, since $flag_{(1,2,3,4)}(6, 6, 6, 6) = (0R, 0W) \neq (1R, 0W) = flag_{(1,2,3,6)}(6, 6, 6, 6)$.

Here, $P_x(DIS(V_i)) = \frac{|DIS(V_i)|}{m^g} = \frac{|DIS(V_i)|}{6^4}$.

Besides, we define the probability measure within the disagreement region as $Q_{XY} = P_{X|V_i} \cdot P_{Y|X}$, and therefore $R_Q(h) = \frac{R_p(h)}{P_X(DIS(V_i))}$.

### 3.5 Disagreement coefficient $\theta$ [6]

With the 49 unique values of effective $\tau$ as mentioned above, we have the values of $P_x(DIS(Ball(h^*, \tau))$ as per Table 2.

For the disagreement coefficient

$$\theta = \sup_{\tau \in (0,1)} \frac{P_x(DIS(Ball(h^*, \tau)))}{\tau}$$

we have $\theta = \left.\frac{P_x(DIS(Ball(h^*,\tau)))}{\tau}\right|_{\tau=0.5} = \frac{0.985}{0.5} = 1.9691.$

### 3.6 Total number of queries by CAL [7]

In *Mastermind*, we can only query one possible code in each guess, and thus $k = 1$. But let us assume that we can query $k$ items in each guess and we would like to know what would be the total number of queries $kn$, where $n$ is the number of guess.

Now, if we write $r := R_Q(h)$, for $h \in V_i$, $\mathbb{P}(h$ `doesn't survive in` $V_{i+1}) = 1 - (1 - r)^k$. Therefore

$$\mathbb{P}_{h \in V_i}(h \texttt{ survived}) \leq |V_i|(1 - r)^k \leq |V_i|e^{-rk} := \frac{\delta}{n}$$

so that when we do the union over all $n$ epochs, we can get $\delta$.

So when $k \geq \frac{log\frac{n|V_i|}{\delta}}{r}$, we have $d(h, h^*) = R_p(h) \leq r \cdot P_x(DIS(V_i))$ and therefore $V_{i+1} \subset Ball(h^*, r \cdot P_x(DIS(V_i)))$.

We also want $P_x(DIS(V_{i+1})) \leq \frac{1}{2}P_x(DIS(V_i))$, which, after iterations, will give us:-

[5] Program: DIS()

[6] Program: get_theta()

[7] Program: total_CAL_queries()

Table 3: $k$ and $n$ for $\epsilon = d_{\texttt{2nd best}}$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $g$ | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| $m$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| $d_{\texttt{2nd best}}$ | 0.91 | 0.67 | 0.63 | 0.56 | 0.5 | 0.45 | 0.41 | 0.37 | 0.34 |
| $\epsilon$ | 0.91 | 0.67 | 0.63 | 0.56 | 0.5 | 0.45 | 0.41 | 0.37 | 0.34 |
| $kn$ | 1.68 | 12 | 17.77 | 27.62 | 40.02 | 54.04 | 69.03 | 84.55 | 100.29 |
| $k$ | 11.83 | 20.52 | 26.2 | 33.02 | 40.02 | 46.78 | 53.12 | 59 | 64.44 |
| $n$ | 0.14 | 0.59 | 0.68 | 0.84 | 1 | 1.16 | 1.3 | 1.43 | 1.56 |

Table 4: $k$ and $n$ for $\epsilon = 0.5$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $g$ | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| $m$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| $d_{\texttt{2nd best}}$ | 0.91 | 0.67 | 0.63 | 0.56 | 0.5 | 0.45 | 0.41 | 0.37 | 0.34 |
| $\epsilon$ | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| $kn$ | 13.18 | 21.35 | 26.9 | 33.42 | 40.02 | 46.32 | 52.16 | 57.54 | 62.47 |
| $k$ | 13.18 | 21.35 | 26.9 | 33.42 | 40.02 | 46.32 | 52.16 | 57.54 | 62.47 |
| $kn$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

$$d(h, h^*) \leq \frac{1}{2^n} := \epsilon$$

We thus get $n = log_2 \frac{1}{\epsilon}$.

From the definition of $\theta$, we would have $\forall \tau \in (0,1),\ P_x(DIS(Ball(h^*, \tau))) \leq \theta\tau$ Then we have:-

$$P_x(DIS(V_{i+1})) \leq P_x(DIS(Ball(h^*, r \cdot P_x(DIS(V_i))))) \leq \theta \cdot r \cdot P_x(DIS(V_i))$$

To have $P_x(DIS(V_{i+1})) \leq \frac{1}{2} P_x(DIS(V_i))$, we choose $r = \frac{1}{2\theta}$. Therefore

$$k = 2\theta \Big[ loglog_2 \frac{1}{\epsilon} + log \frac{|\mathcal{H}|}{\delta} \Big] = 2\theta \Big[ log(log_2 \frac{1}{\epsilon}) + log \frac{m^g}{\delta} \Big]$$

$$\texttt{Total number of queries by CAL} = kn = 2\theta \Big[ log(log_2 \frac{1}{\epsilon}) + log \frac{m^g}{\delta} \Big] log_2 \frac{1}{\epsilon}$$

From Table 1, we see that for the flag $(2R, 2W)$, $d = 0.5$, so we set $\epsilon = 0.5$. By using $\delta = 0.1$ and under our setting $g = 4, m = 6$, we have $k = 40.02, n = 1$ and $\texttt{Total queries} = kn = 40.02$.

## 4 Conclusion

Since $(k = 41, n = 1)$ is in fact doing all $41$ guesses in one go, the above result tells us that we can nearly break the hidden code with probability $0.1$ by making 41 initial guess in one go without receiving any previous flags. If we go back to the original setting of *Mastermind* (i.e. $k = 1$), we would expect that $(k = 1, n = 41)$ can only do better than $(k = 41, n = 1)$ as we would receive flags from the codemaker for the previous guess, and therefore our new guesses can be based on the received flags.

As a comparison, the number of $41$ guesses is much larger than the average number of guesses of $4.478$ (Kunth 1977) and $4.341$ (Koyama and Lai 1993) under their algorithms specially designed for *Mastermind*.

Since the 1990s, there is another popular *Mastermind* variation with $m = 8$. If we do the same analysis with $\epsilon = 0.5$, we would have $kn = 52.16$. If we set $\epsilon = 0.41$, which is same as the second best distance $d$ (except $h^*$ whose $d$ is always the best, i.e. 0), we would have $kn = 69.03$. For other $kn$ values under different $m$, please refer to Table 3 and 4 [8].

[8] Variable: setting_infos

## 5 Future Work

As a generic algorithm, CAL algorithm performs worse than the existing algorithms specially designed for *Mastermind*. However, further exploration in other aspects may prove worthwhile.

First, *Mastermind* is a game based on code-breaking. Other variations of code-breaking problem, for example, different types of feedback (i.e. different flags formulae) and different paradigms between the codemaker and the codebreaker, can also be analyzed under Machine Learning framework, under the situation that the hidden code $h^*$ is pre-selected and exists.

Second, the CAL analysis can shed light on the Littlestone dimension of *Mastermind* and other code-breaking scenarios. Under the online learning framework, the codebreaker would randomly pick a guess $x$ from $DIS(V_i)$ to query, while under the Littlestone dimension adversarial framework, the codemaker (i.e.the "world") would decide the next guess $x$ for the codebreaker. The anlaysis on CAL offers important insights on the lower bound of the Littlestone dimension.

## References


[1] COHN D. & ATLAS, L. & LADNER, R. (1994) Improving generalization with active learning. Machine Learning, 15(2):201–221, 1994.

[2] KNUTH, D.E. (1976-77) "The Computer as a Master Mind." J. Recr. Math. 9, 1-6, 1976-77.

[3] KOYAMA, K. & LAI, T.W. (1993) "An Optimal Mastermind Strategy." J. Recr. Math. 25, 251-256, 1993.

[4] HANNEKE, S. (2014) Theory of Active Learning

[5] SHWARTZ, S.S. & DAVID, S.B. (2014) Understanding machine learning: From theory to algorithms, Cambridge university press