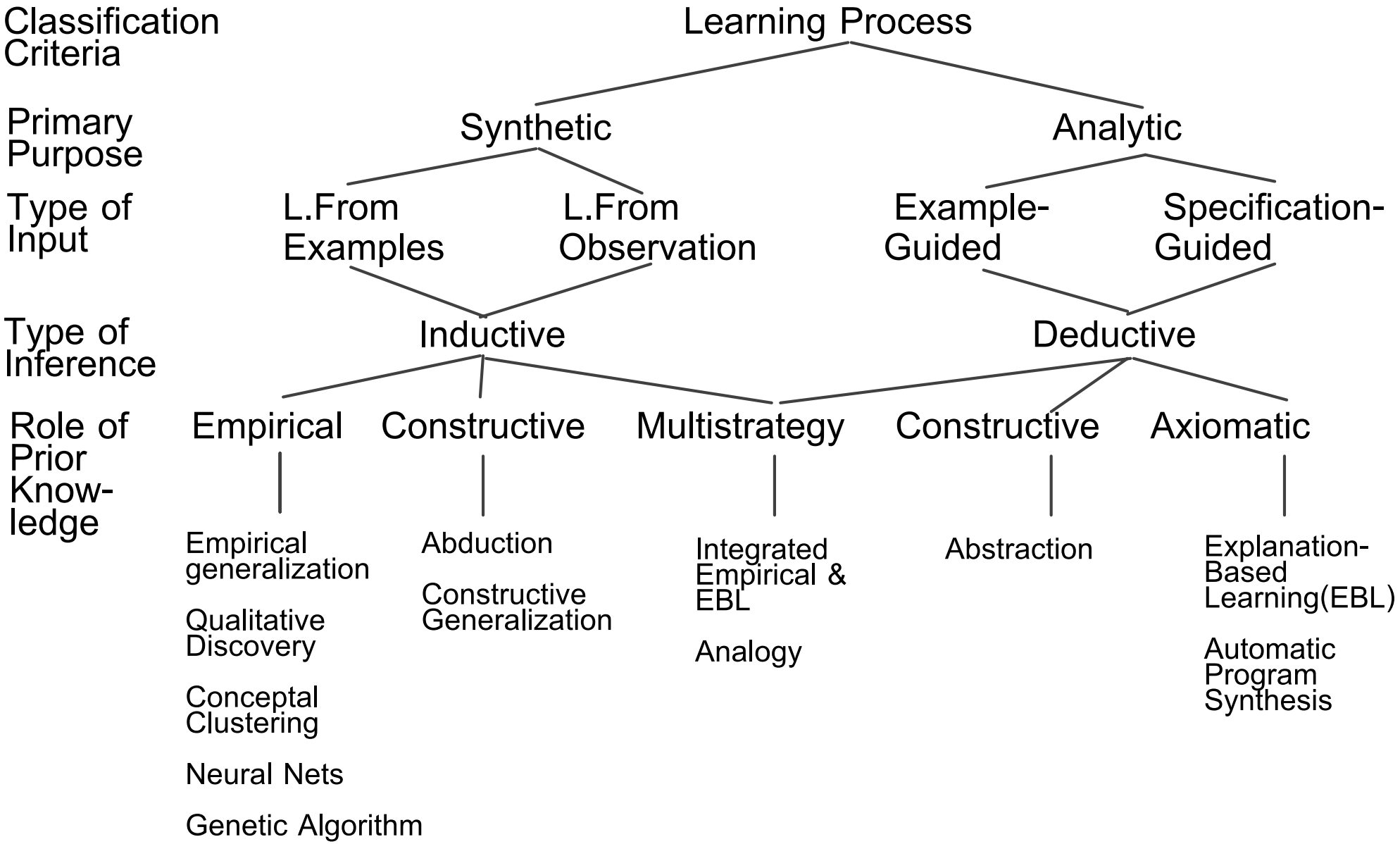
5. Machine Learning

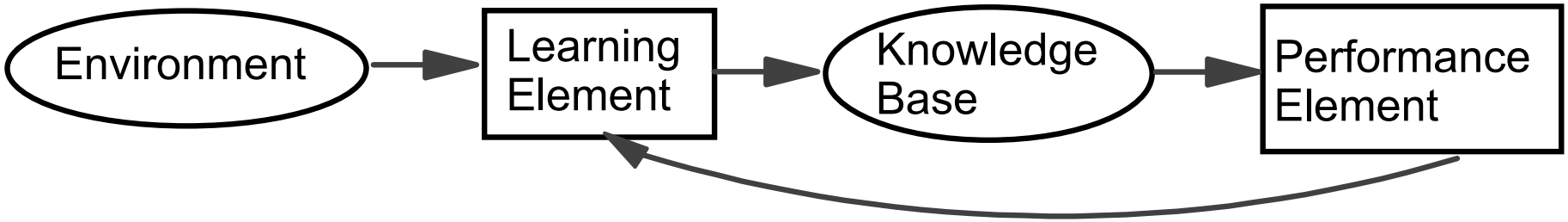


นิยามของ learning

- any process by which a system improves its performance
- the acquisition of explicit knowledge
- skill acquisition
- theory formation, hypothesis formation, inductive inference

Model ของ Learning

- จากนิยามแรก เราแสดง model ของระบบเรียนรู้ดังรูป



  - evironment ให้ information ต่าง ๆกับ learning element
  - learning element ใช้ information ที่ได้สร้าง knowledge base
  - performance element ใช้ knowledge base ทำงาน
  - ผลของงานที่ทำโดย performance element อาจ feedback กลับไปให้ learning element อีกเพื่อปรับปรุง knowledge base

- Environment -- information ต่างๆที่สามารถให้กับระบบเรียนรู้เพื่อนำไปใช้ในการทำงาน
  - information ที่ระบบรับเข้ามาสามารถนำไปให้ performance element ใช้ได้เลย (rote learning: การเรียนรู้โดยการจำ)
  - information ที่ระบบรับเข้ามา specific เกินไป ระบบเรียนรู้ต้องสร้าง general rules (learning from examples)
  - information ที่เกี่ยวข้องเชิง analogy กับงานที่ performance element ต้องการทำ ระบบต้องสร้าง analogy rules (learning ay analogy)
- Knowledge Base -- representation ของ knowledge base
  - predicate calculus
  - frame
  - decision tree

5.1 Learning by Simulating Evolution : Genetic Algorithm

- Genetic algorithmได้แนวคิดจาก *individuals, mating, chromosome crossover, gene mutation, fitness, natural selection*
- เป็นการเรียนรู้แบบหนึ่ง ซึ่งใช้จัดการกับปัจหาของ local maximaในการsearch
- วิธีsearchทั่วไปจะมองว่าlocal maximaเป็นกับดัก และจะหลีกเลี่ยงกับดักโดยใช้วิธีต่าง ๆเช่น backtrackingหรือparallel search ด้วยinitial stateที่ต่าง ๆกัน เป็นต้น แต่genetic algorithmไม่ใช่

Chromosomes Determine Hereditary Traits

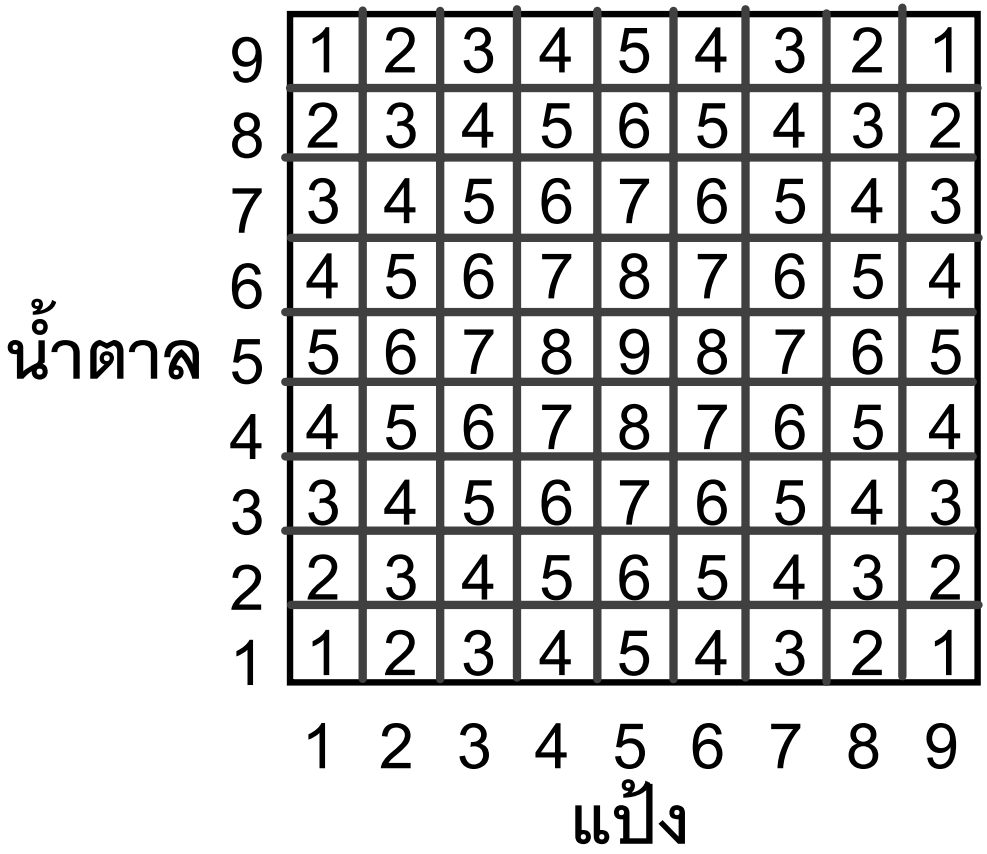
- แต่ละcellในพืชชั้นสูงและสัตว์ประกอบด้วย 1 nucleus และnucleusหนึ่ง ๆประกอบด้วยหลาย chromosome
- Chromosomeจะมีgeneซึ่งเป็นตัวกำหนดลักษณะพิเศษของสิ่งมีชีวิต
- Chromosomesจะอยู่เป็นคู่ โดยที่จะได้รับมาจากพ่อและแม่อย่างละ 1
- ในขณะที่chromosomes 2คู่จากพ่อแม่จะรวมกันเพื่อผลิตchromo-somesลูกนั้น บางครั้งจะเกิดcrossover ซึ่งเป็นการที่genesจาก chromosomesพ่อแม่ปะปนกัน เกิดchromosomesใหม่ขึ้น2คู่
- ขณะที่cellแบ่งตัวจะเกิดกระบวนการchromosome-copying ซึ่งบาง ครั้ง จะมีการเปลี่ยนแปลงของgenesซึ่งต่างจากgenesพ่อและแม่
- การที่เกิดgeneที่ไม่เคยมีมาก่อนนี้เรียกว่า mutation

The Fittest Survive

- กฎ " evolution through natural selection" ของ Charles Darwin
  - สิ่งมีชีวิตมีแนวโน้มที่จะสืบทอดลักษณะพิเศษให้ลูกหลาน
  - ธรรมชาติจะผลิตสิ่งมีชีวิตที่มีลักษณะพิเศษแตกต่างไปจากเดิม
  - สิ่งมีชีวิตที่เหมาะสมที่สุด (fittest) (สิ่งมีชีวีตที่มีลักษณะพิเศษที่ธรรมชาติพอใจมากที่สุด) มีแนวโน้มที่จะมีลูกหลานมากกว่าตัวที่ไม่เหมาะสม ดังนั้นประชากรจะโน้มเอียงไปทางตัวที่เหมาะสม
  - เมื่อช่วงเวลาผ่านไปนานๆ การเปลี่ยนแปลงจะสะสมไปเรื่อยๆ และเกิด speciesใหม่ที่เหมาะกับสภาพแวดล้อม
- Natural selection เกิดจากการเปลี่ยนแปลงที่เป็นผลของcrossover และ mutation

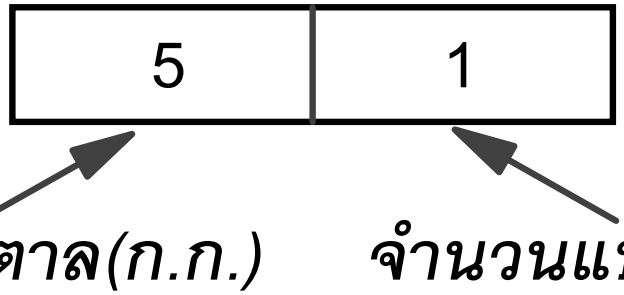
Genetic Algorithms Involve Myriad Analogs

ปัญหา : Mr. Kookie เป็นคนทำคุ๊กกี้ต้องการoptimizeส่วนผสมของจำนวนของน้ำตาลและแป้ง สมมุติว่าคุณภาพของคุ๊กกี้ที่ได้เป็นฟังก์ชั่นตามรูป



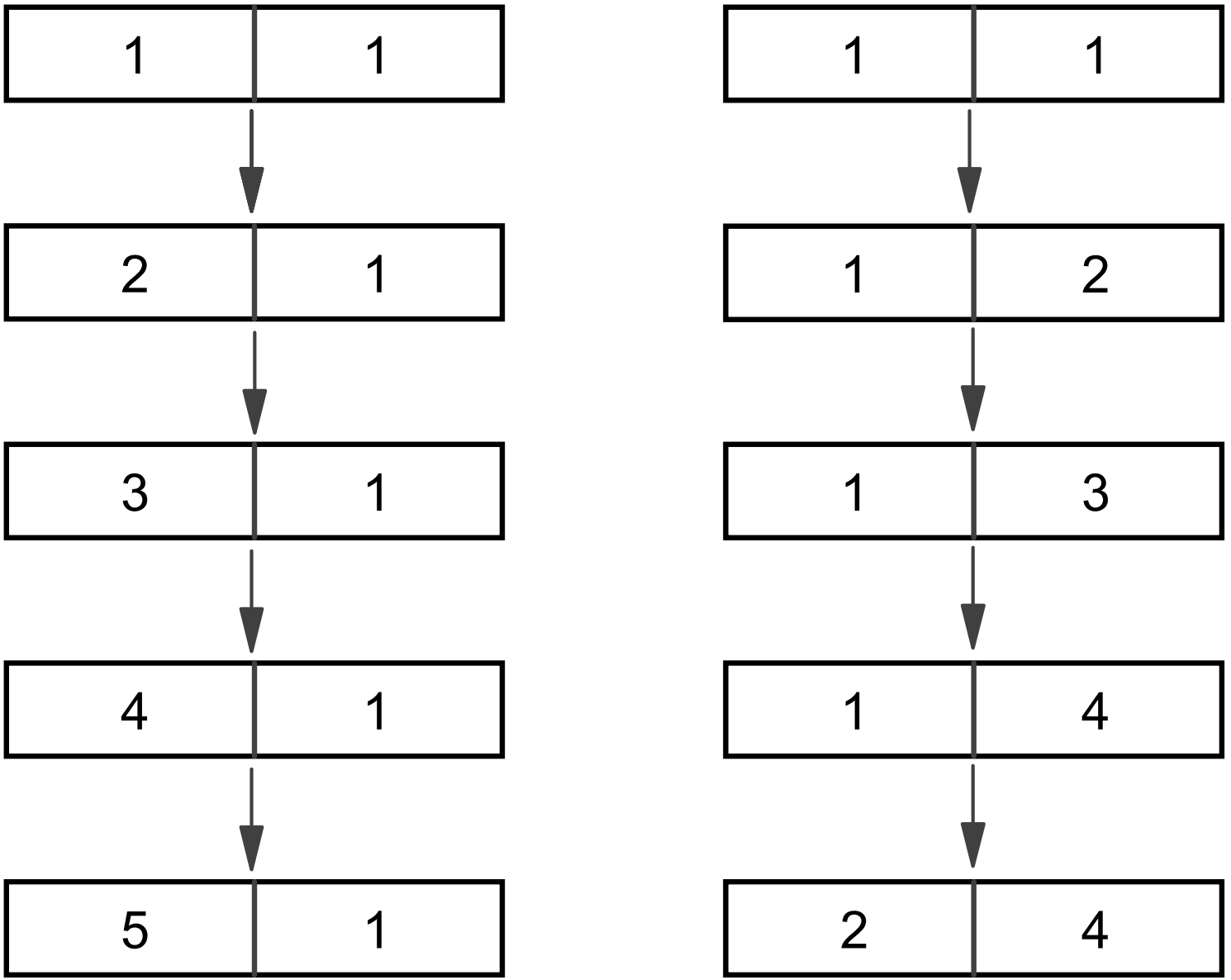
Bump Mountain

- Kookieให้แต่ละถาดของคุ๊กกี้เป็น"individual" ดังรูป

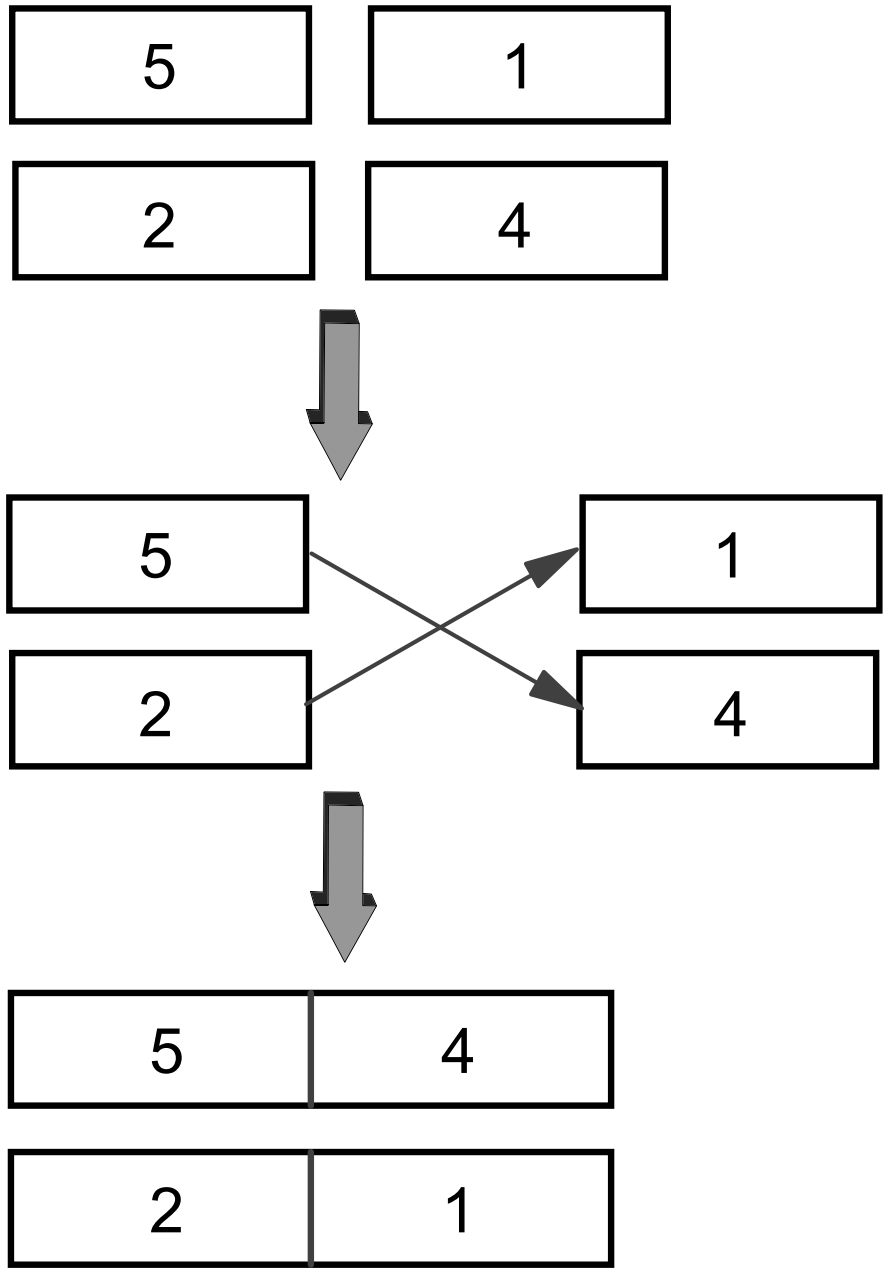


- แต่ละ"chromosome"ประกอบด้วย 2 "genes" และแต่ละ"genes"เป็นตัวเลขตั้งแต่ 1 ถึง 9
- Geneแรกบอกจำนวนน้ำตาล และ geneที่สองบอกจำนวนแป้ง
- Chromosome
  - มีลิสต์ของสมาชิกที่เรียกว่า genes
  - กำหนด fitness
  - สามารถถูกสร้างขึ้นจากลิสต์ของgenes หรือ crossingระหว่าง chromosomeคู่หนึ่งที่มีอยู่
  - สามารถถูกmutatedจากchromosomeหนึ่งที่มีอยู่โดยการเปลี่ยน geneตัวหนึ่ง
  - สามารถผลิตgeneได้

- Kookieกำหนดให้แต่ละindividualมีcopyของchromosomeได้เพียงตัวเดียว
- Kookieจำลองแบบmutationของchromosome โดยเลือกgeneหนึ่งในchromosomeนั้นแบบสุ่ม แล้วเปลี่ยนค่าของgeneด้วยการบวกหรือลบ 1แบบสุ่ม โดยที่ค่าที่ได้ต้องอยู่ระหว่าง 1 ถึง 9
- รูปที่ 5.1.1 แสดงวิวัฒนาการของchromosome
- Kookieจำลองแบบcrossoverของchromosomes โดยตัดที่กึ่งกลางของ 2 chromosomes แล้วนำแต่ละส่วนมาต่อกัน ดังรูปที่ 5.1.2 (ในกรณีทั่วไปที่ chromosomeประกอบด้วยgenesมากกว่า2ตัว การตัดและการต่อจะซับซ้อนยิ่งขึ้น)



รูปที่ 5.1.1 การวิวัฒนาการของchromosome



รูปที่ 5.1.1 Crossover**ของ**chromosome 5-1 **กับ** chromosome 2-4

The Standard Method Equates Fitness

- fitnessของchromosomeคือความน่าจะเป็นที่chromosomeจะอยู่รอดในgenerationต่อไป
- Standard methodสำหรับคำนวณfitness

$$f_i = \frac{q_i}{\sum_j q_j}$$

โดยที่ f_i คือ fitnessของchromosomeตัวที่ i ซึ่งมีค่าอยู่ตั้งแต่ 0 ถึง 1

q_i คือ qualityของคุ๊กกี้ที่ถูกกำหนดโดยchromosomeตัวที่ i

- สมมุติว่าประชากรประกอบด้วย4 chromosomesคือ 1-4,3-1,1-2,1-1 fitnessของแต่ละchromosomeแสดงได้ในตารางหน้าต่อไป

Chromosomes	Quality	Standard fitness
1 4	4	0.40
3 1	3	0.30
1 2	2	0.20
1 1	1	0.10

- ในการที่จะจำลองnatural selection
  - สร้าง "population(ประชากร)" เริ่มต้นจาก 1 chromosome
  - Mutate gene(s)ในchromosome(s)ปัจจุบัน และผลิตลูกที่ได้จากmutation
  - ผสมพันธุ์chromosomes และผลิตลูก
  - เพิ่มลูกที่เกิดใหม่ในประชากร
  - สร้างgenerationใหม่โดยเลือกchromosomesที่มีค่าfitnessesสูง

Genetic Algorithm Generally Involve Many Choices

- ในประชากรหนึ่งๆควรจะมีchromosomesจำนวนเท่าไร?
  ถ้าน้อยไป มีแนวโน้มที่ประชากรใหม่จะมีลักษณะเหมือนกันทั้งหมด และcrossoverก็จะไม่มีผล
  ถ้ามากไป จะเสียเวลาคำนวณมาก
- Mutation rateควรจะเป็นเท่าไร? ถ้าต่ำไป ลักษณะใหม่จะเกิดช้า
  ถ้าสูงไป generationใหม่จะไม่เกี่ยวเนื่องกับgenerationเดิม
- ยอมให้มีการผสมพันธุ์หรือไม่? ถ้ายอม เลือกคู่ผสมอย่างไร และ crossoverกำหนดอย่างไร
- Chromosomeเดียวกันในประชากรหนึ่งๆมีหลายตัวได้หรือไม่

It Is Easy to Climb Bump Mountain Without Crossover

- สมมุติว่าKookieกำหนดการจำลองแบบของnatural selectionตามนี้
  - เริ่มจากchromosome 1-1 เพียงตัวเดียว
  - ไม่มีchromosomeใดที่มีหลายตัวในประชากรหนึ่ง ๆ
  - Chromosomes 4 ตัวหรือน้อยกว่าจะอยู่รอดไปถึงgenerationใหม่
  - ตัวที่อยู่รอดจะแข่งขันกับchromosomesใหม่ เพื่อกำหนด chromosomesที่จะอยู่ในgenerationต่อไป
  - ในแต่ละตัวที่อยู่รอด เลือกgeneตัวหนึ่งแบบสุ่มเพื่อทำmutation ถ้าตัวที่ได้จากmutationยังไม่เคยมีมาเลย เพิ่มเข้าไปในประชากร
  - ไม่มีcrossover
  - Chromosomeที่มีค่าสูงสุดจะอยู่รอดไปถึงgenerationต่อไป
  - ตัวที่อยู่รอดที่เหลือถูกเลือกจากchromosomesที่เหลือ*แบบสุ่ม* ตามค่าfitness

- จากการทดลอง1000ครั้ง ส่วนผสมที่ทำให้คุณภาพของคุ๊กกี้ดีที่สุดถูกผลิตในgenerationที่16โดยเฉลี่ย
- ในจำนวนนี้ การทดลองที่โชคดีที่สุดผลิตchromosomeที่ดีที่สุดในgenerationที่8

Generation 0:

Chromosome	*Quality*
1 1	*1*

- จากmutation chromosome 1-2 ถูกผลิต

Generation 1:

Chromosome	*Quality*
1 2	*2*
1 1	*1*

- Chromosome 1-2กลายพันธุ์เป็น1-3 และ1-1เป็น 1-2ซึ่งมีอยู่แล้ว

Generation 2:

Chromosome	*Quality*
1 3	*3*
1 2	*2*
1 1	*1*

- Chromosome 1-3กลายพันธุ์เป็น1-4 และ1-2เป็น 2-2 และ1-1เป็น2-1 ซึ่งประชากรทั้งหมดมี6ตัว และ 4ตัวถูกเลือก

Generation 3:

Chromosome	*Quality*
1 4	*4*
1 3	*3*
1 2	*2*
2 1	*2*

- จากmutation ผลิตchromosomesใหม่3ตัว

chromosome	quality
2 4	5
2 3	4
3 1	3

Generation 4:

Chromosome	*Quality*
2 4	*5*
1 4	*4*
1 3	*3*
2 1	*2*

- ทุกchromosomesกลายพันธุ์ผลิตลูก

Generation 5:

Chromosome	*Quality*
2 5	*6*
1 5	*5*
2 3	*4*
2 2	*3*

- ทุกchromosomesกลายพันธุ์ผลิตลูก และ chromosome 1-5 อยู่รอดในgenerationหน้า

Generation 6:

Chromosome	*Quality*
3 5	*7*
1 5	*5*
3 2	*4*
1 4	*4*

Generation 7:

Chromosome	*Quality*
4 5	*8*
1 5	*5*
1 4	*4*
3 1	*3*

Generation 8:

Chromosome	*Quality*
5 5	*9*
4 5	*8*
2 5	*6*
2 1	*2*

- **3-5กลายพันธุ์เป็น4-5 และ3-2เป็น3-1**
  ส่วน1-4กลายพันธุ์เป็น1-5 และ1-5เป็น1-4

chromosome	quality
4 5	8
3 1	3

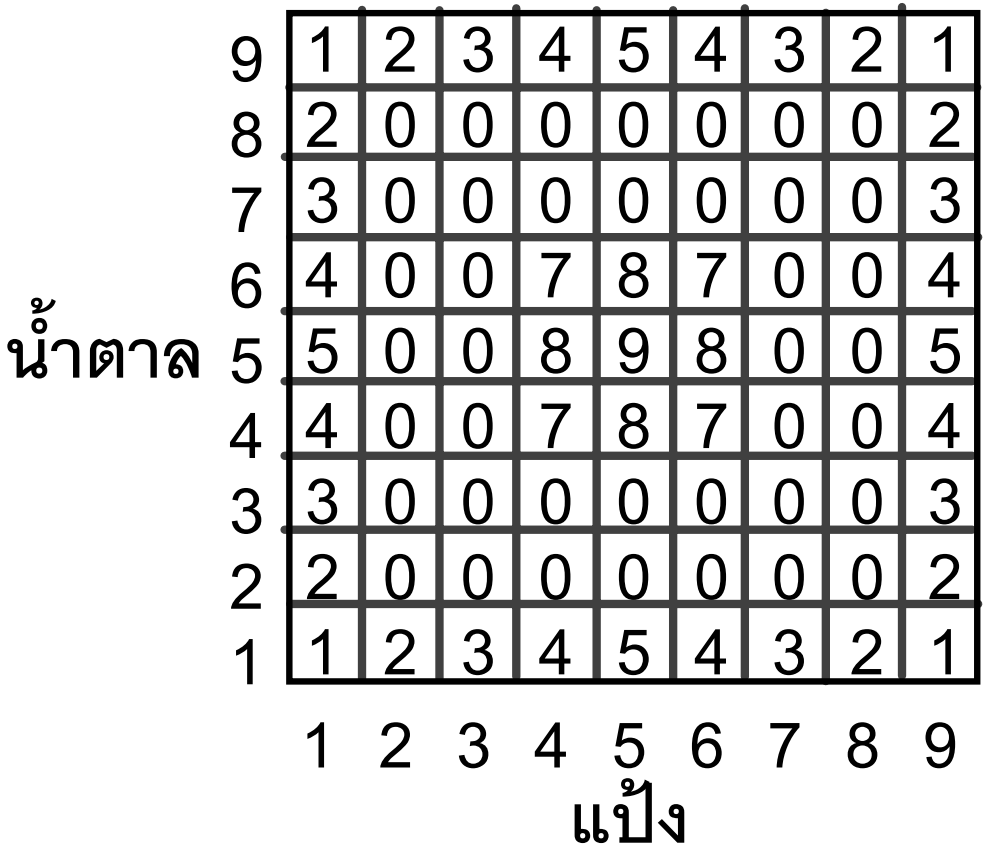
- **ที่จุดนี้ 4-5กลายพันธุ์เป็น5-5ซึ่งเป็นคำตอบที่ดีที่สุด**

Crossover Enables Genetic Algorithm to Traverse Obstructing Moats

Kookieต้องการดูว่า crossoverมีผลดีอย่างไร จึงทดลองดังนี้

- ทำcrossoverจากchromosomesที่อยู่รอดจากgenerationที่แล้ว
- เลือกคู่ผสมพันธุ์แบบสุ่ม
- สลับgeneของแต่ละคู่ผสมพันธุ์ และผลิตchromosomesลูก 2ตัว
  ถ้าchromosomesลูกยังไม่เคยมีมาเลย ให้เพิ่มเข้าไปในประชากร
  เพื่อแข่งขันที่จะอยู่รอดในgenerationหน้า

จากผลการทดลอง1000ครั้ง ส่วนผสมที่ดีที่สุดถูกผลิตในgeneration ที่14โดยเฉลี่ย เร็วกว่าไม่ใช้crossover 2 generation



Moat Mountain

- **สมมุติว่าคุณภาพของคุ๊กกี้เป็นฟังก์ชั่นของน้ำตาลและแป้งตามรูปบน**

- ในกรณีนี้ mutationจากgenerationที่อยู่ภายนอกmoat ไม่สามารถผลิตchromosomeที่อยู่ภายในmoatได้ เพราะว่าchromosomeตรงกลางจะมีค่าเป็น 0 ซึ่งไม่สามารถอยู่รอดในgenerationหน้าได้
- ถ้าจับคู่chromosomesพ่อแม่ที่เหมาะสมเช่น 1-5และ 5-1 เพื่อทำcrossover จะสามารถผลิตลูกที่ข้ามmoatได้
- จากการทดลอง1000ครั้ง ส่วนผสมที่ดีที่สุดถูกผลิตในgeneration ที่155โดยเฉลี่ย
- สาเหตุที่ผลไม่ดีนัก เพราะว่าก่อนที่chromosomeจะกลายพันธุ์เป็นchromosomesที่อยู่บริเวณ 1-5 หรือ 5-1นั้น โดยมากตายไปก่อนที่จะไปสู่บริเวณนั้นสำเร็จ

The Rank Method Links Fitness to Quality Rank

- Rank methodเป็นวิธีที่ใช้ควบคุมการเลือกchromosomes
- ไม่สนใจqualityว่ามีค่าเท่าไร เพียงแค่จัดลำดับเรียงตามquality ที่มีค่าสูงสุดจนถึงต่ำสุด
- กำหนดให้ p ค่าคงที่ค่าหนึ่ง เป็นความน่าจะเป็นที่chromosome ลำดับที่2 จะถูกเลือกเมื่อตัวที่อยู่ลำดับที่1ไม่ถูกเลือก และเป็นความน่าจะเป็นที่ลำดับที่3 จะถูกเลือกเมื่อลำดับที่2ไม่ถูกเลือก และจะเป็นดังนี้จนกระทั่งถึงลำดับสุดท้ายซึ่งจะถูกเลือกเมื่อลำดับก่อนหน้ามันไม่ถูกเลือกเลย

- สมมุติว่า p=0.667 และchromosomesที่เราสนใจอยู่คือ 1-4, 3-1, 1-2, 1-1 และ 7-5 (ในกรณีของ moat mountain)
  ตารางด้านล่างแสดงrank fitnessเทียบกับstandard fitness

Chromosome	Quality	Rank	Standard Fitness	Rank Fitness
1 4	4	1	0.40	0.667
1 3	3	2	0.30	0.222
1 2	2	3	0.20	0.074
1 1	1	4	0.10	0.025
7 5	0	5	0.00	0.012

- จากผลการทดลอง1000ครั้ง ส่วนผสมที่ดีที่สุดถูกผลิตในgeneration ที่75โดยเฉลี่ย ซึ่งแสดงให้เห็นว่าrank fitnessเหมาะสมกว่าstandard fitness จากการใช้rank fitness ตัวที่อยู่ตรงกลางในmoatสามารถอยู่รอดถึงgenerationหน้าได้

Survival of The Most Diverse

- Fitnessที่กล่าวถึงข้างต้นไม่ได้พิจารณาdiversity ซึ่งเป็นตัวทำให้ chromosomesแสดงgenesที่ต่างไปจากเดิม และบ่อยครั้งในธรรม-ชาติที่speciesซึ่งลักษณะแตกต่างไปจากspeciesที่fitกับธรรมชาติสามารถอยู่รอดได้ดี

The Rank-Space Method Links Fitness to Both Quality Rank and Diversity Rank

- ในการที่จะวัดdiversityของchromosomeหนึ่งๆ ทำได้โดยคำนวณ 1/ระยะห่างกำลังสองระหว่างchromosomeนั้นกับchromosomesอื่นที่ถูกเลือกแล้วว่าให้อยู่รอดในgenerationต่อไป

$$\sum_i \frac{1}{d_i^2}$$

- พิจารณาchromosomes 5-1, 1-4, 3-1, 1-2, 1-1 และ 7-5 ตัวที่มีค่าqualityสูงสุดคือ 5-1 ตารางด้านล่างแสดงลำดับของ5ตัวที่เหลือ โดยเรียงตามqualityและ1/ระยะห่างกำลังสองจาก 5-1

Chromosome	Score	$1/d^2$	Diversity Rank	Quality Rank
1 4	4	0.040	1	1
3 1	3	0.250	5	2
1 2	2	0.059	3	3
1 1	1	0.062	4	4
7 5	0	0.050	2	5

- วิธีที่ง่าย ๆในการcombine quality rank กับ diversity rank คือ การบวกrankทั้งสองเข้าด้วยกันเป็นcombined rank

- เมื่อได้combined rankซึ่งคิดทั้งqualityและdiversityแล้ว การเลือกกระทำได้เหมือนเดิมโดยกำหนดfitnessของตัวแรกเป็น p (ดูตาราง)

Chromosome	Rank Sum	Combined Rank	Fitness
1 4	2	1	0.667
3 1	7	4	0.025
1 2	6	2	0.222
1 1	8	5	0.012
7 5	7	3	0.074

- จากตาราง เราเลือก chromosome 1-4 เป็นตัวที่สองต่อจาก 5-1
- หลังจากนี้เราจะเลือกอีก2ตัวที่เหลือ ในครั้งนี้เราต้องคำนวณหา 1/ระยะห่างกำลังสอง โดยคิดทั้ง 5-1และ1-4 (ดูตารางถัดไป)

Chromosome	$\sum_i \frac{1}{d_i^2}$	Diversity Rank	Quality Rank	Combined Rank	Fitness
3 1	0.327	4	1	4	0.037
1 2	0.309	3	2	3	0.074
1 1	0.173	2	3	2	0.222
7 5	0.077	1	4	1	0.667

- **สมมุติว่า**chromosome 7-5**ถูกเลือก และการเลือก**chromosome **สุดท้ายทำได้โดยใช้ตารางด้านล่างนี้**

Chromosome	$\sum_i \frac{1}{d_i^2}$	Diversity Rank	Quality Rank	Combined Rank	Fitness
3 1	0.358	3	1	3	0.111
1 2	0.331	2	2	2	0.222
1 1	0.190	1	3	1	0.667

- Rank-space methodนี้ แตกต่างจากstandard methodที่ standard methodจะไม่เลือก7-5เลย แต่วิธีนี้7-5อาจถูกเลือกได้ สรุปจากวิธีนี้ chromosome 5-1, 1-4, 7-5, 1-1 ถูกเลือกตามลำดับ แต่ถ้าเป็น standard methodจะเลือก 5-1, 1-4, 3-1, 1-2ตามลำดับ
- จากการทดลอง1000ครั้งโดยใช้rank-space method ด้วยp=0.66 เริ่มจากchromosome 1-1 คำตอบที่ดีที่สุดถูกผลิตได้ในgeneration ที่15โดยเฉลี่ย
- Rank-space methodสามารถหาคำตอบที่ดีที่สุดได้ดีกว่าวิธีอื่นๆ ในปัญหาของmoat mountain (ดูตาราง)

Mountain	Standard Method	Quality Rank	Rank Space
Bump	14	12	12
Moat	155	75	15

- ในจำนวนการทดลอง1000ครั้งโดยใช้rank-space method ครั้งที่ดีที่สุด chromosome 5-5ถูกผลิตในgenerationที่7

Generation 0:

Chromosome	*Quality*
1 1	*1*

- จากmutation chromosome 2-1ถูกผลิต

Generation 1:

Chromosome	*Quality*
2 1	*2*
1 1	*1*

- Mutationผลิตchromosome 3-1 ส่วนcrossover ไม่มีผลในกรณีนี้เพราะว่าgenesตัวที่สองของ chromosomesทั้งสองมีค่าเหมือนกัน

Generation 2:

Chromosome	*Quality*
3 1	*3*
2 1	*2*
1 1	*1*

- Mutationผลิตchromosome 4-1และ2-2 ส่วนcrossoverยังคงไม่มีผล ในการเลือกครั้งนี้ chromosome 2-1ไม่ถูกเลือก

Generation 3:

Chromosome	*Quality*
4 1	*4*
3 1	*3*
1 1	*1*
2 2	*0*

- Mutationผลิตchromosomesใหม่3ตัวคือ 5-1, 1-2, 2-3 ส่วนcrossoverของ2-2กับ4-1ผลิต2-1และ4-2

chromosome	quality
5 1	5
1 2	2
2 3	0
2 1	2
4 2	0

Generation 4:

Chromosome	*Quality*
5 1	*5*
3 1	*3*
1 2	*2*
2 3	*0*

- Mutationผลิต6-1, 3-2, 2-2, 2-4 ส่วนcrossover ผลิต2-1, 1-1, 5-2, 3-2, 5-3

chromosome	quality
6 1	4
3 2	0
2 2	0
2 4	0
2 1	2
1 1	1
5 2	0
3 2	0
5 3	0

Generation 5:

Chromosome	*Quality*
5 1	*5*
3 1	*3*
1 2	*2*
2 4	*0*

- ที่จุดนี้เกิดcrossoverของ5-1กับ2-4ได้5-4ซึ่งเป็นchromosomeที่ดีในgenerationหน้า

Generation 6:

Chromosome	*Quality*
5 4	*8*
1 4	*4*
3 1	*3*
1 2	*2*

- และในท้ายที่สุด 5-4กลายพันธุ์เป็น5-5

Generation 7:

Chromosome	*Quality*
5 5	*9*
1 4	*4*
1 2	*2*
5 2	*0*

(0)

1	2	3	4	5	4	3	2	1
2	0	0	0	0	0	0	0	2
3	0	0	0	0	0	0	0	3
4	0	0	7	8	7	0	0	4
5	0	0	8	9	8	0	0	5
4	0	0	7	8	7	0	0	4
3	0	0	0	0	0	0	0	3
2	0	0	0	0	0	0	0	2
1	2	3	4	5	4	3	2	1

(1)

1	2	3	4	5	4	3	2	1
2	0	0	0	0	0	0	0	2
3	0	0	0	0	0	0	0	3
4	0	0	7	8	7	0	0	4
5	0	0	8	9	8	0	0	5
4	0	0	7	8	7	0	0	4
3	0	0	0	0	0	0	0	3
2	0	0	0	0	0	0	0	2
1	2	3	4	5	4	3	2	1

(2)

1	2	3	4	5	4	3	2	1
2	0	0	0	0	0	0	0	2
3	0	0	0	0	0	0	0	3
4	0	0	7	8	7	0	0	4
5	0	0	8	9	8	0	0	5
4	0	0	7	8	7	0	0	4
3	0	0	0	0	0	0	0	3
2	0	0	0	0	0	0	0	2
1	2	3	4	5	4	3	2	1

(3)

1	2	3	4	5	4	3	2	1
2	0	0	0	0	0	0	0	2
3	0	0	0	0	0	0	0	3
4	0	0	7	8	7	0	0	4
5	0	0	8	9	8	0	0	5
4	0	0	7	8	7	0	0	4
3	0	0	0	0	0	0	0	3
2	0	0	0	0	0	0	0	2
1	2	3	4	5	4	3	2	1

(4)

1	2	3	4	5	4	3	2	1
2	0	0	0	0	0	0	0	2
3	0	0	0	0	0	0	0	3
4	0	0	7	8	7	0	0	4
5	0	0	8	9	8	0	0	5
4	0	0	7	8	7	0	0	4
3	0	0	0	0	0	0	0	3
2	0	0	0	0	0	0	0	2
1	2	3	4	5	4	3	2	1

(5)

1	2	3	4	5	4	3	2	1
2	0	0	0	0	0	0	0	2
3	0	0	0	0	0	0	0	3
4	0	0	7	8	7	0	0	4
5	0	0	8	9	8	0	0	5
4	0	0	7	8	7	0	0	4
3	0	0	0	0	0	0	0	3
2	0	0	0	0	0	0	0	2
1	2	3	4	5	4	3	2	1

(6)

1	2	3	4	5	4	3	2	1
2	0	0	0	0	0	0	0	2
3	0	0	0	0	0	0	0	3
4	0	0	7	8	7	0	0	4
5	0	0	8	9	8	0	0	5
4	0	0	7	8	7	0	0	4
3	0	0	0	0	0	0	0	3
2	0	0	0	0	0	0	0	2
1	2	3	4	5	4	3	2	1

(7)

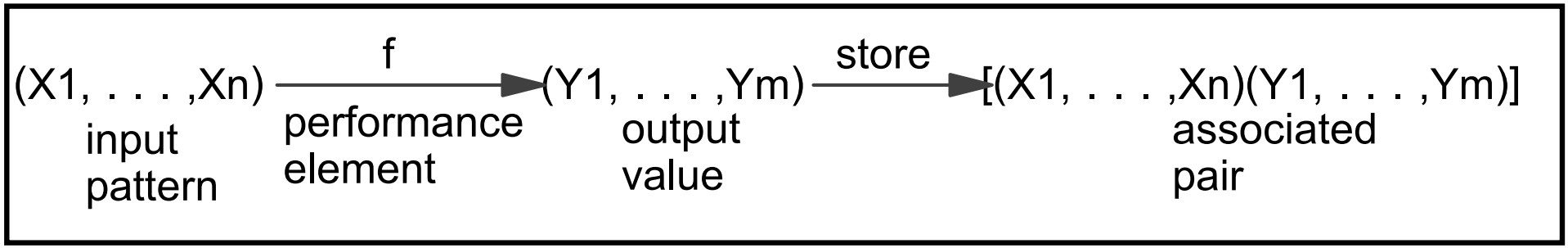
1	2	3	4	5	4	3	2	1
2	0	0	0	0	0	0	0	2
3	0	0	0	0	0	0	0	3
4	0	0	7	8	7	0	0	4
5	0	0	8	9	8	0	0	5
4	0	0	7	8	7	0	0	4
3	0	0	0	0	0	0	0	3
2	0	0	0	0	0	0	0	2
1	2	3	4	5	4	3	2	1

Local Maxima Are Easier to Handle when Diversity Is Maintained

- Searchทั่วไปพยายามหลีกเลี่ยงlocal maxima
- Genetic algorithmจะให้diversityเป็นส่วนประกอบของ fitness แล้วผลิตindividualsที่อยู่ห่างออกจากlocal maxima
- เมื่อมีindividualsอยู่ทั่วlocal maximaทุกตัวแล้ว ก็จะมีโอกาสที่ individualsเหล่านี้จะหาทางไปglobal maximaได้ในที่สุด

5.2 Rote Learning

- Rote learning เป็นวิธีการเรียนรู้แบบง่ายที่สุดโดย 'การจำ'
- เก็บความรู้ใหม่ไว้ใน memory และเมื่อต้องการก็เพียงแค่ดึงความรู้นั้นมาใช้ แทนที่จะคำนวณหรือ inference หาความรู้นั้น
- เราสามารถมองว่า performance element เป็นฟังก์ชั่นที่เปลี่ยน input pattern (X1, . . . ,Xn) ไปเป็น output value (Y1, . . . ,Ym) ดังนั้น rote learning ก็คือการเก็บคู่ลำดับ [(X1,...,Xn),(Y1,...,Ym)] ไว้ใน memory หลังจากนั้น เมื่อเราต้องการหา f(X1, . . . ,Xn) ก็ทำโดยการดึง (Y1, . . . ,Ym) จากคู่ลำดับนี้เท่านั้น



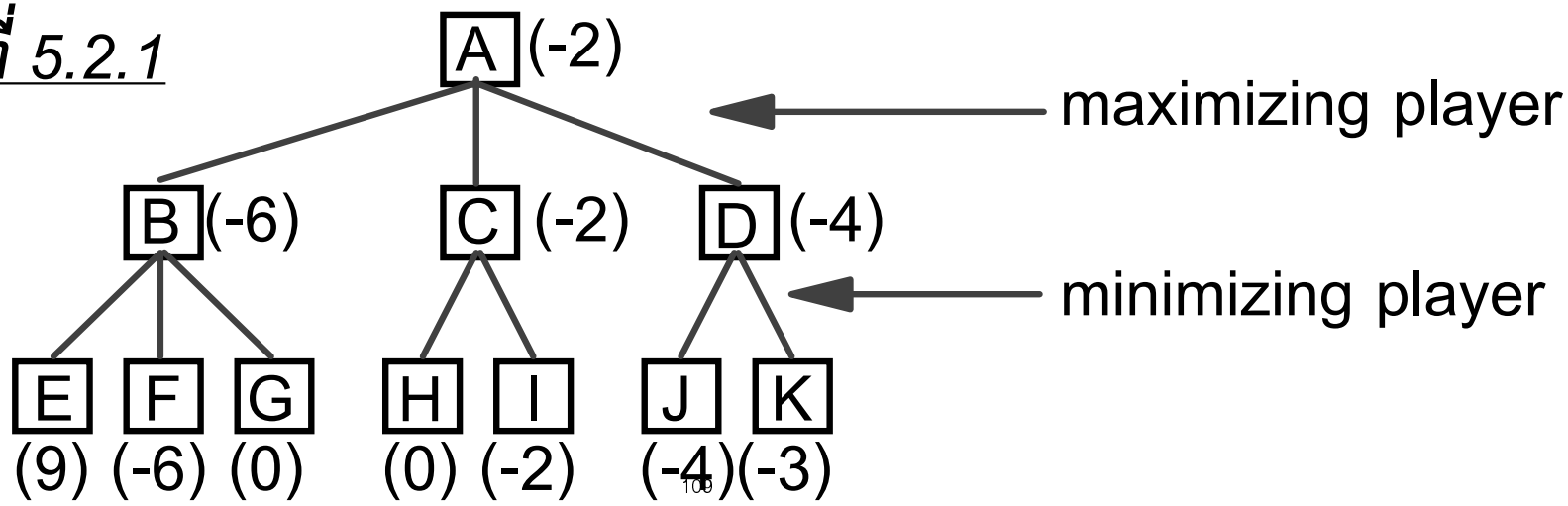
- **สิ่งที่ต้องพิจารณาในการทำ** rote learning
  - memory organization : **ต้องดีเพื่อให้ดึงความรู้ได้รวดเร็ว**
  - stability of environment : rote learning **ใช้ไม่ได้ในกรณีที่** environment **เปลี่ยน (ความรู้ไม่** valid **ใน** environment**ใหม่)**
  - store-versus-compute trade off : rote learning **ต้องไม่ลด** performance **ของระบบ (**cost **ของการเก็บและดึงความรู้ต้องถูกกว่า** cost **ของการคำนวณหรือ** inference **ความรู้)**

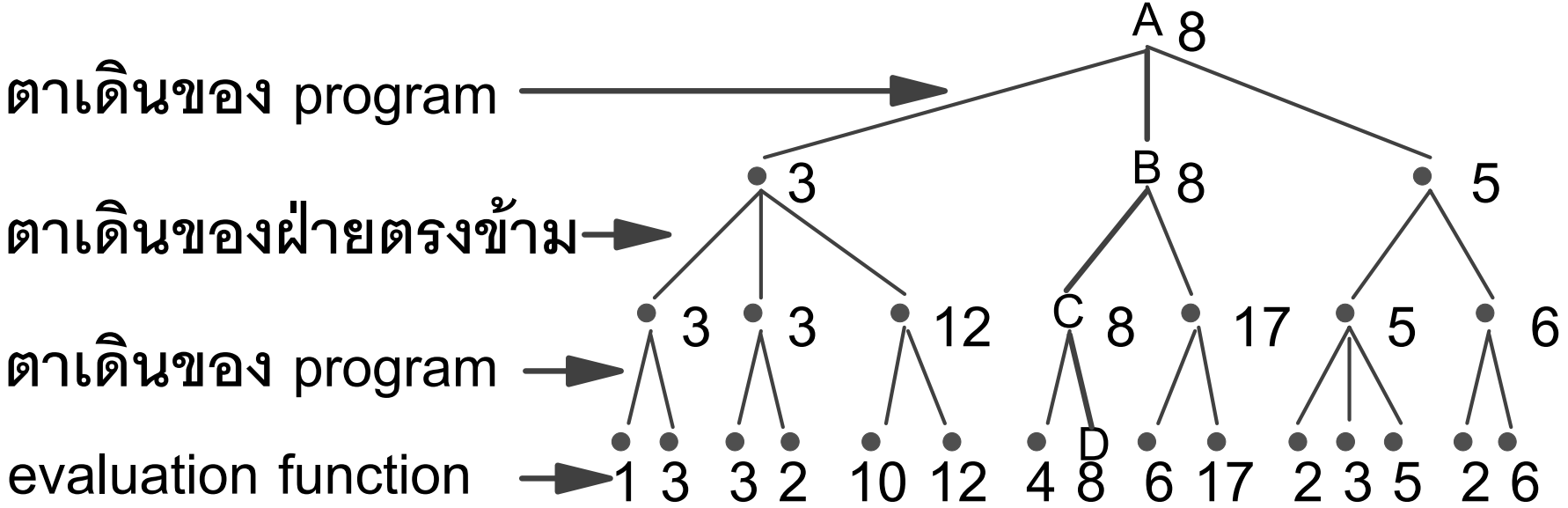
Rote learning **ใน** Checkers **ของ** Samuel

- Checkers **เป็นเกมส์ที่ยากในการเรียนให้เก่ง**
- **ตาเดินที่เป็นไปได้ทั้งหมดใน** checkers **มีประมาณ** 10^{40}
- Checkers program **ใช้** minimax game-tree search **ในการเลือกตาเดินครั้งต่อไป**
- **ใช้** static evaluation function (heuristic function) **เพื่อประเมินความดี (โอกาสชนะ) ของ** node **ใน** game-tree **ซึ่งแต่ละ** node **แสดง** board position

- Minimax game-tree เป็น tree ที่ใช้ในการเล่นเกมส์ที่มีผู้เล่น 2 คน (player 1 และ player 2) ดูรูปที่ 5.2.1
- Player 1 หรือ maximizing player เริ่มเล่นจาก node A ซึ่งเมื่อเดินไป 1 ก้าว ได้ nodes ลูกเป็น B C หรือ D
- ที่แต่ละ node B C D, player 2 หรือ minimizing player เดิน 1 ก้าว ได้ nodes ลูกเป็น E F G , H I และ J K ตามลำดับ
- ที่จุดนี้ เราคำนวณ static evaluation function ของแต่ละ node
- ค่าของ function จะถูก backed up ขึ้นมาให้ node แม่ โดยที่ minimizing player พยายามเลือก node ที่มีค่า function น้อยที่สุด ในขณะที่ maximizing player เลือกตัวที่มีค่ามากที่สุด

รูปที่ 5.2.1





รูปที่ 5.2.2 ตัวอย่างของ minimax game-tree search

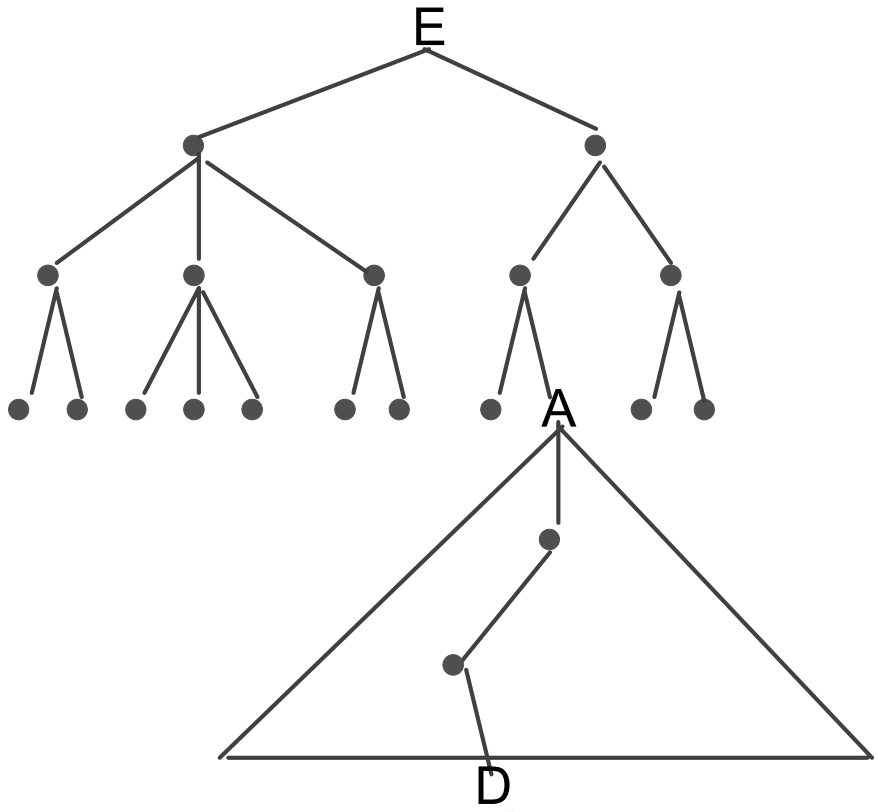
- ให้ A เป็น board position ซึ่งต่อไปเป็นตาเดินของ program
- Program ค้นหาทุกทางที่เป็นไปได้ล่วงหน้า 3 ตาเดิน (look-ahead) โดยที่ ตาแรกเป็นของ program ตาที่ 2 เป็นของฝ่ายตรงข้าม และ ตาที่ 3 เป็นของ program ที่จุดนี้ ใช้ static evaluation function วัดค่าของแต่ละ board postition

- ในกรณีของ ตาเดินของ program ค่าของ board position ที่ดีที่สุด จะถูก backed-up ขึ้นมาให้เป็น backed-up value ของ node ที่อยู่ด้านบน ในกรณีของ ตาเดินของฝ่ายตรงข้าม ฝ่ายตรงข้ามจะเลือกตาเดินที่ทำให้ค่า function มีค่าน้อยที่สุดเท่าที่จะทำได้ คือ ค่าของ board position ที่แย่ที่สุดจะถูกส่งขึ้นมาให้ node ที่อยู่ด้านบน
- ในรูปที่ 5.2.2 ตาเดินที่ดีที่สุดของ program คือไป B ซึ่งฝ่ายตรงข้ามจะเดินไปยัง C และ program จะเดินไปยัง D ตามลำดับ

<u>การพัฒนาความสามารถของ program</u>

- เราสามารถพัฒนาความสามารถของ program ให้มีประสิทธิภาพเพิ่มขึ้นได้ โดยการเพิ่มจำนวน look-ahead ในตัวอย่างด้านบน program คำนวณ look-ahead 3 ก้าว และถ้าเพิ่มได้มากเท่าไร ค่าของ function จะยิ่งถูกต้องเพิ่มขึ้นเท่านั้น

- Rote learning ใน checkers program นี้พัฒนาความสามารถของการ look-ahead โดยการจำ backed-up value ของ board position
- Program จะเก็บ board position ระหว่างการเล่น คู่กับ backed-up value เช่น [A,8] และเมื่อ A ถูกค้นพบอีกครั้งใน game-tree ค่าของ A จะถูกเรียกมาใช้ได้โดยไม่ต้องคำนวณใหม่ (เท่ากับ 8)



รูปที่ 5.2.3 rote learning เพิ่มประสิทธิภาพของ look-ahead

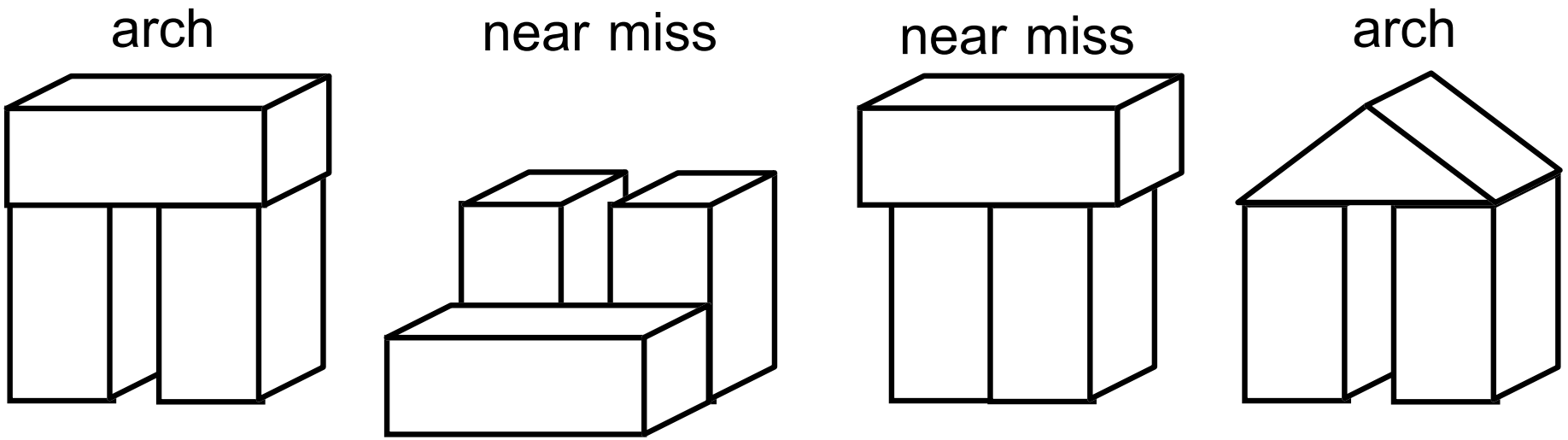
- การที่เราไม่ต้องคำนวณค่าของ A ใหม่ ทำให้ program มีประสิทธิภาพในการคำนวณเพิ่มขึ้น (เร็วขึ้น) นอกจากนั้นแล้ว สิ่งที่สำคัญอีกอย่างในการใช้ backed-up value คือ ค่า backed-up value ของ A มีความถูกต้องสูงกว่าการคำนวณค่า static evaluation function ของ A เพราะว่ามันรวมการ look-ahead ไว้ด้วย
- ในตัวอย่างรูปที่ 5.2.3 program กำลังตัดสินใจว่าจะเลือกตาเดินใดดีที่ position E
- Program ค้นหาล่วงหน้า 3 ตาเดิน แล้วใช้ evaluation function คำนวณความดีของแต่ละ node แต่ในกรณีของ node A ใช้ค่า backed-up value แทนการคำนวณ (ซึ่งเป็นค่าของ node D)
- เมื่อ program จำค่า backed-up value ไว้มากเท่าไร ความลึกของ search ก็จะมากขึ้นเท่านั้น (จาก 3 เป็น 6 เป็น 9 . . .)

memory organization ที่ใช้

- เพื่อเป็นการประหยัด space ของ memory เลือกเก็บเฉพาะ board position ของตาเดินของ player 1 ซึ่ง board position ที่เก็บไว้นี้สามารถใช้กับในกรณีของ player 2 ได้โดยการ convert
- การดึง board position มาใช้ทำได้โดยใช้ characteristics ของ board (เช่น จำนวนของเบี้ยบน board, มี king หรือไม่มี, . . .)
- จัดการกับปัจหาของ store-versus compute trade off โดยใช้วิธีที่เรียกว่า *least recently used replacement*
  - ให้แต่ละ board position ที่เก็บไว้มี *อายุ*
  - ทุกครั้งที่ board position ถูกดึงมาใช้ อายุของมันจะถูกหาร 2
  - ทุกครั้งที่มีการเก็บ position ใหม่ใน memory อายุของทุก positions เดิมจะถูกบวกด้วย 1
  - ตัวที่มีอายุมากที่สุดจะถูกลบออกจาก memory

5.3 Learning by Analyzing Differences

- โปรแกรมของ Winston[1975] เรียน structural concept ใน block world domain เช่น *arch, tent, house*
- เป็นการเรียนรู้โดยการวิเคราะห์ความแตกต่างที่ปรากฏในลำดับของตัวอย่างที่ให้
- ตัวอย่างที่ให้ประกอบด้วย ตัวอย่างบวก และ ตัวอย่างลบแบบ near miss
- ผู้สอนต้องให้ตัวอย่างตามลำดับซึ่งเตรียมไว้เป็นอย่างดีเรียงทีละตัว
- โปรแกรมจะเรียน concept หรือแก้ไข concept เมื่อได้รับตัวอย่างแต่ละตัว
- รูปที่ 5.3.1 แสดง ตัวอย่างบวก และ near miss ของ concept "arch"

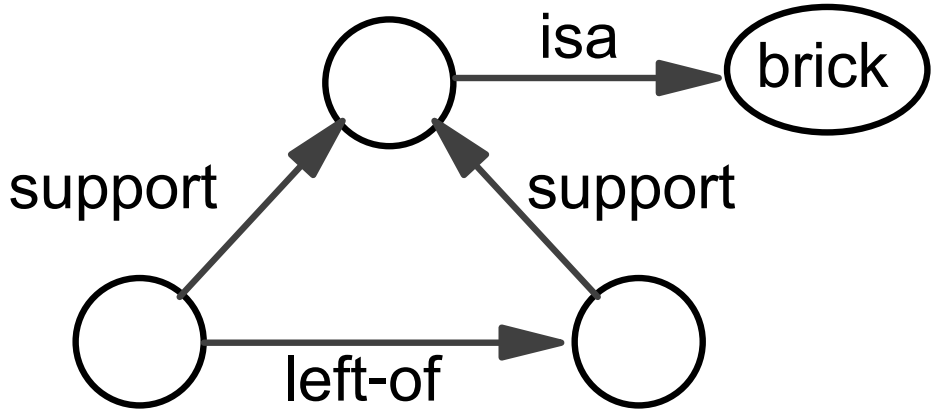


รูปที่ 5.3.1 ตัวอย่างบวกและ near miss ของ arch

- จากตัวอย่างทั้ง 4 โปรแกรมเรียนความรู้ว่าอะไรคือ arch
- ตัวอย่างแรกบอกให้รู้ว่า arch คือสิ่งที่ประกอบด้วย bricks แนวตั้ง และ brick แนวนอนที่ถูกรองรับโดย bricks แนวตั้ง
- ตัวอย่างที่ 2 ซึ่งเป็น near miss อธิบายสิ่งที่ไม่ใช่ arch : สิ่งที่ประกอบด้วย bricks แนวตั้ง และ brick แนวนอนซึ่งไม่ถูกรองรับโดย bricks แนวตั้ง
- ตัวอย่างที่ 3 เป็น near miss และตัวอย่างที่ 4 เป็น arch

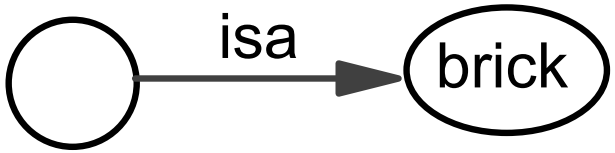
- จากตัวอย่างที่ 1 โปรแกรมสร้าง description (ในรูปของ semantic network) ซึ่งแสดงดังรูป และใช้เป็น initial description

รูปที่ 5.3.2



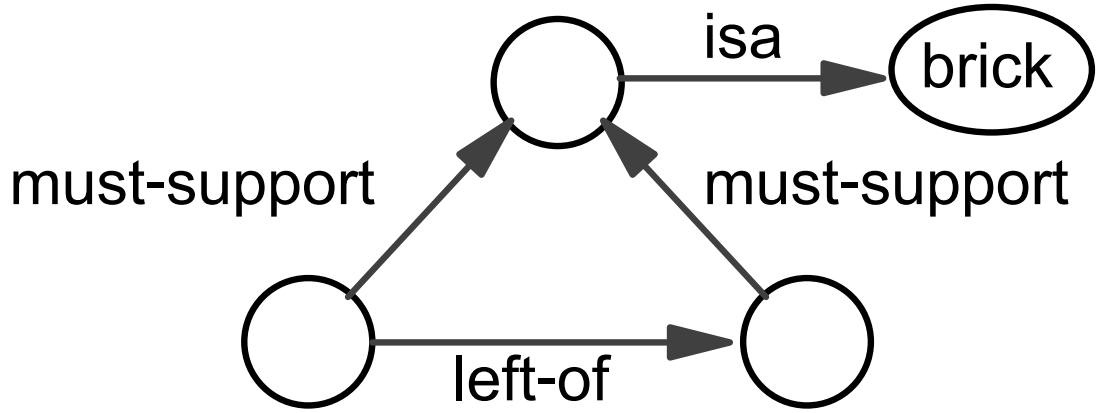
- จากตัวอย่างที่ 2 โปรแกรมสร้าง description ของ near miss ดังรูป

รูปที่ 5.3.3



- โปรแกรมหาความแตกต่าง (difference) ของ 2 ตัวอย่าง และรู้ว่า support links จำเป็นสำหรับ concept arch

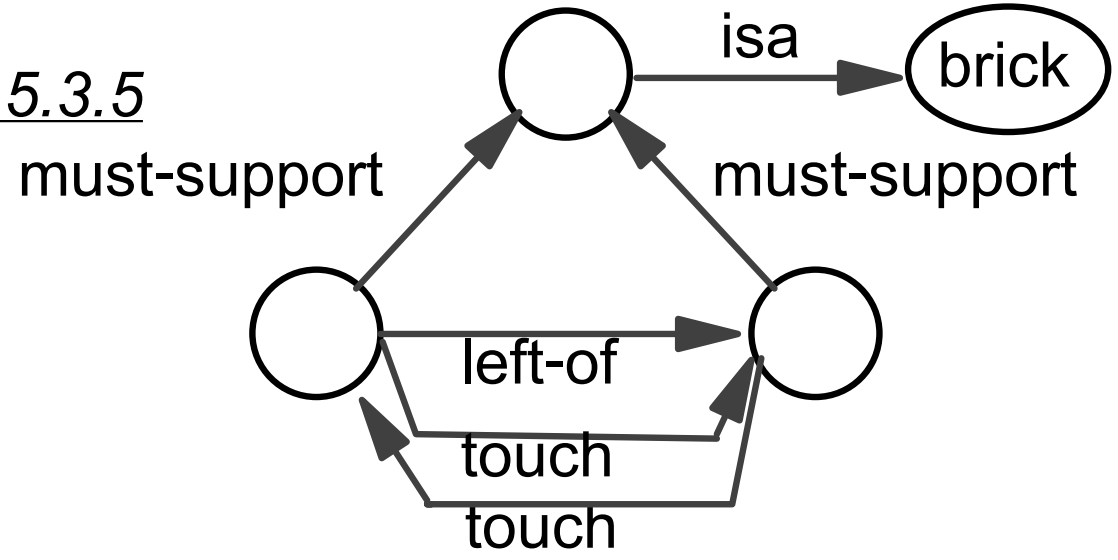
- โปรแกรมสร้าง links ใหม่ชื่อ must-supports แทนที่ links เดิม กรณีนี้ near miss ให้ information สำหรับ *require-link heuristic* และเราเรียก description ใหม่ที่ได้ว่า *evolving model*



รูปที่ 5.3.4

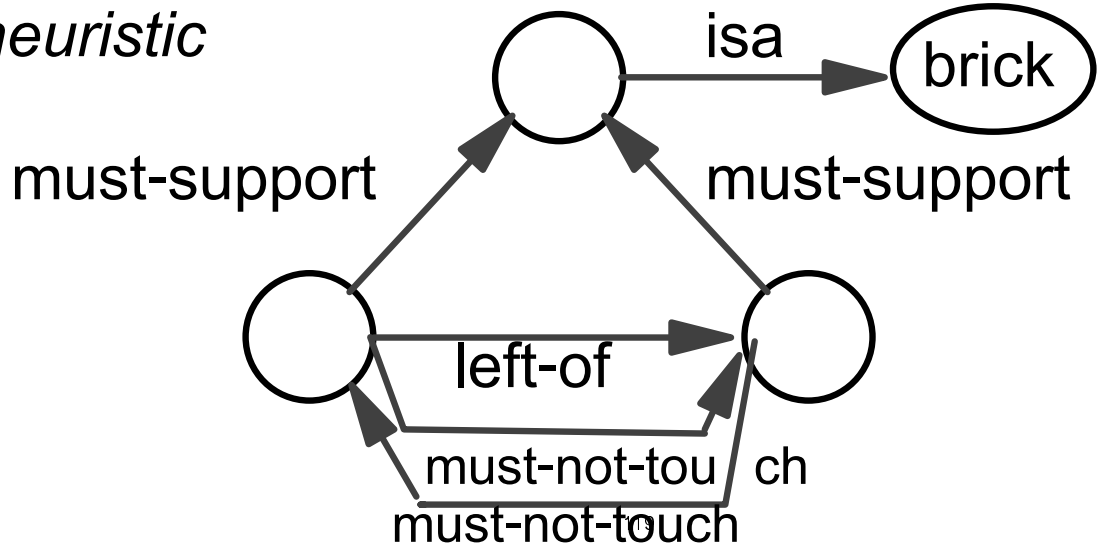
- near-miss เป็นตัวอย่างที่ให้ information สำหรับ specialize concept
- จากตัวอย่างที่ 3 โปรแกรมสร้าง description ได้ดังรูปที่ 5.3.5

รูปที่ 5.3.5



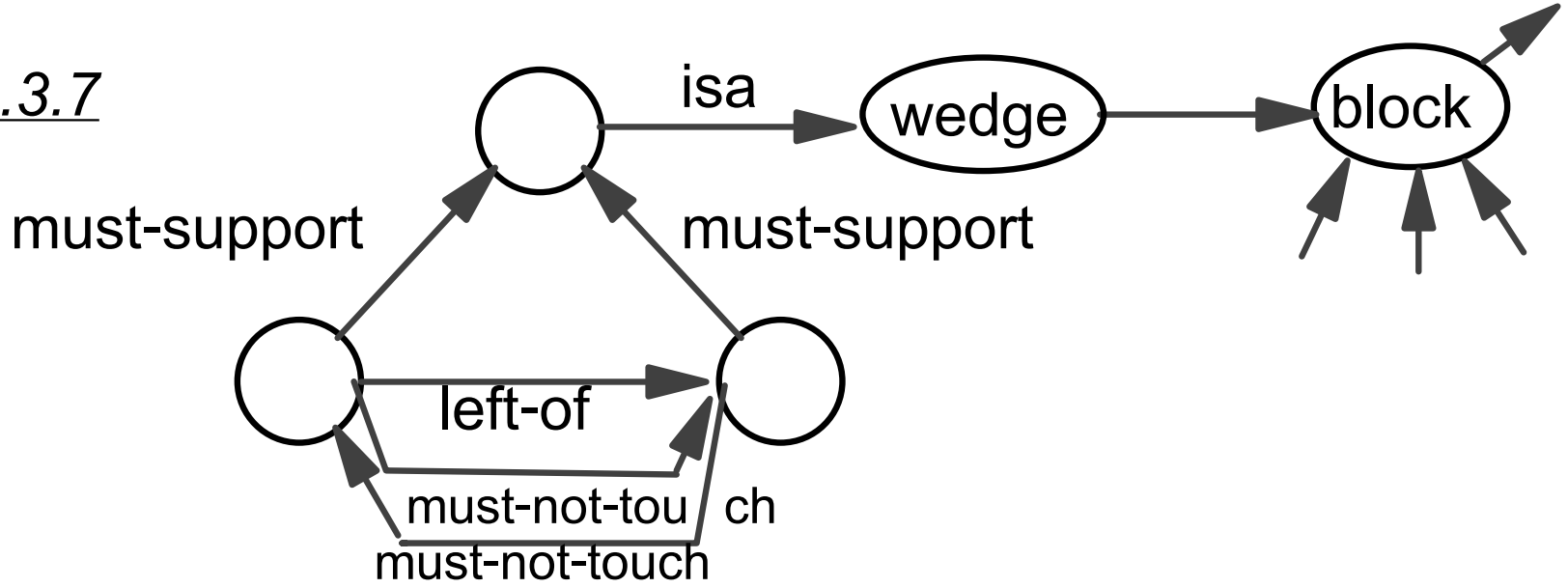
- จากรูปที่ 5.3.4 และ 5.3.5 โปรแกรมหาความแตกต่างได้คือ มี touch links อยู่ใน near miss ซึ่งแสดงให้รู้ว่าที่จริงแล้วไม่ควรมี
- โปรแกรมเพิ่ม links เข้าไปใน evolving model เพื่อสร้าง model ใหม่ (รูปที่ 5.3.6) กรณีนี้ near miss ให้ information สำหรับ *forbid-link heuristic*

รูปที่ 5.3.6



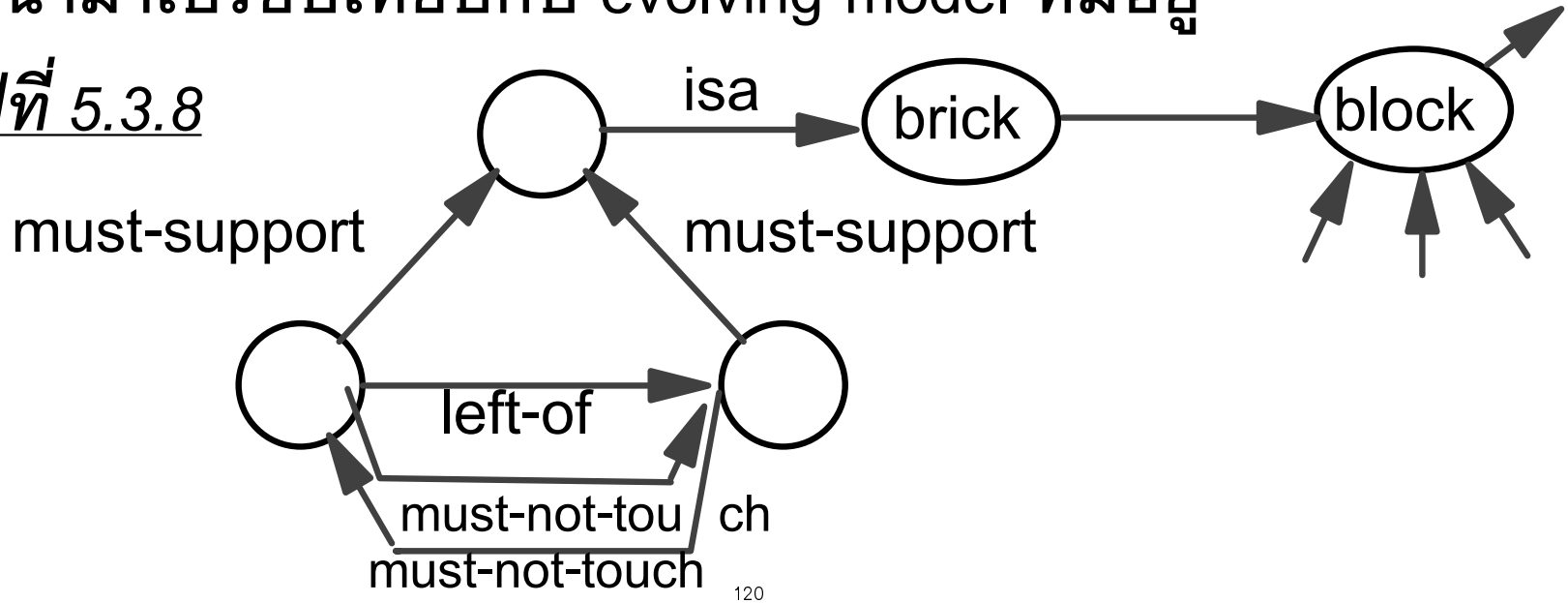
- จากตัวอย่างที่ 4 โปรแกรมสร้าง description ได้ดังรูปที่ 5.3.7

รูปที่ 5.3.7

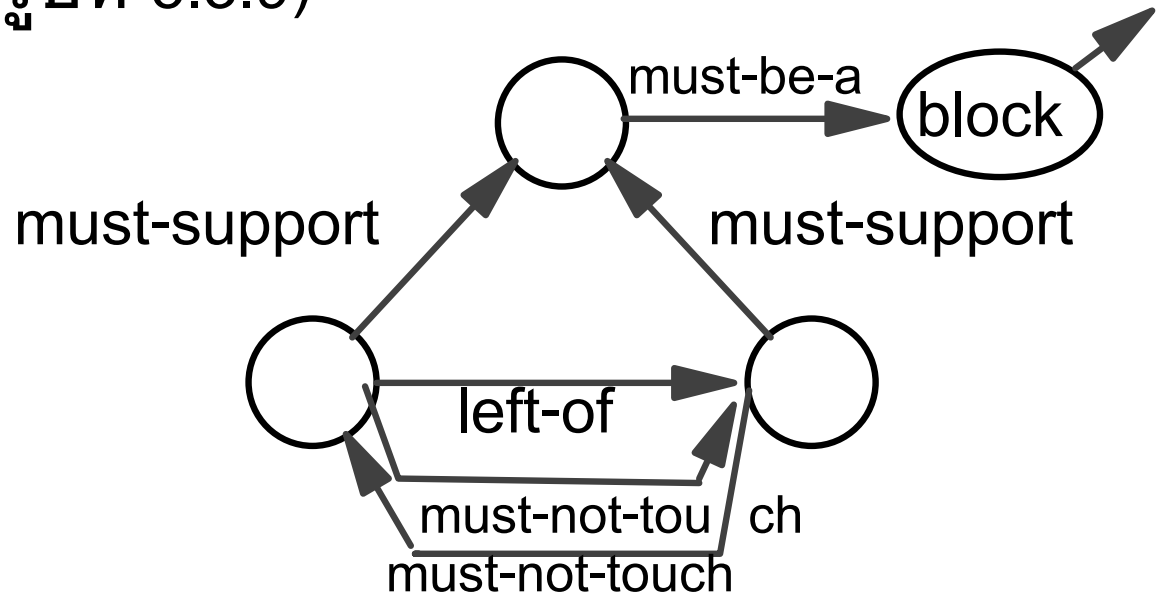


- เมื่อนำมาเปรียบเทียบกับ evolving model ที่มีอยู่

รูปที่ 5.3.8



- เราสร้าง evolving model ได้จากการแทนที่ brick และ wedge ด้วย *class* ที่ general กว่า (ในกรณีที่เรามี information ส่วนนี้อยู่ด้วย) และ isa link ด้วย must-be-a link ในกรณีนี้โปรแกรมใช้ *climb-tree heuristic* (รูปที่ 5.3.9)



รูปที่ 5.3.9

- ในกรณีที่เราไม่มี information ที่แสดง *classification tree* โปรแกรมจะสร้าง class ใหม่ "brick-or-wedge" class ขึ้นเพื่อใช้แทน "block" ในกรณีนี้โปรแกรมใช้ *enlarge-set heuristic*

- และถ้าหากว่าในกรณีที่ ไม่มี objects อื่นอีกเลยยกเว้นแต่ brick กับ wedge โปรแกรมจะตัด isa link ออก ในกรณีนี้โปรแกรมใช้ *drop-link heuristic*

Algorithm ของโปรแกรม

- near-miss ใช้สำหรับ specialize model โดยใช้
  - require link heuristic
  - forbid link heuristic
- positive example ใช้สำหรับ generalize model โดยใช้
  - climb-tree heuristic
  - enlarge-set heuristic
  - drop-link heuristic

Specialization :

Specialization to make a model more restrictive,

- Make the evolving model to the example to establish correspondences among parts
- Determine whether there is a single, most important difference between the evolving model and the near miss
  - If there is a single, most important difference,
    (a) If the evolving model has a link that is not in the near miss, use the require-link heuristic
    (b) If the near miss has a link that is not in the model, use the forbid-link heuristic
  - Otherwise, ignore the example

Generalization :

Generalization to make a model more permissive,

- Match the evolving model to the example to establish correspondences among parts
- For each difference, determine the difference type:
  - If a link points to a class in the evolving model different from the class to which the link points in the example,
    (a) If the classes are part of a classification tree, use the climb-tree heuristic
    (b) If the classes form an exhaustive set, use the drop-link heuristic
    (c) Otherwise, use the enlarge-set heuristic
- If a link is missing in the example, use the drop-link heuristic
- Otherwise, ignore the difference

5.4 Version Spaces

- Mitchell [1977,1978] เสนอวิธีการที่เรียกว่า version spaces
- เรียน description ที่อธิบายตัวอย่างบวกและไม่อธิบายตัวอย่างลบ
- รูปที่ 5.4.1 คือ ตัวอย่างบวกของการเรียน concept 'Car' ซึ่งแสดงโดย frame

Car023

origin :	Japan
manufacturer :	Honda
color :	Blue
decade :	1970
type :	Economy

รูปที่ 5.4.1 ตัวอย่างบวกของ concept 'Car'

- สมมุติว่าแต่ละ slot ของ frame เป็นค่าที่แสดงในรูปที่ 5.4.2

origin ∈ { Japan, USA, Britain, Germany, Italy }
manufacturer ∈ { Honda, Toyota, Ford, Chrysler, Jaguar, BMW, Fiat }
color ∈ { Bule, Green, Red, White }
decade ∈ { 1950, 1960, 1970, 1980, 1990, 2000 }
type ∈ { Economy, Luxury, Sports }

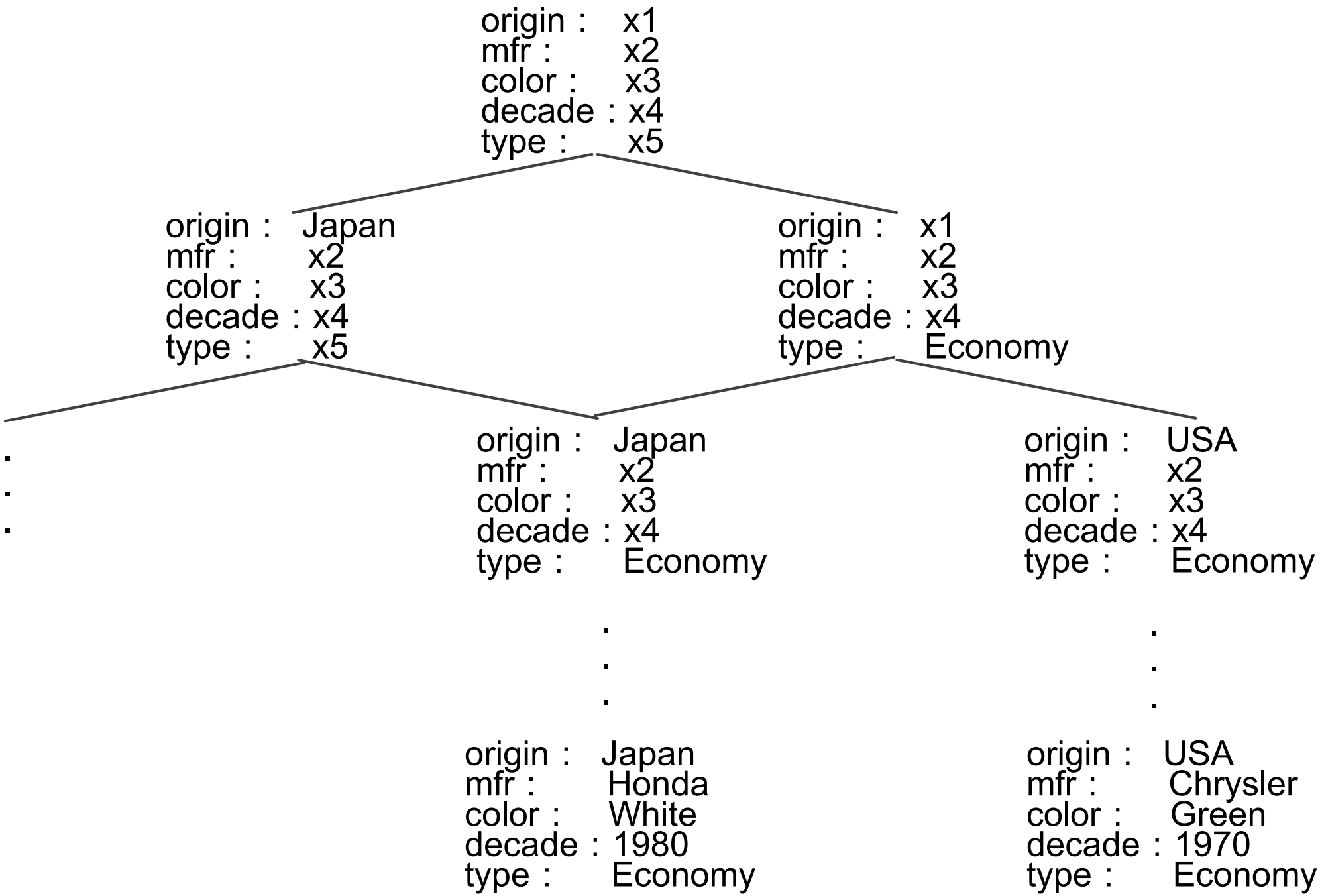
รูปที่ 5.4.2 ภาษาที่ใช้แสดง

- Concept description แสดงในรูปของ slots และ slot values เช่น concept ของ "Japanese economy car" แสดงได้ดังรูปที่ 5.4.3

origin : Japan
manufacturer : x1
color : x2
decade : x3
type : Economy

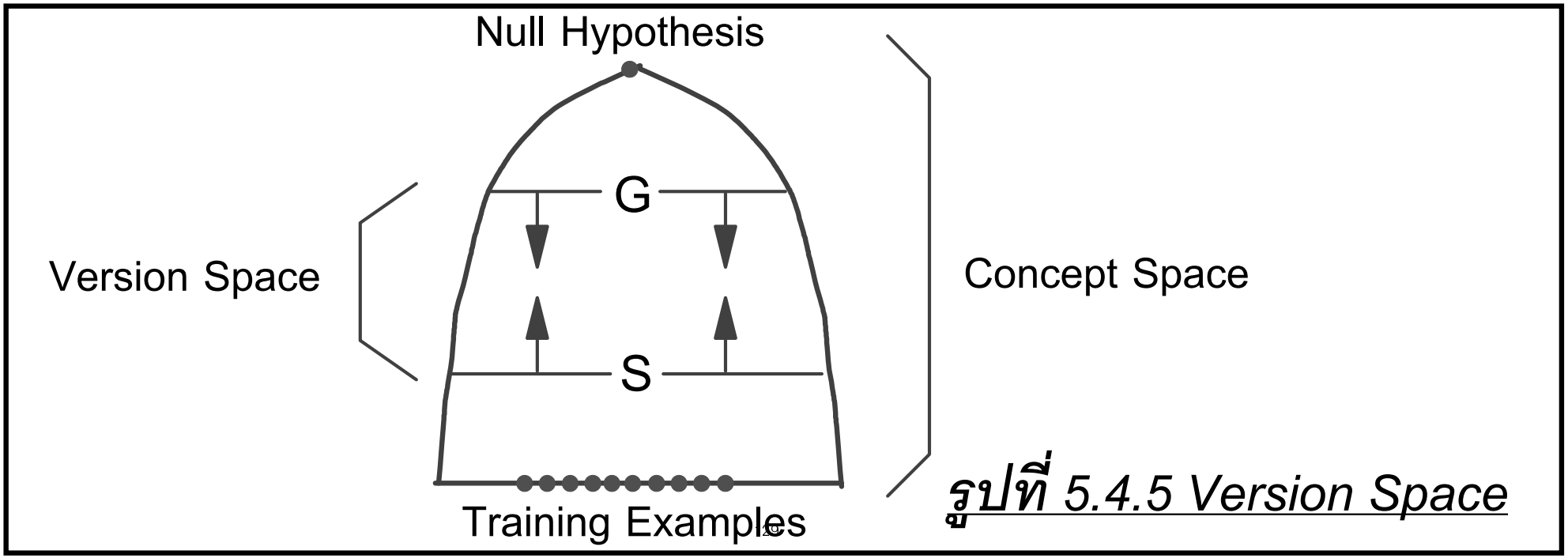
รูปที่ 5.4.3 Concept "Japanese Economy Car"

- ปัญหาของการเรียนรู้คือ " กำหนด ภาษาที่ใช้แสดงให้ (เช่นในรูปที่ 5.4.2) และตัวอย่างบวก ตัวอย่างลบให้ (เช่นในรูปที่ 5.4.1) จงหา concept description ที่สอดคล้องกับตัวอย่าง (อธิบายตัวอย่างบวก และ ไม่อธิบายตัวอย่างลบ) "
- Concept space คือ space ที่มีสมาชิกเป็น descriptions โดยที่สมาชิกเหล่านั้นมี partial ordering (ตัวที่ general กว่าจะอยู่เหนือตัวที่ specific กว่า) ดังรูปที่ 5.4.4
  - ตัวที่อยู่บนสุด คือ ตัวที่ general มากที่สุด
  - ตัวที่อยู่ล่างสุด คือ ตัวที่ specific มากที่สุด (ตัวอย่างบวก)
  - target concept description จะอยู่ระหว่างบนสุดกับล่างสุด
- ค้นหา target concept description โดยสร้าง hypothesis ที่เป็นเซ็ตย่อยของ concept space และเรียกเซ็ตย่อยนี้ว่า version space



รูปที่ 5.4.4 Concept Space

- Version space เป็น space ที่มีสมาชิกเป็น description ซึ่งสอดคล้องกับตัวอย่าง
- การแสดง version space แสดงโดยเซ็ตย่อย 2 เซ็ต คือ G และ S
  - เซ็ต G ประกอบด้วย most *general* descriptions ที่สอดคล้องกับตัวอย่างที่เคยพบมาทั้งหมด
  - เซ็ต S ประกอบด้วย most *specific* descriptions ที่สอดคล้องกับตัวอย่างที่เคยพบมาทั้งหมด
  - version space จะอยู่ระหว่าง G และ S (ดูรูปที่ 5.4.5)



รูปที่ 5.4.5 Version Space

- หลักการของ version space คือ ทุกครั้งที่เราได้รับตัวอย่างบวกตัวใหม่ เราจะทำให้ S general มากขึ้น และทุกครั้งที่ได้รับตัวอย่างลบ เราจะทำให้ G specific มากขึ้น จนในที่สุด S และ G ลู่เข้าสู่ค่าเดียวกัน ซึ่งเรียกว่า target concept description

<u>algorithm : Candidate Elimination</u>

1. G := {null description}
2. S := {first positive example}
3. Accept a new example E
   IF E is positive THEN
      Remove from G any descriptions that do not cover the example.
      Update S to contain the most specific set of descriptions in the
      version space that cover the example and the current elements of S.
   END ELSE IF E is negative THEN
      Remove from S any descriptions that cover the example.
      Update G to contain the most genereal set of desciptions in the
      version space that do not cover the example.
4. IF S and G are both singleton sets and S = G THEN Output the element
   ELSE IF S and G are both singleton sets and S<>G THEN examples were
            inconsistent.
   ELSE goto 3.

ตัวอย่างของการเรียน concept "Japanese economy car"

origin : Japan	origin : Japan	origin : Japan
mfr : Honda	mfr : Toyota	mfr : Toyota
color : Blue	color : Green	color : Blue
decade : 1980	decade : 1970	decade : 1990
type : Economy	type : Sports	type : Econamy
(+)	(-)	(+)

origin : USA	origin : Japan
mfr : Chrysler	mfr : Honda
color : Red	color : White
decade : 1980	decade : 1980
type : Economy	type : Economy
(-)	(+)

- จากตัวอย่างบวก 3 ตัว และ ตัวอย่างลบ 2 ตัวด้านบน เราเริ่มด้วยการสร้าง G และ S จากตัวอย่างแรก

  G = {(x1,x2,x3,x4,x5)}

  S = {(Japan,Honda,Blue,1980,Economy)}

  โดยที่ (x1,x2,x3,x4,x5) เป็นค่าของ slot ที่ 1, 2, 3, 4, 5 ตามลำดับ

- ตัวอย่างที่ 2 เป็นตัวอย่างลบ ดังนั้นเรา specialize G เพื่อไม่ให้ version space อธิบายตัวอย่างลบนี้ โดยการเปลี่ยนตัวแปรให้เป็นค่าคงที่

  G = {(x1,Honda,x3,x4,x5), (x1,x2,Blue,x4,x5),

  (x1,x2,x3,1980,x5), (x1,x2,x3,x4,Economy)}

  ส่วน S ไม่เปลี่ยนแปลง = {(Japan,Honda,Blue,1980,Economy)}

- ตัวอย่างที่ 3 เป็นบวก = (Japan,Toyota,Blue,1990,Economy)

  เรากำจัด description ใน G ที่ไม่สอดคล้องกับตัวอย่างนี้

  G = {(x1,x2,Blue,x4,x5), (x1,x2,x3,x4,Economy)}

  และ generalize S ให้รวมตัวอย่างนี้

  S = {(Japan,x2,Blue,x4,Economy)}

  ที่จุดนี้เราได้ version space ที่แสดง

  "Japanese blue economy car", "blue car" หรือ "Economy car"

- ตัวอย่างที่ 4 เป็นลบ = (USA,Chrysler,Red,1980,Economy)

  G = {(x1,x2,Blue,x4,x5), (x1,x2,Blue,x4,Economy),
  (Japan,x2,x3,x4,Economy)}

  S = {(Japan,x2,Blue,x4,Economy)}
- ตัวอย่างที่ 5 เป็นบวก = (Japan,Honda,White,1980,Economy)

  G = {(Japan,x2,x3,x4,Economy)}

  S = {(Japan,x2,x3,x4,Economy)}

ที่จุดนี้ ได้คำตอบ S=G แสดง "Japanese economy car"

<u>สิ่งสำคัญเกี่ยวกับ version space</u>

- algorithm เป็น least-commitment algorithm -- ในแต่ละขั้นตอน version space จะถูก pruned ให้เป็น space ที่เล็กลงน้อยที่สุดเท่าที่เป็นไปได้ ดังนั้น ถึงแม้ว่าตัวอย่างบวกทุกตัวเป็น Japanese cars ก็ตาม algorithm ก็จะไม่ตัดความน่าจะเป็นที่ concept อาจจะรวม car อื่นๆทิ้ง จนกระทั่งพบตัวอย่างลบ

- กระบวนการค้นหาเป็นแบบ exhaustive breadth-first search : เห็นได้จากการ update เซ็ต G ดังนั้นทำให้ algorithm มีประสิทธิ-ภาพต่ำในกรณีที่ space ใหญ่มาก ๆ ซึ่งอาจทำให้ดีขึ้นโดยใช้ heuristic เข้าช่วยในการค้นหา
- เซ็ต S ประกอบด้วยสมาชิกเพียงตัวเดียว เพราะว่า ตัวอย่างบวก 2 ตัวใด ๆมี generalization เพียงหนึ่งเดียว ดังนั้น version space จึงไม่สามารถเรียน disjunctive concept (concept ที่อยู่ในรูปของ or เช่น "Japanese econamy car or Japanese sport car"
- จุดอ่อนอีกอย่างของ version space คือ ไม่สามารถจัดการกับ noisy example (example ที่มีข้อมูลบางส่วนผิดพลาด) เช่น ถ้า example ตัวที่ 3 (Japan Toyota Blue 1990 Economy) เราให้ class ผิดเป็น (-) algorithm จะไม่สามารถเรียน concept "Japanese economy car" ได้ถูกต้อง

5.5 Identification Tree Learning

- Identification-tree learning เป็นวิธีการเรียนรู้ที่ใช้มากที่สุดใน machine learning
- ระบบเรียนรู้ประเภทนี้ เรียน identification tree จากตัวอย่างของหลายๆ class ซึ่ง tree ที่ได้ใช้ classify ตัวอย่าง
- ตัวอย่างของปัญหา : ต้องการหาว่าอะไรคือปัจจัยที่ทำให้คนที่ไปผึ่งแดดตามชายทะเล บางคนก็จะมีผิวเปลี่ยนเป็นสี tan แต่บางคนต้องได้รับความทรมานจากผิวไหม้ โดยข้อมูลที่สังเกตุได้มี ความแตกต่างของ สีผม ส่วนสูง น้ำหนัก ของผู้ที่ไปผึ่งแดด และบางคนก็ใช้โลชั่น บางคนก็ไม่ใช้

class

attribute →

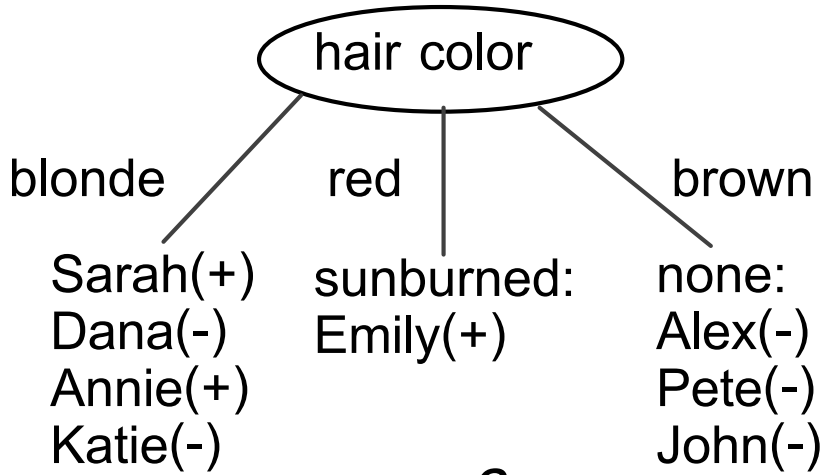
value

Name	Hair	Height	Weight	Lotion	Result
Sarah	blonde	average	light	no	sunburned
Dana	blonde	tall	average	yes	none
Alex	brown	short	average	yes	none
Annie	blonde	short	average	no	sunburned
Emily	red	average	heavy	no	sunburned
Pete	brown	tall	heavy	no	none
John	brown	average	heavy	no	none
Katie	blonde	short	light	yes	none

ตารางแสดงข้อมูลที่สังเกตุได้

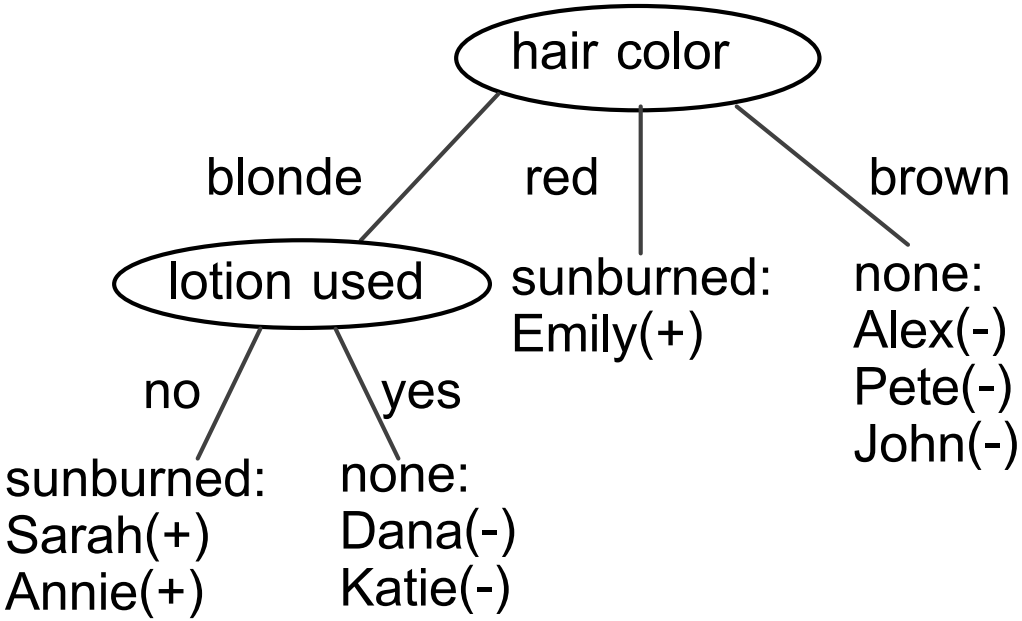
- ถ้าเราต้องการรู้ว่าใครบางคนถ้าไปผึ่งแดดแล้ว ผิวจะไหม้ (sunburned) หรือไม่ ก็อาจจะทำได้โดยเทียบข้อมูลของคนนั้นกับตารางด้านบน ซึ่งความน่าจะเป็นที่ข้อมูลจะอยู่ในตารางเป็น 8/(3*3*3*2) = 15% จะเห็นว่าวิธีนี้ไม่ใช่วิธีที่ดี

- identification tree คือ decision tree ซึ่ง
  - แต่ละ node ใน tree แสดง attribute
  - link ที่ต่อกับ node แสดง value
  - leaf ของ tree แสดง class
- การสร้าง tree ทำโดยสร้าง node ขึ้นทีละ node เพื่อ test คุณสมบัติของตัวอย่าง (ข้อมูล) แล้วแยกตัวอย่างตาม value ของตัวอย่าง ทำจนกระทั่งตัวอย่างในแต่ละ leaf เป็นตัวอย่างของ class เดียวกัน
- รูปที่ 5.5.1 แสดงการเลือก attribule hair color เป็น node แรก ที่จุดนี้ tree ที่สร้างขึ้นแยกตัวอย่างได้ในกรณีที่ hair color เป็น red (class เป็น sunburned) และ brown (class เป็น none) แต่ในกรณีที่ hair color เป็น blonde ยังแยกตัวอย่างไม่ได้ (มีตัวอย่างที่เป็นทั้ง sunburned และ none ปะปนกันอยู่)



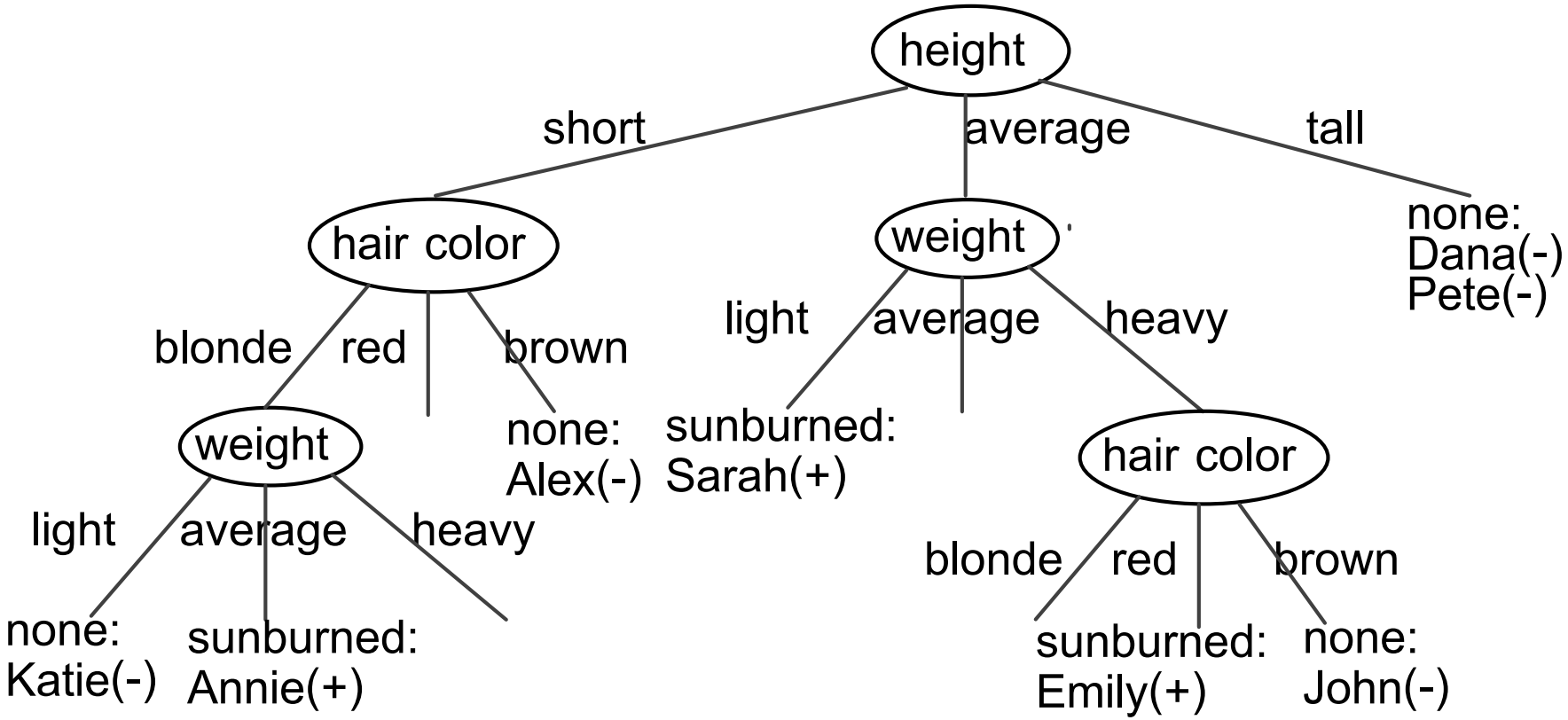
รูปที่ 5.5.1 สร้าง identification tree โดย node แรกเป็น hair color

- ในกรณีที่ hair color เป็น blonde เราสร้าง node ใหม่ตามรูป



รูปที่ 5.5.2 สร้าง node ที่ 2 เป็น lotion used

- identification tree ที่สอดคล้องกับตัวอย่าง อาจมีได้มากกว่า 1 เช่น เราอาจสร้าง tree ได้ดังรูป



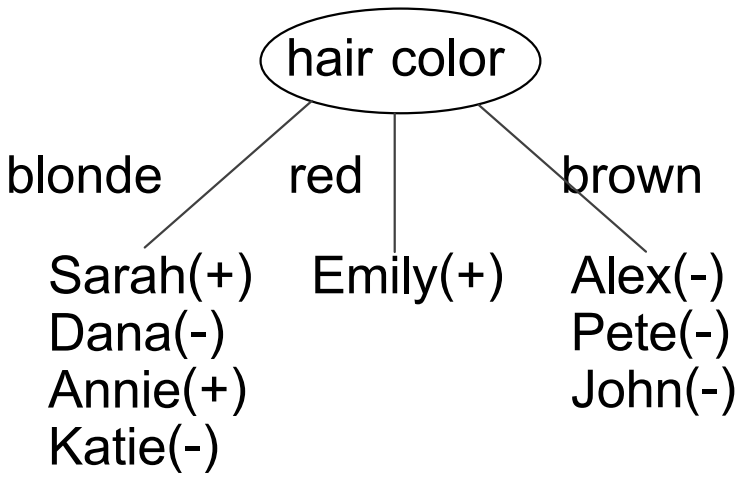
<u>รูปที่ 5.5.3 identification tree ที่แยกตัวอย่างได้</u>

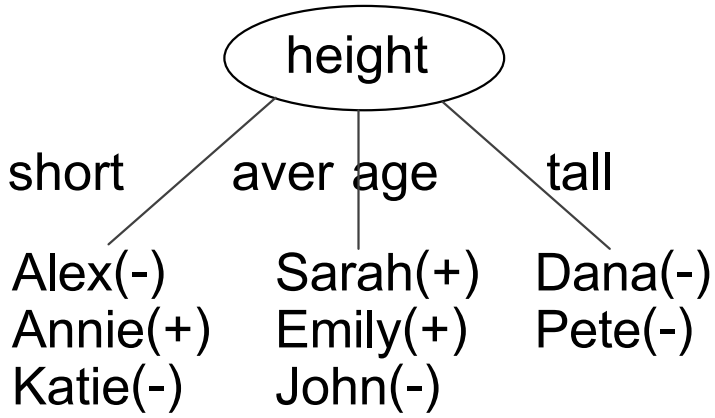
<u>แต่ไม่ถูกต้องตามความรู้สึกของคน</u>

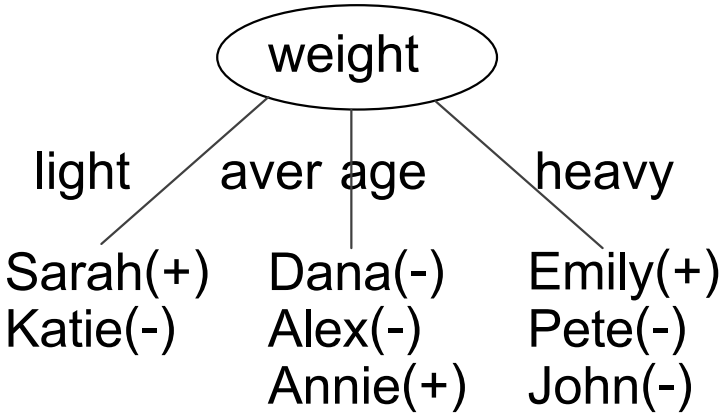
- ระหว่าง trees 2 ต้นในตัวอย่างที่แล้ว เราอยากได้ tree ต้นแรก
- Occam's razor ในการสร้าง identification tree
  - identification tree ที่มีขนาดเล็กที่สุดที่สอดคล้องกับตัวอย่าง เป็น tree ที่ดีที่สุด
- การหา tree ต้นที่เล็กที่สุดจะเสียค่าใช้จ่ายในการคำนวณมาก ดังนั้น จึงไม่สามารถกระทำได้ในทางปฏิบัติ

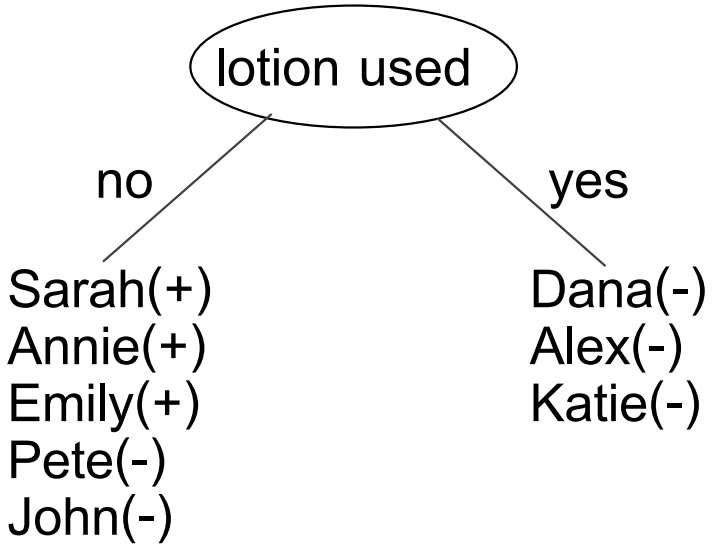
<u>การเลือกตัว test</u>

- หลักการสร้าง tree แบบหนึ่งคือพยายามเลือก test node ที่แยกตัวอย่างเป็นเซ็ตย่อย โดยที่ ทำให้สมาชิกส่วนใหญ่ในแต่ละเซ็ตย่อยเป็น class เดียวกันมากที่สุด
- ตัวอย่างเช่นในรูปที่ 5.5.4 แสดงผลของแต่ละ test node ในกรณีของ test node เป็น hair color สามารถแยกตัวอย่างเป็น 3 เซตย่อย เซ็ตย่อยแรก (blond) มีตัวอย่างของ 2 classes ปนกันอยู่ ส่วนเซ็ตย่อยที่ 2 (red) และ 3 (brown) มีตัวอย่างของ class sunburned และ none อยู่อย่างเดียวตามลำดับ



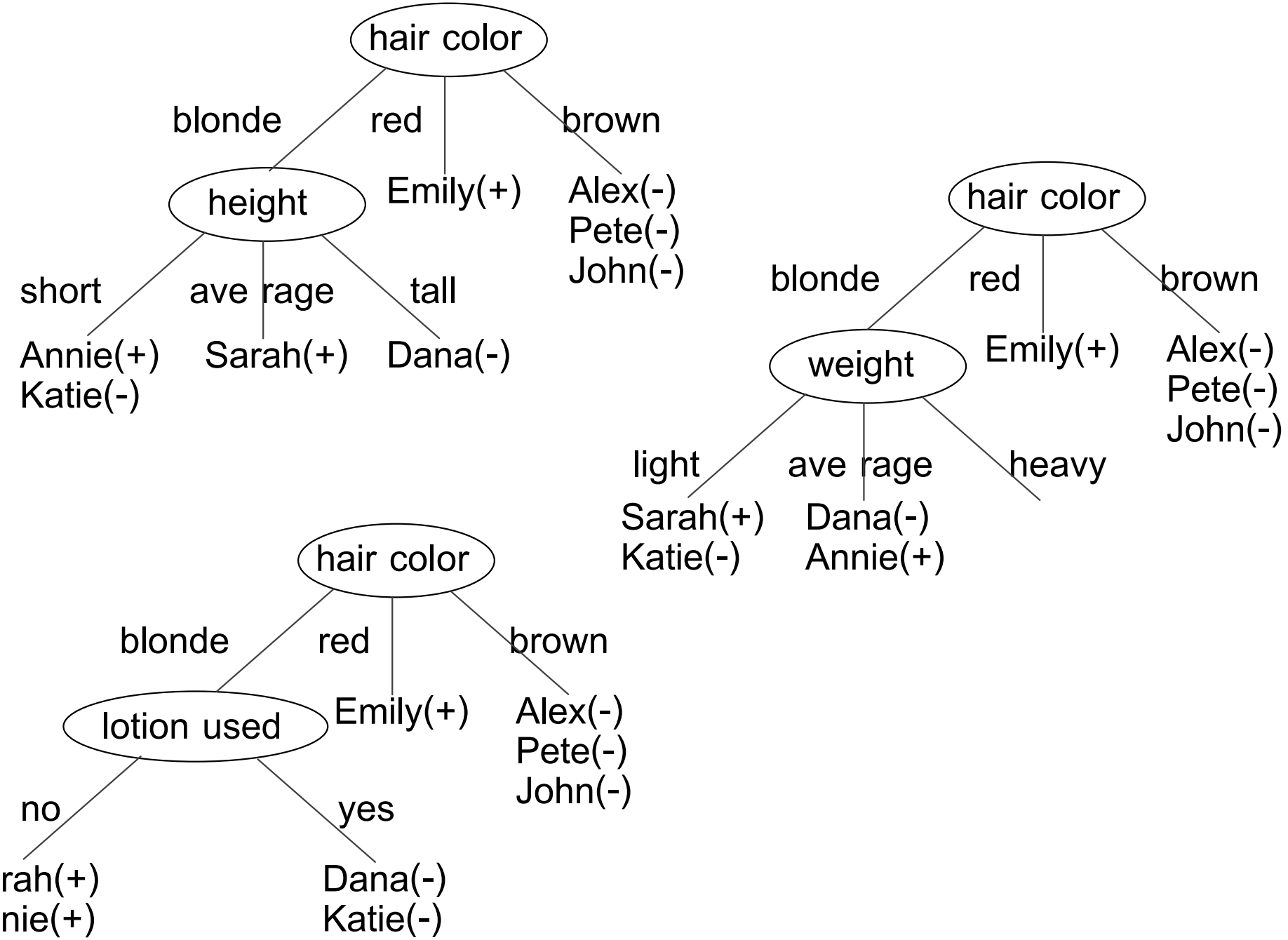






รูปที่ 5.5.4 ผลของแต่ละ test node :

hair color แยกตัวอย่างออกเป็น class ได้ดีที่สุด



รูปที่ 5.5.5 ผลของแต่ละ test node ที่เพิ่มเข้าใน subtree ของรูปที่ 5.5.4

- หลังจากนี้ เราแบ่งสมาชิกในเซ็ตย่อยที่ 1 ออกเป็น class ต่อไป
- รูปที่ 5.5.5 แสดงการสร้าง tree ต่อจากเดิม โดยให้ test node ที่เพิ่มเข้ามาเป็น height, weight และ lotion used ตามลำดับ
- ผลของ test node ที่เป็น lotion used สามารถแยกตัวอย่างออกเป็น class ได้อย่างสมบูรณ์ ที่จุดนี้เราหยุดการสร้าง tree

<u>วัดความสามารถในการแยกตัวอย่างได้อย่างไร</u>

- ID-3 เป็นโปรแกรมที่สร้าง identification tree โดยใช้ information theory เป็นตัววัดความสามารถในการแยกตัวอย่าง (gain) ของแต่ละ node

$$Gain(node) = \sum_b ((\frac{n_b}{n_t}) * (\sum_c -\frac{n_{bc}}{n_b} \log_2 \frac{n_{bc}}{n_b}))$$

โดยที่ n_b คือ จำนวนตัวอย่างของ link b

n_t คือ จำนวนตัวอย่างของทุก link รวมกัน

n_{bc} คือ จำนวนตัวอย่างของ class c ที่ link b

- เช่น node hair color ในรูปที่ 5.5.4 มีค่า gain เป็น

$$Gain(hair\ color) = \frac{4}{8}(-\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4}) + \frac{1}{8}*0 + \frac{3}{8}*0 = 0.5$$

ถ้าเราคำนวณค่า Gain ของทุก node จะได้ดังนี้

Gain(hair color) = 0.5 Gain(height) = 0.69

Gain(weight) = 0.94 Gain(lotion) = 0.61

ในกรณีนี้เราเลือก node hair color เพราะมีค่า Gain น้อยที่สุด (แยกตัวอย่างออกเป็น class ได้ดีที่สุด)

- เมื่อคำนวณ Gain ของทุก node ในรูปที่ 5.5.5 จะได้

Gain(height) = 0.5 Gain(weight) = 1 Gain(lotion) = 0

ซึ่งแสดงว่า node lotion ดีที่สุด

- Gain ได้รับการพิสูจน์ว่าเป็นฟังก์ชั่นที่ดีที่สุดฟังก์ชั่นหนึ่งโดยไม่ขึ้นกับชนิดของข้อมูล

การเปลี่ยนจาก tree เป็น rule

- เมื่อเราสร้าง tree เรียบร้อยแล้ว เราสามารถเปลี่ยน tree ให้อยู่ในรูปของ rule "IF THEN" ได้ โดยแสดงทุก path เริ่มต้นจาก root node ไปยัง leaf node ทุกครั้งที่พบ test node เพิ่ม test node กับค่าของ test ไว้ในส่วนของ IF และเมื่อพบ leaf node ให้ใส่ class ไว้ในส่วนของ THEN
- จาก tree ที่สร้างในรูป เราเปลี่ยนเป็น rules ได้ดังนี้

(1) IF the person's hair color is blonde
the person uses lotion
THEN nothing happens

(2) IF the person's hair color is blonde
the person uses no lotion
THEN the person turns red

(3) IF the person's hair color is red
THEN the person turns red

(4) IF the person's hair color is brown
THEN nothing happens

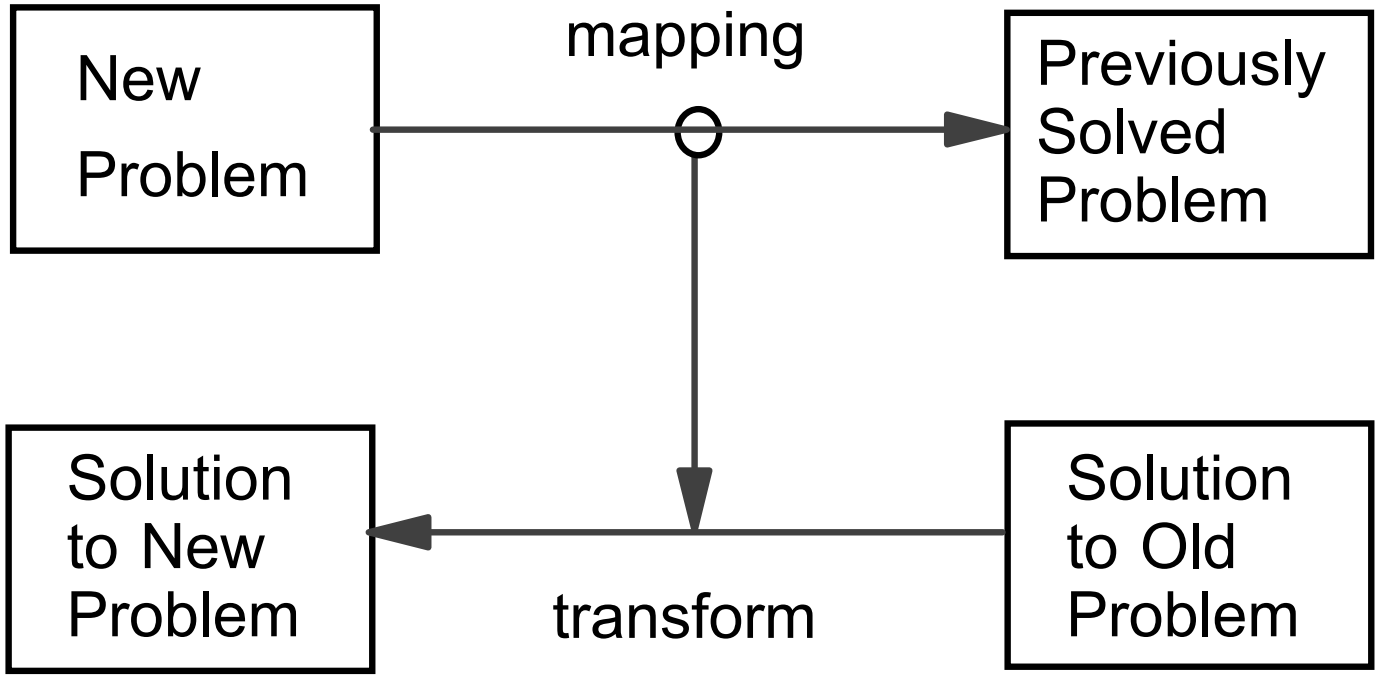
- ตารางด้านล่างแสดงการเปรียบเทียบการใช้ learning tools (ID-3) และไม่ใช้ ในการพัฒนา expert system

	Application	No. of Rules	Develop (Man Ys)	Maintain (Man Ys)	Learning Tools
MYCIN	Medical Diagnosis	400	100	N/A	N/A
XCON	VAX computer configuration	8,000	180	30	N/A
GASOIL	Hydrocarbon separtation system configuration	2,800	1	0.1	ExpertEase and Extran7
BMT	Configuration of fire-protection equipment in buildings	30,000	9	2.0	1st Class and Rulemaster

- GASOIL และ BMT เป็น expert systems ที่สร้างโดย learning tools ซึ่งพัฒนามาจากโปรแกรม ID-3

5.6 Learning by Analogy

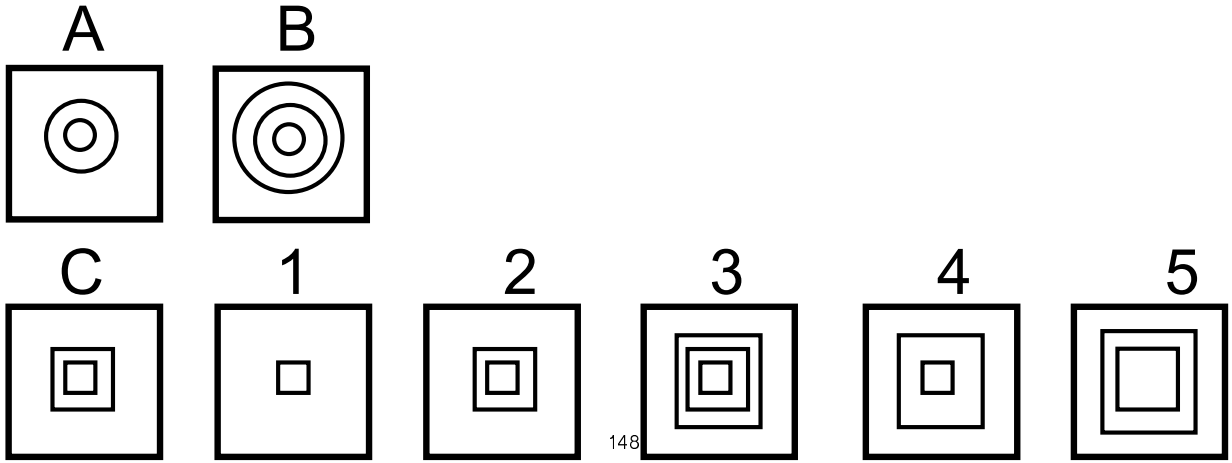
- Analogy เป็นวิธีการเรียนรู้อย่างหนึ่งที่เราใช้กัน
- เป็นเครื่องมือในการแก้ปัญหาใหม่ที่เราไม่เคยพบ โดยใช้ประสบการณ์ที่คล้ายกันช่วยในการแก้ปัญหา
- Learning by analogy เป็นเทคนิคในการแก้ปัญหาที่มีลักษณะคล้ายกับปัญหาเก่าที่เราเคยหาคำตอบได้แล้ว โดยแก้ไขคำตอบเดิมให้สอดคล้องกับปัญหาที่กำลังพบอยู่
  - match ปัญหาใหม่กับ เซ็ตของปัญหาเก่า เพื่อหาว่าอันไหนที่ใกล้เคียงกับปัญหาใหม่มากที่สุด
  - transform คำตอบของปัญหาเก่าที่เลือกได้ ให้สอดคล้องกับปัญหาใหม่ (ดูรูปที่ 5.6.1)



รูปที่ 5.6.1 Transformational Analogy

Analogy ในปัญหา IQ-test

จงเลือกรูป X (1, 2, 3, 4 หรือ 5) ที่ทำให้ ถ้า A เป็น B แล้ว C จะเป็น X



- ถ้าใช้หลักการในรูปที่ 5.6.1
  - มองว่า A คือ ปัญหาเก่า และ B คือ คำตอบของ A
  - C คือ ปัญหาใหม่
  - หาว่า กฎอะไรที่ใช้ในการเปลี่ยนจาก A เป็น B
  - map จาก A ไป C
  - ใช้กฎและผลการ map มาสร้างกฎที่จะใช้ในการเปลี่ยน C เป็น X
- โปรแกรม ANALOGY เป็นโปรแกรมที่จัดการกับปัญหานี้
- วิธีการของ ANALOGY คือ อธิบายให้ได้ว่า A เปลี่ยนเป็น B ได้อย่างไร หรือ กฎอะไรที่ใช้เปลี่ยน A เป็น B
- จากตัวเลือกที่ให้มา เราหากฎในการเปลี่ยนจาก C เป็นตัวเลือกนั้น match กฎที่ได้ในแต่ละตัวเลือกกับกฎที่ใช้เปลี่ยน A เป็น B แล้วเลือกกฎที่ match ได้ดีที่สุดเป็นคำตอบ

- ANALOGY ใช้กฎที่ประกอบด้วย 2 ส่วน
  - กฎอธิบาย relations ระหว่าง objects

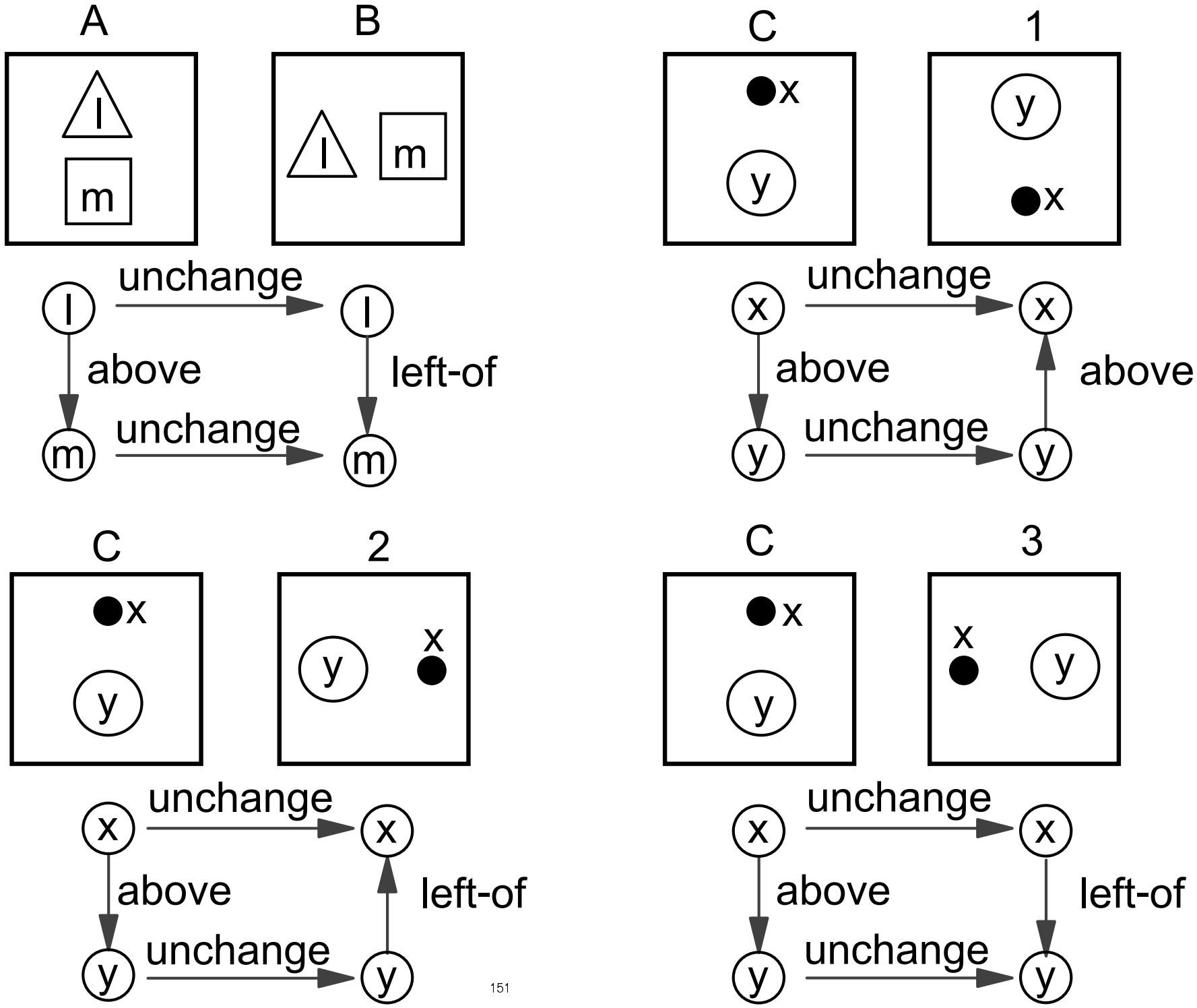
    above, left of, inside of
  - กฎอธิบาย transformations ระหว่าง objects

    be scaled, be rotated, be reflected, be deleted, be added
- ANALOGY ใช้ geometric analogy net (special case ของ semantic network)

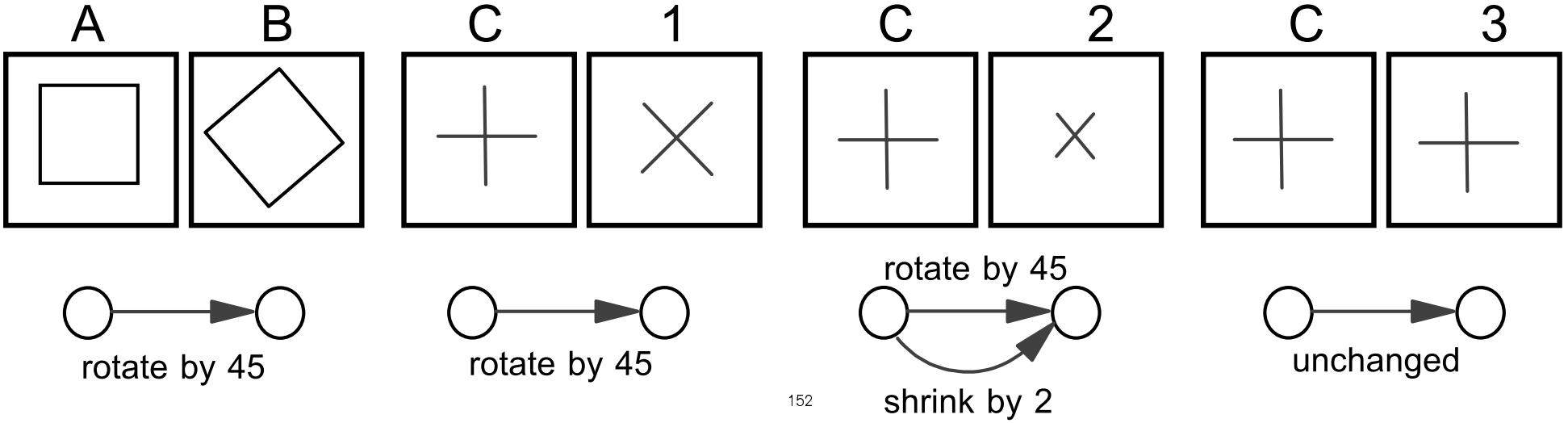
  geometric analogy net ประกอบด้วย
  - node แสดง object เช่น dot, circle, triangle, square, rectangle
  - link แสดง relation เช่น above, left of inside of
  - link แสดง transformation เช่น rotated, expanded, unchanged

รูปที่

5.6.2


A
B
l
m
l
m
unchange
l
l
above
left-of
m
unchange
m
C
1
x
y
y
x
unchange
x
x
above
above
unchange
y
y
C
2
x
y
y
x
unchange
x
x
above
left-of
unchange
y
y
C
3
x
y
x
y
unchange
x
x
above
left-of
unchange
y
y

- จากรูปที่ 5.6.2 ANALOGY จะลอง match objects ของ A-B กับ objects ของ C-X เช่น
  - match I กับ x, m กับ y
  - หรือ match I กับ y, m กับ x
- C-3 เป็นตัวเลือกที่ match กับ A-B ที่สุด โดย match I กับ x และ m กับ y ซึ่งทำให้ link ส่วนอื่นๆ match กันได้พอดี
- ในกรณีที่แล้วเราพิจารณาเฉพาะ relations ระหว่าง objects (ไม่มี transformation ระหว่าง objects)
  รูปด้านล่าง แสดงกรณีที่ objects ใน A เปลี่ยนเป็น B โดย rotation ซึ่งกรณีนี้ ANOLOGY จะเลือก C-1 เป็นคำตอบ



การประเมินค่าของการ match ที่ไม่พอดี

- ในกรณีที่ match ของ relations หรือ transformations ไม่พอดี
  ANALOGY วัดความคล้ายของกฎที่อธิบาย A-B กับ C-X แล้วเลือก X ที่ให้ความคล้ายของกฎที่ใกล้เคียงกับกฎของ A-B มากที่สุด
- ANALOGY ให้ weight ของ relation มากกว่าของ transformation
  เช่น ให้ weight ของ retation ใดๆเป็น 1 และของ transformation ใดๆเป็นค่าที่น้อยกว่า 1
  จากการทดลอง weight ที่ดีที่สุด คือ
  unchanged = 0.5, scaled = 0.4, rotated = 0.35, scaled and rotated = 0.3, reflected = 0.12, scaled and reflected = 0.075, rotated and reflected=0.05, scaled, rotated and reflected=0.025
- ANALOGY สามารถแก้ปัฐหาของ IQ-test ได้อย่างดี

5.7 Explanation-Based Learning (EBL)

- เป็นวิธีเรียนรู้ซึ่งเรียนจากตัวอย่างบวกเพียงตัวเดียว
- เช่น การเรียน concept "fork" ในการเล่น chess

  white knight attacks both the black king and black queen

  ในกรณีนี้ฝ่ายดำต้องยอมเสีย queen

  จากตัวอย่างเดียว สิ่งที่เรียนรู้คือ

  if any piece x attacks both the opponent's king and another piece y, then y will be lost.
- ใช้ domain-specific knowledge ช่วยในการเรียน
- กระบวนการของ EBL
  - ใช้ domain knowledge อธิบายว่าทำไมตัวอย่างจึงเป็นตัวอย่างของ concept ในรูปของกฎ
  - generalize กฎที่ได้ เพื่อให้ใช้กับกรณีอื่นได้

Input & output ของ EBL

Input:

- Training example -- ตัวอย่างของ concept ที่จะเรียน (board position ที่แสดง fork)
- Goal concept -- concept ที่จะเรียน (concept fork)
- Operational criterion -- description ที่สามารถนำไปใช้ได้ทันที (attack-both(WKn,BK,BQ) ไม่สามารถนำไปใช้ได้ทันที ต้องแสดงในรูปของตำแหน่งบนกระดาน เช่น

  position(WKn,f7), position(BK,h8), position(BQ,d8))
- Domain theory -- กฎต่างๆที่ใช้แสดงความสัมพันธ์ของ objects และ actions ใน domain นั้น (กฎการเล่น chess)

Output

- generalization ของ training example ซึ่งเพียงพอสำหรับอธิบาย goal concept และสอดคล้องกับ operational criterion

การเรียน concept "cup"

- Training example:

  owner(object23,ralph), has-part(object23,concavity12),
  isa(concavity12,concavity), is(concavity12,upward-pointing),
  has-part(object23,handle16), isa(handle16,handle),
  is(object23,light), color(object23,brown), has-part(object23,bottom19),
  is(bottom19,bottom), is(bottom19,flat), . . .

- Domain theory

  liftable(X), stable(X), open-vessel(X) $\rightarrow$ *cup(X)*
  is(X,light), has-part(X,Y), isa(Y,handle) $\rightarrow$ *liftable(X)*
  small(X), made-from(X,Y), low-density(Y) $\rightarrow$ *liftable(X)*
  has-part(X,Y), isa(Y,bottom), is(Y,flat) $\rightarrow$ *stable(X)*
  has-part(X,Y), isa(Y,concavity), is(Y,upward-pointing)
  $\rightarrow$ *open-vessel(X)*

- Goal concept : *cup(X)*

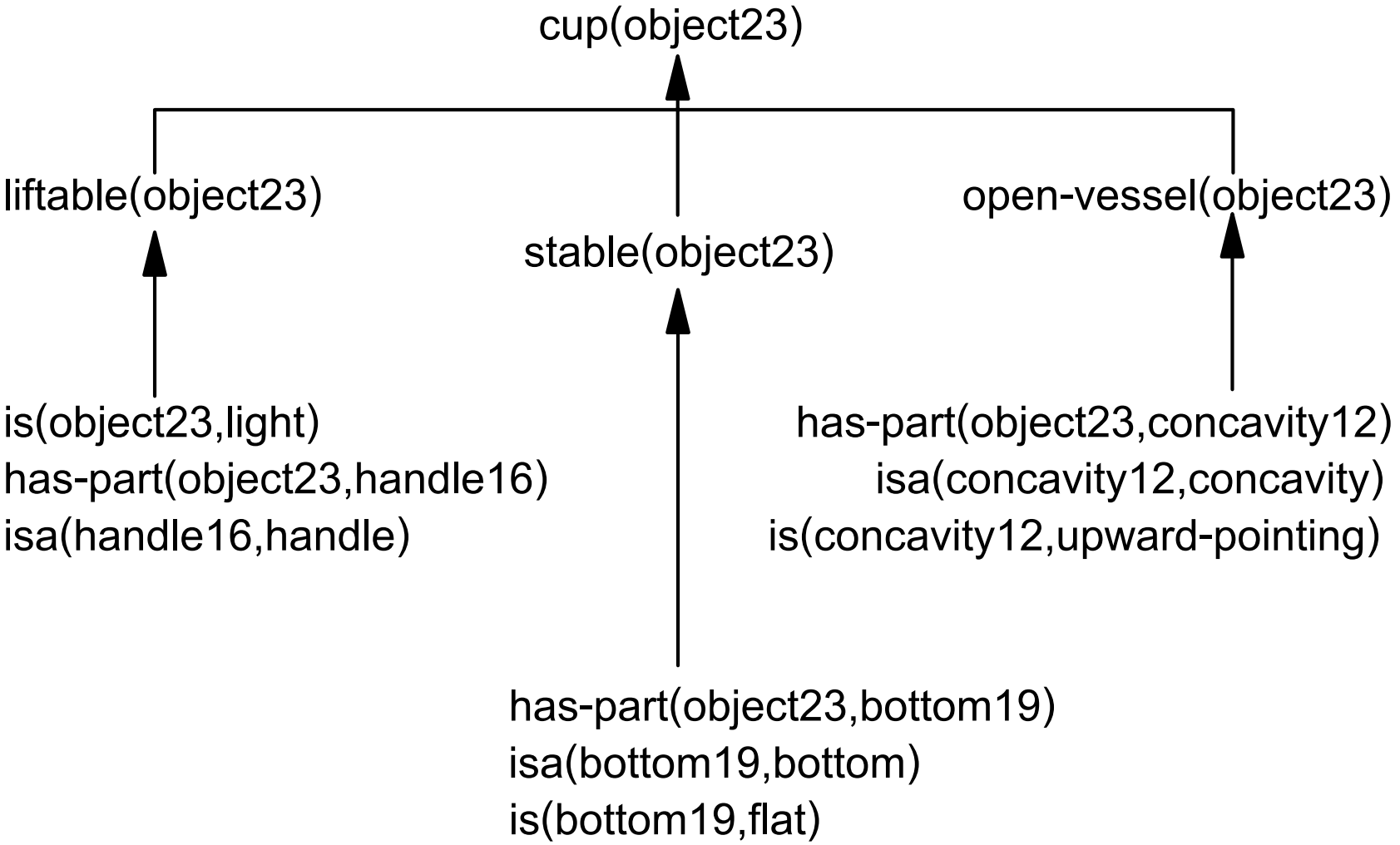
  X is a cup iff X is liftable, stable and open-vessel.

- Operational criterion : ***สิ่งที่แสดงลักษณะต่าง ๆของ*** *cup* ***เช่น*** *isa, has-part, color*

- จากตัวอย่างบวกที่ให้มา เราต้องการสร้าง description ของ cup

  (1) ใช้ domain knowledge อธิบายว่าทำไม object23 จึงเป็น cup

  สร้าง proof tree ของ object23



สังเกตุว่า predicate ที่ไม่เกี่ยวข้องกับ concept เช่น owner, color จะไม่มีใน proof tree

(2) generalize และ ดึง predicate ที่เป็น operational criterion มาสร้างกฎ

- generalization ตาม domain knowledge

  ถ้า argument ของ predicate ที่ตรงกันใน domain knowledge เป็น variable ก็เปลี่ยน argument ที่เป็น constant ให้เป็น variable

  ถ้า argument ของ predicate ที่ตรงกันใน domain knowledge เป็น constant ก็ไม่ต้องเปลี่ยน

เช่น is(object23,light) เปลี่ยนเป็น is(X,light)

โดยที่ X แทน object23

- ดึง predicate ที่เป็น operational criterion มาสร้างกฎได้

  is(X,light),has-part(X,H), isa(H,handle), has-part(X,B),

  isa(B,bottom), is(B,flat), has-part(X,C), isa(C,concavity),

  is(C,upward-pointing) $\rightarrow$ cup(X)

5.8 Learning of Logic Programs

- Learning of logic programs or Inductive Logic Programming (ILP) is a learning that uses logic as knowledge representation
- advantage of ILP
  - powerful representation :

    first-order or higher-order logic programs
  - can use background knowledge
- Quinlan(1990) described the learning system FOIL which employed *information-based heuristic* to guide the search for programs efficiently

Input & Output

Positive examples

sort([1,0],[0,1])

sort([1,0,2],[0,1,2])

. . .

Negative exampls

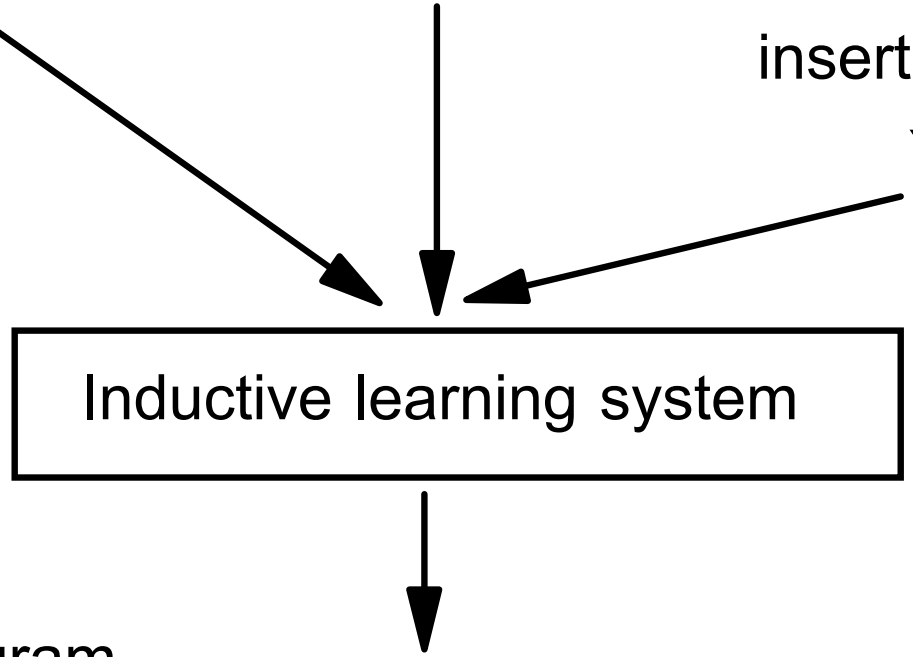
sort([1,0],[1,0])

sort([1,0,2],[0])

. . .

Background knowledge

```
insert(X,[ ],[X]).
insert(X,[Y|Ys],[Y|Zs]) :-
        X > Y, insert(X,Ys,Zs).
insert(X,[Y|Ys],[X,Y|Ys]) :-
        Y >= X.
```



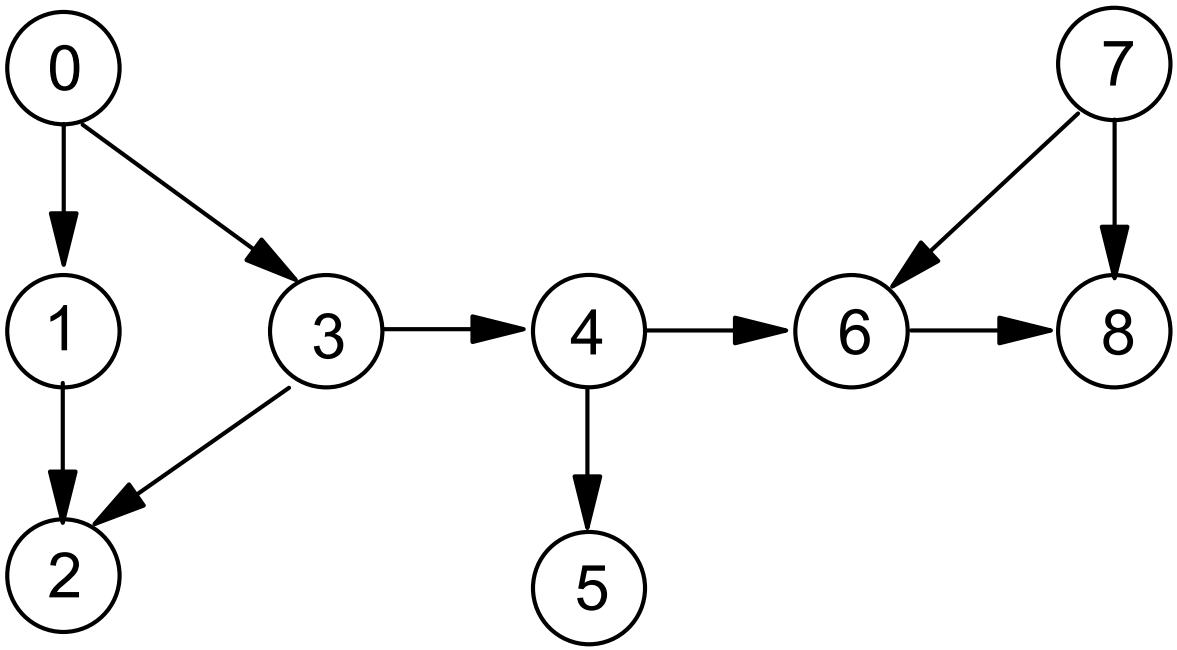
Program

```
sort([ ],[ ]).
sort([X|Xs],Ys) :- sort(Xs,Zs), insert(X,Zs,Ys).
```

FOIL(Information-Based Heuristic)

- FOIL learns funciton-free Horn clauses from examples in batch mode
- The system is based on ideas in the attribute-value learning system, ID-3
- The system employs information-based heuristic to guide the search efficiently
- It uses tuples to represent the examples

An Example of '*can-reach*'



B.K. *linked-to(X,Y)* : <0,1>, <0,3>, <1,2>, <3,2>, <3,4>,
<4,5>, <4,6>, <6,8>, <7,6>, <7,8>

Program *can-reach(X,Y) :- linked-to(X,Y).*

can-reach(X,Y) :- linked-to(X,Z), can-reach(Z,Y).

An Example of '*can-reach*'

can-reach(X1,X2) :-

positive tuples, T_1^+: (19 tuples)

<0,1> <0,2> <0,3> <0,4> <0,5> <0,6> <0,8> <1,2> <3,2> <3,4>

<3,5> <3,6> <3,8> <4,5> <4,6> <4,8> <6,8> <7,6> <7,8>

negative tuples, T_1^-: (62 tuples)

<0,0> <0,7> <1,0> <1,1> <1,3> <1,4> <1,5> <1,6> <1,7> <1,8>

<2,0> <2,1> <2,2> <2,3> <2,4> <2,5> <2,6> <2,7> <2,8> <3,0>

<3,1> <3,3> <3,7> <4,0> <4,1> <4,2> <4,3> <4,4> <4,7> <5,0>

<5,1> <5,2> <5,3> <5,4> <5,5> <5,6> <5,7> <5,8> <6,0> <6,1>

<6,2> <6,3> <6,4> <6,5> <6,6> <6,7> <7,0> <7,1> <7,2> <7,3>

<7,4> <7,5> <7,7> <8,0> <8,1> <8,2> <8,3> <8,4> <8,5> <8,6>

<8,7> <8,8>

The Algorithm of FOIL

Outermost level:

- Establish the training set consisting of positive and negative tuples
- UNTIL there is no positive tuple left DO
  - Find a clause the characterizes part of the target relation
  - Remove all examples that satisfy the right-hand side of this clause from the traning set

The Algortihm of FOIL

Inner loop: find a clause of the form

$p(X1,X2, \ldots ,Xk) :- L1, L2, \ldots , Ln$

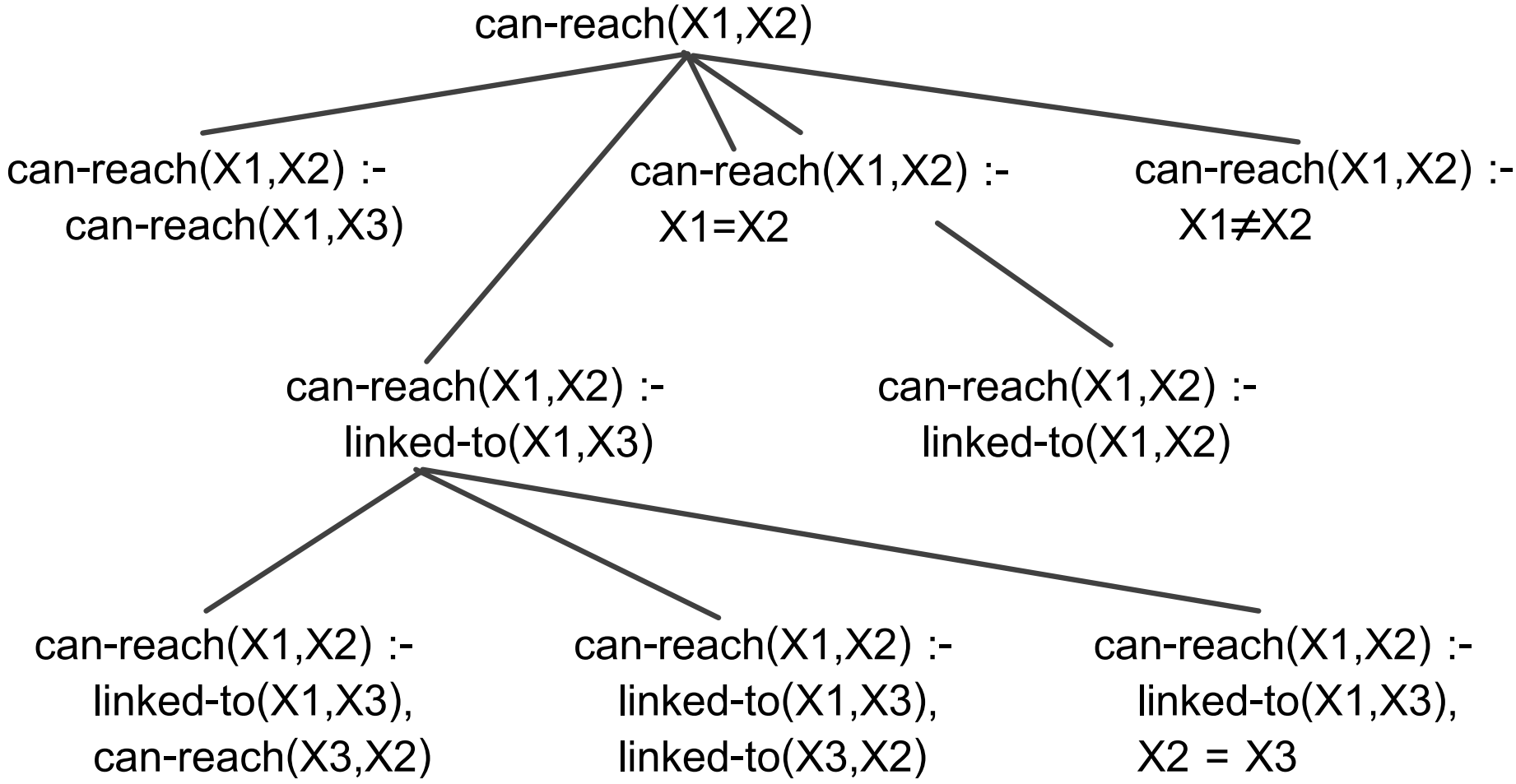
where, Li is $Xj = Xk$, $Xj \neq Xk$, $q(V1, \ldots ,Vn)$ or $\neg q(V1, \ldots ,Vn)$

Xj, Xk are existing variables, q is a background predicate,

Vi is an existing or new variable

- Initialize the local traning set T_i to the traning set and let $i = 1$
- WHILE T_i contains negative tuples:
  - Find a background literal L_i to add to the right-hand side of the clause
  - Produce a new traning set T_{i+1} based on those tuples in T_i that satisfy L_i.
  - Increment i and continue

Hypothesis Space of 'can-reach'



An Example of '*can-reach*'

- Suppose that the first literal selected for the right-hand side is *linked-to(X1,X2)*. The obtained clause is:

  can-reach(X1,X2) :- linked-to(X1,X2)

  The clause covers no negative example.

- In the second outermost loop, T_1 consisting of the remaining positive tuples

  <0,2> <0,4> <0,5> <0,6> <0,8> <3,5> <3,6> <3,8> <4,8>

An Example of '*can-reach*'

- If the literal *linked-to(X1,X3)* is selected, then T2 consists of the triples:

positive triples, T_2^+ : (18)

<0,2,1> <0,2,3> <0,4,1> <0,4,3> <0,5,1> <0,5,3> <0,6,1> <0,6,3> <0,8,1>

<0,8,3> <3,5,2> <3,5,4> <3,6,2> <3,6,4> <3,8,2> <3,8,4> <4,8,5> <4,8,6>

negative triples, T_2^- : (54)

<0,0,1> <0,0,3> <0,7,1> <0,7,3> <1,0,2> <1,1,2> <1,3,2> <1,4,2> <1,5,2>

<1,6,2> <1,72> <1,8,2> <3,0,2> <3,0,4> <3,1,2> <3,1,4> <3,3,2> <3,3,4>

. . . <7,5,8> <7,7,6> <7,7,8>

An Example of '*can-reach*'

- If the second literal selected is *canreach(X3,X2)*, then T_3 consists of only positive triples (10):

  <0,2,1> <0,2,3> <0,4,3> <0,5,3> <0,6,3> <0,8,3> <3,5,4>

  <3,6,4> <3,8,4> <4,8,6>

  can-reach(X1,X2) :- linked-to(X1,X3), can-reach(X3,X2)

- If ,instead of the second literal above, *linked-to(X3,X2)* was selected, then T_3 would consist of only 6 positive triples:

  <0,2,1> <0,2,3> <0,4,3> <3,5,4> <3,6,4> <4,8,6>

  can-reach(X1,X2) :- linked-to(X1,X3), linked-to(X3,X2)

Information-Based Heuristic(Gain)

- The whole purpose of a clause is to characterize a subset of positive tuples in a relation
- It seems appropriate to focus on the information provided by signalling that a tuple is one of the positive kind
- If the current T_i contains T_i^+ positive tuples and T_i^- negative tuples, the information required for this signal from T_i is given by

$$I(T_i) = -\log_2(T_i^+ / (T_i^+ + T_i^-))$$

- If the selection of a literal L_i would give rise to a new set T_{i+1}, the information given by the same signal is similarly

$$I(T_{i+1}) = -\log_2(T_{i+1}^+ / (T_{i+1}^+ + T_{i+1}^-))$$

Information-Based Heuristic(Gain)

- Suppose that T_i^{++} of the positive tuples in T_i are represented by one or more tuples in T_{i+1}. The total information regarding the positive tuples in T_i is :

$$Gain(L_i) = T_i^{++} \times (I(T_i) - I(T_{i+1}))$$

- Gain is large if the positive tuples are more concentrated in T_{i+1} than in T_i

An Example of '*can-reach*'

- Gain(can-reach(X3,X2)) is

$$10 \times (-\log_2(18/(18+54)) + \log_2(10/(10+0))) = 20.0$$

- Gain(linked-to(X3,X2)) is

$$6 \times (-\log_2(18/(18+54)) + \log_2(6/(6+0))) = 12.0$$

- That is the clause

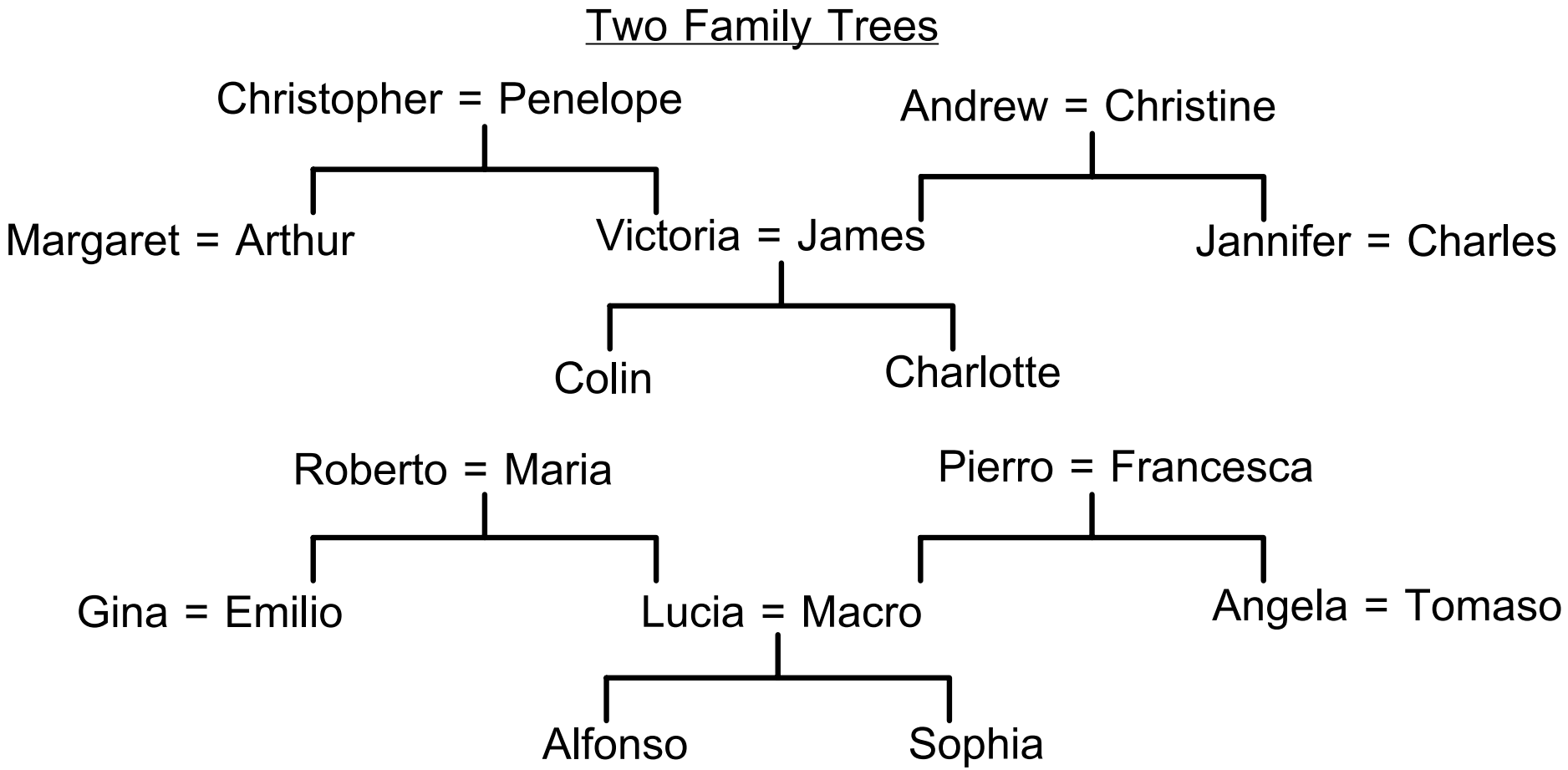
can-reach(X1,X2) :- linked-to(X1,X3), can-reach(X3,X2)

is prefered to

can-reach(X1,X2) :- linked-to(X1,X3), linked-to(X3,X2)

Results

- Learning Family Relationships



A = B หมายถึง A แต่งงานกับ B

Background knowledge :

wife(X,Y) husband(X,Y)

mother(X,Y) father(X,Y)

sister(X,Y) brother(X,Y)

aunt(X,Y) uncle(X,Y)

daughter(X,Y) son(X,Y)

niece(X,Y) nephew(X,Y)

Learned concept:

wife(X,Y) :- father(Y,Z), son(Z,X).

wife(X,Y) :- father(Y,Z), daughter(Z,X).

mother(X,Y) :- father(Z,Y), husband(Z,X).

sister(X,Y) :- daughter(X,Z), father(Z,Y).

- Learning Recursive Relations on Lists
  - List(X) -- X is a list

    Background knowledge:

    list(X) : list([ ]), list([a]), list([b,[a],d]), list([[a],d]), list([d]) . . .

    null(X) : null([ ])

    components(X,Y,Z) :- components([a],a,[ ])
    components([b,[a],d],b,[[a],d])
    components([[a],d],[a],[d]) . . .

    Learned concept:

    list(A) :- components(A,B,C), list(C).

    list(A) :- null(A).
  - Append(X,Y,Z) -- appending X to Y gives Z

    Background knowledge:

    null(X), components(A,B,C)

    Learned concept:

    append(A,B,C) :- A=C, null(B).

    append(A,B,C) :- B=C, null(A).

    append(A,B,C) :- components(C,D,B), components(A,D,E), null(E).

    append(A,B,C) :- components(C,D,E), components(A,D,F), append(F,B,E).

- Learning the Concept of an Arch

  Background knowledge:

  arch(A,B,C) -- A, B and C form an arch with lintel A

  supports(A,B) -- A supports B

  left-of(A,B) -- A is left of B

  touches(A,B) -- the sides of A and B touch

  brick(A) -- A is a brick

  wedge(A) -- A is a wedge

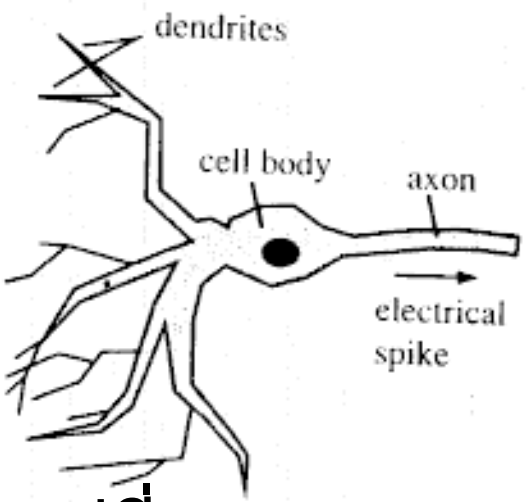
  parallelpiped(A,B) -- A is a brick or a wedge

  Learned concept:

  arch(A,B,C) :- left-of(B,C), supports(B,A), ¬touch(B,C)

5.9 Artificial Neural Networks

- Artificial neural networks (ANN) เป็นการจำลองการทำงานบางส่วนของสมองมนุษย์
- Neuron เป็นเซลที่ประกอบด้วย nucleus, cell body, dendry, synapse axon ดังแสดงในรูปที่ 5.9.1



รูปที่ 5.9.1 neuron

- สมองของมนุษย์มี neuron ประมาณ 10^{11} แต่ละ neuron เชื่อมต่อกับ neuron อื่นประมาณ 10^{4} และมี stwiching time ประมาณ 10^{-3} วินาที ซึ่งช้ามากเมื่อเทียบกับ่คอมพิวเตอร์ (10^{-10} วินาที) แต่ทำงานบางอย่างได้ดีกว่ามาก

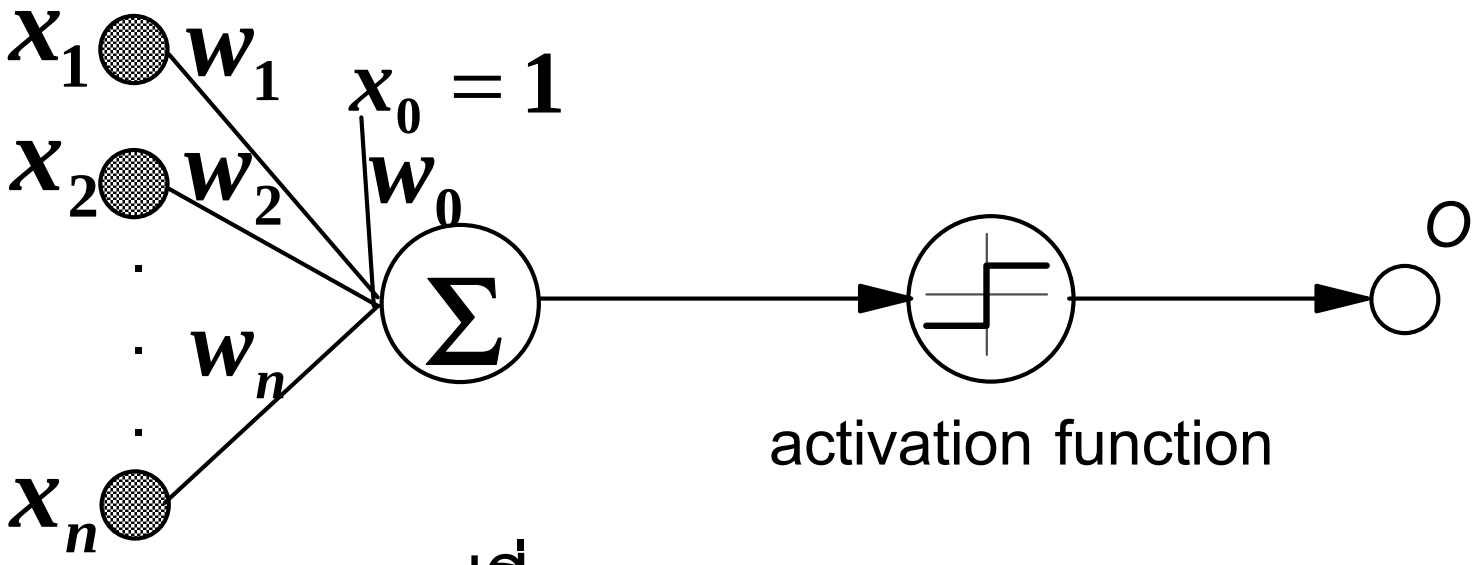
Perceptrons

- Perceptron เป็น ANN ชนิดหนึ่ง ซึ่งประกอบด้วยยูนิตเดี่ยว(รูป 5.9.2)
- input เป็น real-valued vector
- Perceptron คำนวณ linear combination ของ input(x_i) และให้ output(o) เป็น 1 ถ้าค่าที่ได้เกิน threshold และเป็น -1 ถ้าค่าไม่เกิน

$$o(x_1,\ldots,x_n)=\begin{cases}1 & if\ w_0+w_1x_1+\cdots+w_nx_n>0\\ -1 & if\ w_0+w_1x_1+\cdots+w_nx_n<0\end{cases}$$

โดยที่ $-w_0$ คือ ค่า threshold, w_i เป็น real-valued constant หรือ weight

- w_i เป็นตัวกำหนดความสำคัญของ input x_i ที่มีต่อ output



รูปที่ 5.9.2 Perceptron

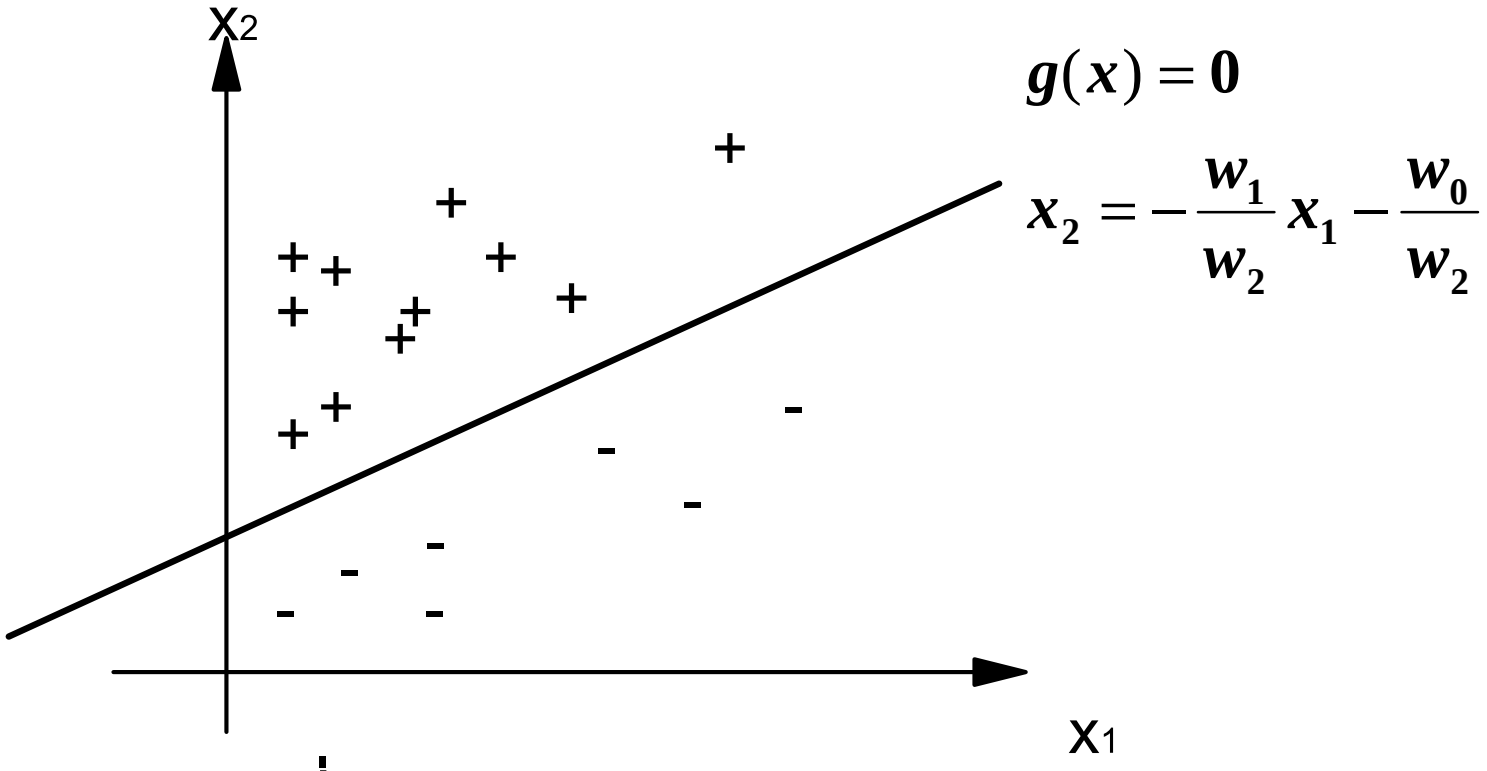
- ให้ $g(x)=\sum_{i=0}^{n} w_i x_i$ หรือในรูปของเวกเตอร์ $g(x)=\vec{w}\cdot\vec{x}$

$$o(x_1,...,x_n)=\begin{cases}1 & if\ g(x)>0\\ -1 & if\ g(x)<0\end{cases}$$

o ใช้ activation function เป็น stepwise bipolar (output = 1 หรือ -1)

- ในกรณีของ 2 inputs ดังแสดงในรูป 5.9.3

$$g(x)=w_0+w_1x_1+w_2x_2$$



รูปที่ 5.9.3 decision surface

- เราสามารถมอง perceptron เป็น hyperplane decision surface (g(x)=0)ใน n-dimensional surface
- เราสามารถสอน perceptron ให้แยกตัวอย่าง (คู่ลำดับ input-output)
- คุณสมบัติหนึ่งของ perceptronคือ ทุก function ที่ perceptron สามารถคำนวณได้ มันก็สามารถที่จะถูกสอนให้เรียน function นั้นได้

Perceptron Learning Rule

- จงหาค่าของเวกเตอร์น้ำหนัก $\vec{w}$ ที่ทำให้ perceptron เอ๊าท์พุตเป็น +1 หรือ -1 ได้ถูกต้องสำหรับทุกตัวอย่างที่สอน
- Perceptron learning rule
  - เริ่มต้นจากการสุ่มค่าของน้ำหนัก w_i
  - เทียบ perceptron กับทุกตัวอย่างที่สอนทีละตัว และแก้ไขน้ำหนักเมื่อ perceptron แยกตัวอย่างผิดพลาด
  - วนทำซ้ำกับตัวอย่างที่สอน จนกระทั่ง perceptron แยกตัวอย่างได้ถูกต้องทั้งหมด
  - น้ำหนักถูกปรับตาม $w_i = w_i + \Delta w_i$

    โดยที่ $\Delta w_i = \alpha\ (t - o) x_i$ = α x error x input

    t เป็น target output,

    o เป็น output จาก perceptron, α เป็นค่าคงที่แสดง learning rate

- ในกรณีที่ perceptron แยกตัวอย่างได้ถูกต้อง

  (t-o) จะมีค่าเป็น 0 Δw ไม่เปลี่ยนแปลง

- ในกรณีที่ perceptron ให้เอ๊าท์พุตเป็น -1 แต่ target output = 1

  เพื่อที่จะทำให้ perceptron เอ๊าท์พุตเป็น 1 น้ำหนักต้องถูกปรับ

  ให้สามารถเพิ่มค่าของ $\vec{w} \cdot \vec{x}$

  -- ถ้า $x_i>0$, w_i จะเพิ่มขึ้นและจะทำให้ perceptron เอ๊าท์พุตได้

     ถูกต้องยิ่งขึ้น ($\Delta w_i = \alpha(t-o)x_i > 0$)

  -- ถ้า $x_i<0$, w_i จะลดลงและจะทำให้ perceptron เอ๊าท์พุตได้

     ถูกต้องยิ่งขึ้น

- ในกรณีที่ perceptron ให้เอ๊าท์พุตเป็น 1 แต่ target output = -1

  w_i ของ x_i ที่เป็นค่าบวก จะลดลง,

  w_i ของ x_i ที่เป็นค่าลบ จะเพิ่มขึ้น

ตารางที่ 5.9.1 การเรียนรู้ของ Perceptron กับฟังก์ชัน AND

(binary activation function (output = 0 หรือ 1))

Perceptron Learning Example											
		Bias Input X0 = ±1					Alpha = 0.05				
Input	Input				Net Sum	Target	Actual	Alpha	Weight Values		
X1	X2	1.0*W0	X1*W1	X2*W2	Input	Out	Out	*Error	W0	W1	W2
									0.1	0.1	0.1
0	0	0.10	0.00	0.00	0.10	0	1	−0.05	0.05	0.10	0.10
0	1	0.05	0.00	0.10	0.15	0	1	−0.50	−0.45	0.10	−0.40
1	0	−0.45	0.10	0.00	−0.35	0	0	0.00	−0.45	0.10	−0.40
1	1	−0.45	0.10	−0.40	−0.75	1	0	0.50	0.05	0.60	0.10
0	0	0.05	0.00	0.00	0.05	0	1	−0.50	−0.45	0.60	0.10
0	1	−0.45	0.00	0.10	−0.35	0	0	0.00	−0.45	0.60	0.10
1	0	−0.45	0.60	0.00	0.15	0	1	−0.50	−0.95	0.10	0.10
1	1	−0.95	0.10	0.10	−0.75	1	0	0.50	−0.45	0.60	0.60
0	0	−0.45	0.00	0.00	−0.45	0	0	0.00	−0.45	0.60	0.60
0	1	−0.45	0.00	0.60	0.15	0	1	−0.50	−0.95	0.60	0.10
1	0	−0.95	0.60	0.00	−0.35	0	0	0.00	−0.95	0.60	0.10
1	1	−0.95	0.60	0.10	−0.25	1	0	0.50	−0.45	1.10	0.60
0	0	−0.45	0.00	0.00	−0.45	0	0	0.00	−0.45	1.10	0.60
0	1	−0.45	0.00	0.60	0.15	0	1	−0.50	−0.95	1.10	0.10
1	0	−0.95	1.10	0.00	0.15	0	1	−0.50	−1.45	0.60	0.10
1	1	−1.45	0.60	0.10	−0.75	1	0	0.50	−0.95	1.10	0.60
0	0	−0.95	0.00	0.00	−0.95	0	0	0.00	−0.95	1.10	0.60
0	1	−0.95	0.00	0.60	−0.35	0	0	0.00	−0.95	1.10	0.60
1	0	−0.95	1.10	0.00	0.15	0	1	−0.50	−1.45	0.60	0.60
1	1	−1.45	0.60	0.60	−0.25	1	0	0.50	−0.95	1.10	1.10
0	0	−0.95	0.00	0.00	−0.95	0	0	0.00	−0.95	1.10	1.10
0	1	−0.95	0.00	1.10	0.15	0	1	−0.50	−1.45	1.10	0.60
1	0	−1.45	1.10	0.00	−0.35	0	0	0.00	−1.45	1.10	0.60
1	1	−1.45	1.10	0.60	0.25	1	1	0.00	−1.45	1.10	0.60
0	0	−1.45	0.00	0.00	−1.45	0	0	0.00	−1.45	1.10	0.60
0	1	−1.45	0.00	0.60	−0.85	0	0	0.00	−1.45	1.10	0.60
1	0	−1.45	1.10	0.00	−0.35	0	0	0.00	−1.45	1.10	0.60
1	1	−1.45	1.10	0.60	0.25	1	1	0.00	−1.45	1.10	0.60
0	0	−1.45	0.00	0.00	−1.45	0	0	0.00	−1.45	1.10	0.60
0	1	−1.45	0.00	0.60	−0.84	0	0	0.00	−1.45	1.10	0.60
1	0	−1.45	1.10	0.00	−0.35	0	0	0.00	−1.45	1.10	0.60
1	1	.45	1.10	0.60	0.25	1	1	0.00	−1.45	1.10	0.60

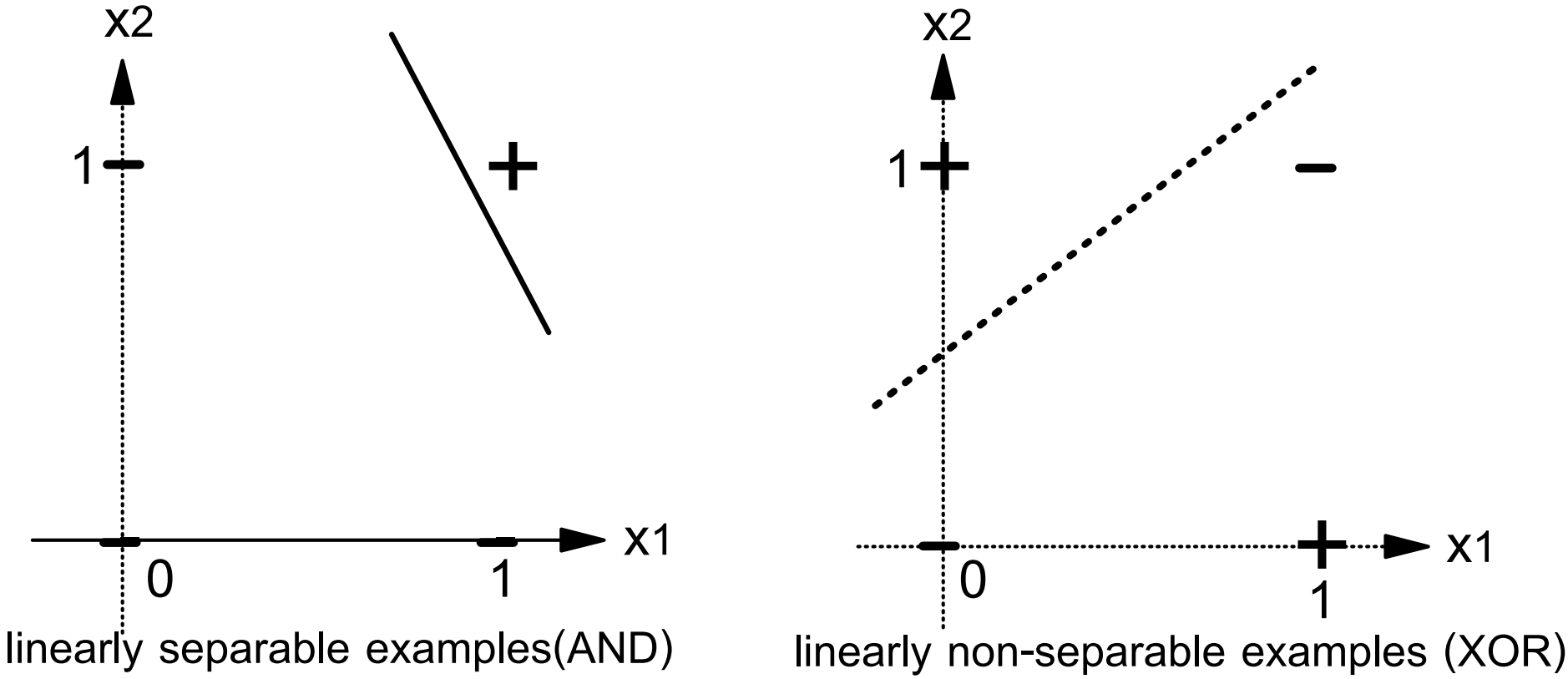
ตารางที่ 5.9.2 **ความพยายามของ** perceptron **ที่จะเรียนฟังก์ชัน** XOR

(binary activation function (output = 0 **หรือ** 1))

Perceptron Learning Example											
		Bias Input X0 = +1					Alpha = 0.80				
Input	Input				Net Sum	Target	Actual	Alpha	Weight Values		
X1	X2	1.0*W0	X1*W1	X2*W2	Input	Out	Out	*Error	W0	W1	W2
									0.1	0.1	0.1
0	0	0.10	0.00	0.00	0.10	0	1	-0.20	-0.10	0.10	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	0.10	0.60
1	0	0.40	0.10	0.00	0.50	1	1	0.00	0.40	0.10	0.60
1	1	0.40	0.10	0.60	1.10	0	1	-0.50	-0.10	-0.40	0.10
0	0	-0.10	0.00	0.00	-0.10	0	0	0.00	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10
0	0	0.40	0.00	0.00	0.40	0	1	-0.50	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10
0	0	0.40	0.00	0.00	0.40	0	1	-0.50	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10
0	0	0.40	0.00	0.00	0.40	0	1	-0.50	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10
0	0	0.40	0.00	0.00	0.40	0	1	-0.50	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10
0	0	0.40	0.00	0.00	0.40	0	1	-0.50	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10
0	0	0.40	0.00	0.00	0.40	0	1	-0.50	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10
0	0	0.40	0.00	0.00	0.40	0	1	-0.50	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10

ข้อจำกัดของ Perceptron Learning Rule

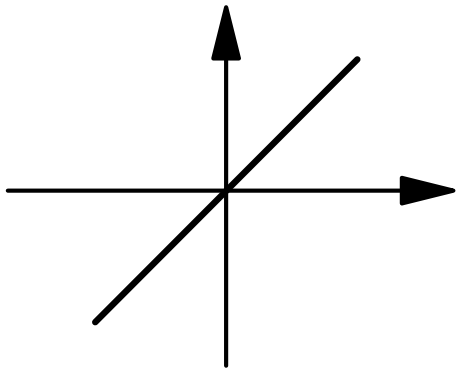
- concept หรือ function ที่สามารถเรียนได้โดย perceptron learning rule นั้น จะต้องเป็น linearly separable function
- ถ้าไม่เป็นแบบ linearly separable, perceptron จะไม่ลู่เข้า (รูปที่ 5.9.4)



รูปที่ 5.9.4 linearly and linearly non-separable function

Delta Rule (Gradient Descent)

- delta rule คล้ายกับ perceptron learning rule แต่จะลู่เข้าสู่ค่าที่ทำให้ error น้อยที่สุด ในกรณีที่ตัวอย่างไม่เป็นแบบ linearly separable
- ใช้หลักการของ gradient descent เพื่อหาคำตอบจาก space ของเวกเตอร์น้ำหนักที่เป็นไปได้
- เป็นพื้นฐานของอัลกอริทึม Backprogagation
- ใช้ activation function เป็น linear function ซึ่งหาอนุพันธ์ได้ (รูปที่ 5.9.5)

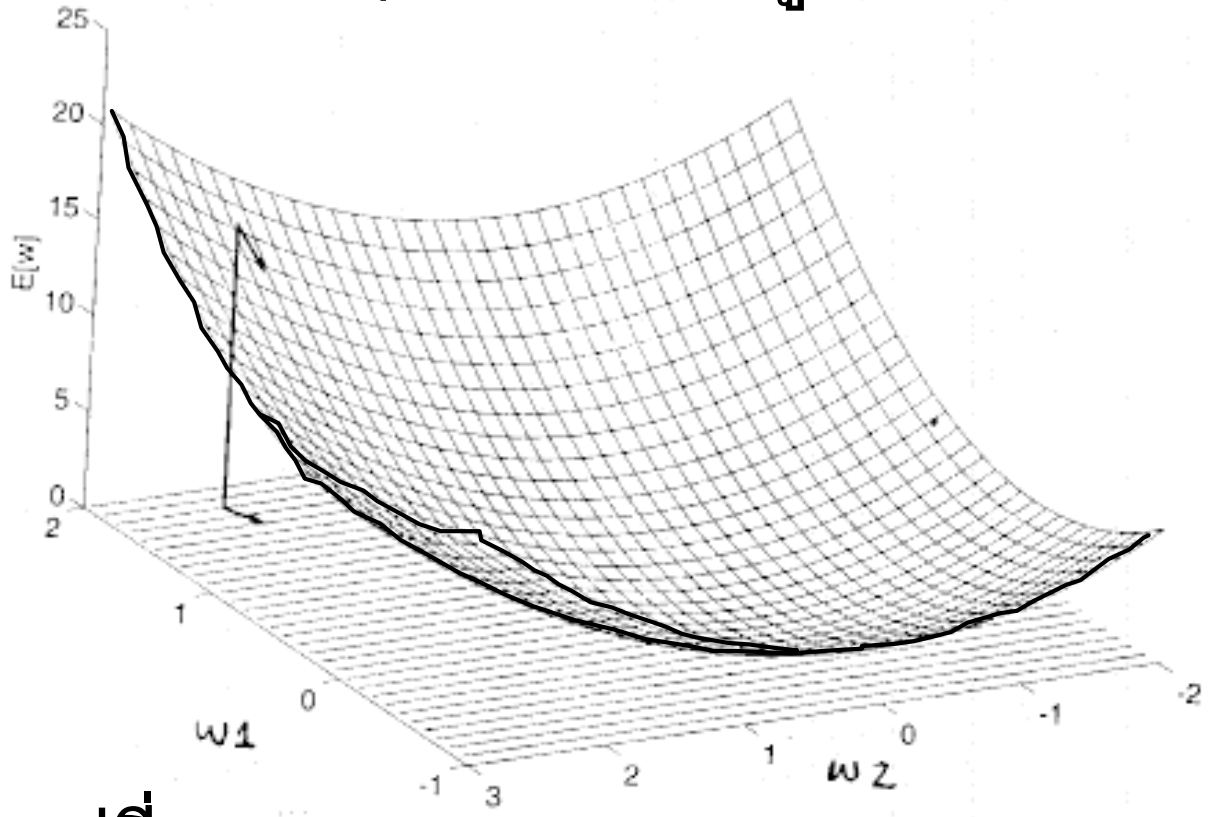


รูปที่ 5.9.5 linear activation function

- เอ๊าท์พุตของ perceptron แสดงโดย $o(\vec{x}) = \vec{w} \cdot \vec{x}$

- นิยาม training error (E(w)) $E(\vec{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$

  โดยที่ D เป็นเซตของตัวอย่าง, td เป็น target output ของตัวอย่าง d od เป็น output ของ perceptron สำหรับตัวอย่าง d
- $E(\vec{w})$เป็น parabolic function ของ$\vec{w}$ ซึ่งมีค่าต่ำสุดค่าเดียว
- ตัวอย่างของ$E(\vec{w})$ (2 inputs) แสดงในรูปที่ 5.9.6



รูปที่ 5.9.6 Hypothesis Space of w and E(w)

Derivation of the Delta Rule

- หลักการของ delta rule คือหาค่า $\vec{w}$ ที่ทำให้ $E(\vec{w})$ มีค่าน้อยที่สุด
- เริ่มจากเวกเตอร์น้ำหนักเริ่มต้นแล้วปรับค่าเวกเตอร์น้ำหนักทีละน้อย ในทิศทางลงที่ชันที่สุด(steepest descent)ของerror surface(รูป5.9.6)
- เวกเตอร์ที่สัมผัสกับ error surface คำนวณได้จากอนุพันธ์ของ $E(\vec{w})$ เทียบกับ $\vec{w}$ (ให้เวกเตอร์นี้แทนด้วย $\nabla E(\vec{w})$)

$$\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n} \right]$$

เวกเตอร์นี้แสดงทิศในแนวขึ้น, เวกเตอร์ที่มีทิศทางลงจึงเป็น $-\nabla E(\vec{w})$

- ดังนั้นกฎในการปรับค่าเวกเตอร์น้ำหนักเป็น:

$$\vec{w} = \vec{w} + \Delta\vec{w}$$

โดยที่

$$\Delta\vec{w} = -\eta \nabla E(\vec{w})$$

η : learning rate เป็นค่าคงที่เลขบวก

- Delta rule สามารถเขียนให้อยู่ในรูปของสมาชิกแต่ละตัวได้

$$w_i = w_i + \Delta w_i$$

โดยที่

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

- $\frac{\partial E}{\partial w_i}$ คำนวณได้โดย:

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id})$$

xid คือสมาชิก xi ของตัวอย่าง d

$$\therefore \Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

Delta Rule Algorithm

Delta-Rule(*training-examples*, η)

Each training example is a pair $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and *t* is the target output value. η is the learing rate.

- Initialize each *wi* to some small random value
- Until the termination condition is met, Do
  - Initialize each Δwi to zero.
  - For each $\langle \vec{x}, t \rangle$ in training-examples, Do
    - -- Input the instance $\vec{x}$ to the unit and compute the output *o*
    - -- For each linear unit weight *wi*, Do

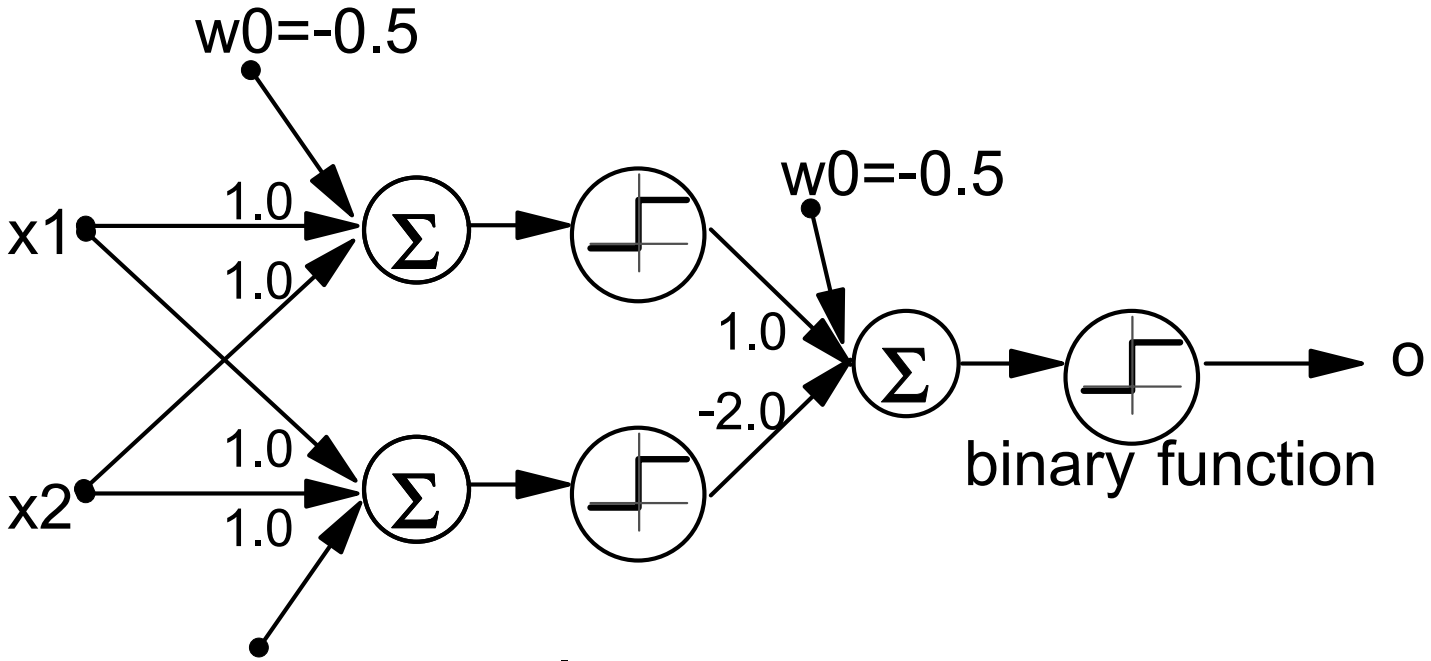
$$\Delta w_i = \Delta w_i + \eta \, (t - o) \, x_i$$

  - For each linear weight wi, Do

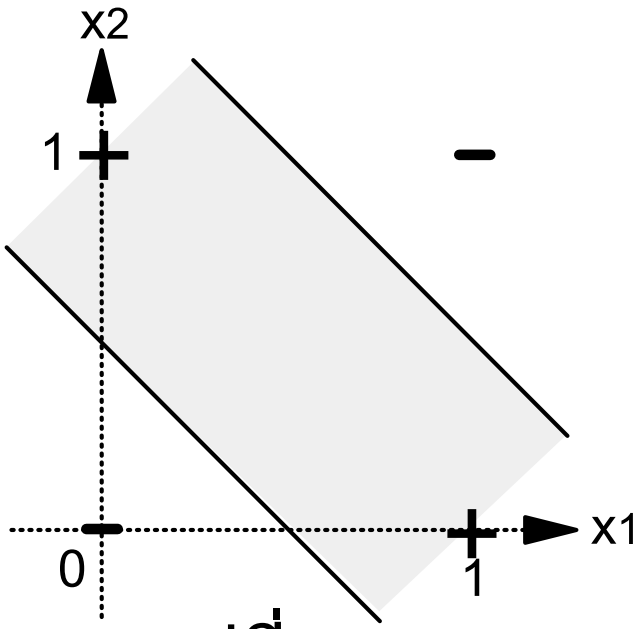
$$w_i = w_i + \Delta w_i$$

Multilayer Network and Backpropagation

- perceptron เดี่ยวสามารถแสดงได้แค่ linear decision surface เท่านั้น
- เน็ตเวิร์กหลายชั้น (multilayer network) สามารถแสดง nonlinear decision surface ได้
- ตัวอย่างของmultilayer networkที่แสดงฟังก์ชัน XORแสดงในรูป 5.9.7 และ decision surface แสดงในรูป 5.9.8



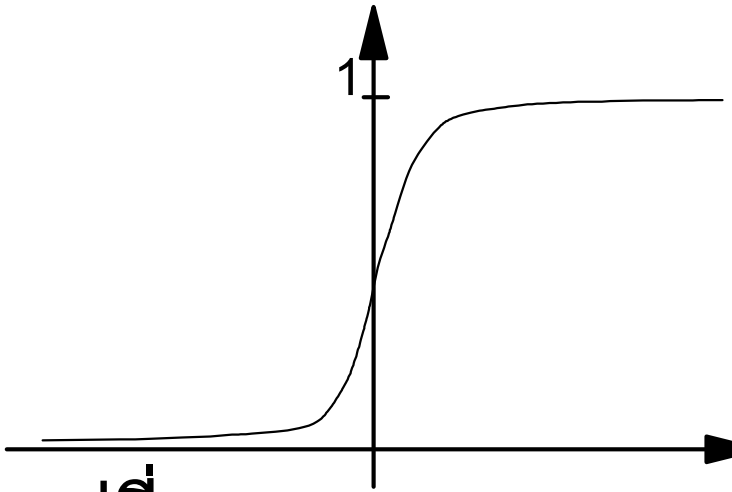
รูปที่ 5.9.7 multilayer network solving XOR



x1	x2	x1 XOR x2
0	0	0
0	1	1
1	0	1
1	1	0

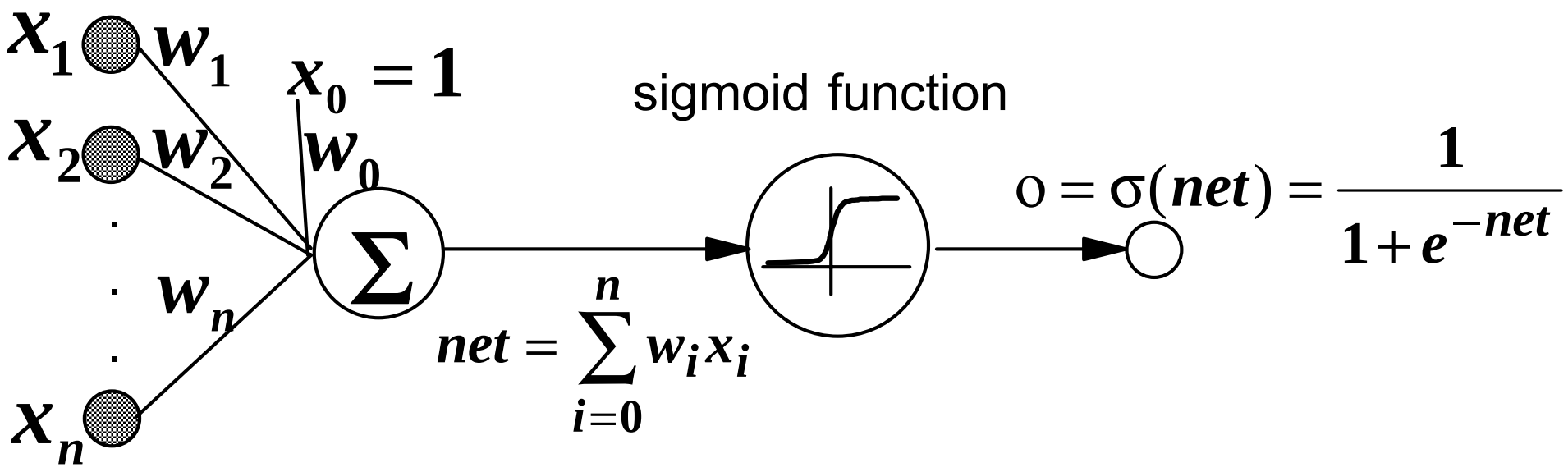
รูปที่ 5.9.8 decision surface **ของ** network **ในรูปที่** 5.9.7

- Multilayer network **ใช้** activation function **ที่สามารถหาค่าอนุพันธ์ได้ เช่น** sigmoid function (**แสดงในรูปที่** 5.9.9)



$$\sigma(y) = \frac{1}{1+e^{-y}}$$

รูปที่ 5.9.9 Sigmoid function



รูปที่ 5.9.10 Perceptron ที่ใช้ Sigmoid function

- คุณสมบัติหนึ่งของ sigmoid function คือ ค่าอนุพันธ์แสดงอยู่ในรูปของเอ๊าท์พุตได้อย่างง่าย

$$\frac{d\sigma(y)}{dy} = \sigma(y) \cdot (1 - \sigma(y))$$

Backpropagation Algorithm

- Backpropagation (BP) algorithm เรียนรู้ค่าเวกเตอร์น้ำหนักสำหรับ multilayer feedforward network (MLFF) โดยการใช้ gradient descent เพื่อหาค่าต่ำสุดของerrorระหว่างเอ๊าท์พุตของเน็ตเวอร์กับtarget value
- ค่า error (E) นิยามเป็น $$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

  โดยที่ outputs คือเซตของ output units ในเน็ตเวิร์ก,

  t_{kd} และ o_{kd} เป็น target value และ output value ตามลำดับของ output unit ที่ k และตัวอย่างที่ d
- BP ค้นหาเวกเตอร์น้ำหนักที่ให้ค่า error ต่ำสุด แต่ในกรณีของ MLFF ค่าต่ำสุดมีมากกว่าหนึ่ง ดังนั้นคำตอบของ BP จึงเป็น local minimum

Backpropagation(training-examples, η*,*n_{in}*,*n_{out}*,*n_{hidden}*)*

Each training example is a pair $\langle \vec{x}, \vec{t} \rangle$, where $\vec{x}$ is the input vector, $\vec{t}$ is the target output vector, η is the learning rate. n_{in}, n_{out}, n_{hidden} are number of network inputs, units in the hidden layer, output units, respectively. The input from unit i into unit j, and the weight from unit i to unit j are denoted x_{ji} and w_{ji}

- Initialize all network weights to small random numbers (e.g., [-0.05..0.05])
- Until the termination condition is met, Do
- For each $\langle \vec{x}, \vec{t} \rangle$ in training-examples, Do

  {Propagate the input forward through the network}

  1. Input the instance $\vec{x}$ to the network, compute the output o_u of every unit u.

  {Propagate the errors backward through the network}

  2. For each network output unit k, calculate its error term δ_k

$$\delta_k = o_k(1-o_k)(t_k - o_k)$$

  3. For each hidden unit h, calculate its error term δ_h

$$\delta_h = o_h(1-o_h) \sum_{k \in outputs} w_{kh}\delta_k$$

  4. Update each network weight w_{ji}: $w_{ji} = w_{ji} + \Delta w_{ji}$

  where $\Delta w_{ji} = \eta \delta_j x_{ji}$

Derivation of Backpropagation Rule

- ให้errorของแต่ละตัวอย่าง d เป็น Ed แล้วน้ำหนัก wji ถูกปรับโดย Δw_{ji} ตาม gradient descent

$$\Delta w_{ji} = -h \frac{\partial E_d}{\partial w_{ji}}$$

โดยที่ Ed เป็น error ของตัวอย่าง d คำนวณรวมทั้งหมดสำหรับทุก output unit

$$E_d(\vec{w}) = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

outputs เป็นเซตของ output units, tk และ ok เป็น target value และ output value ของ unit k สำหรับตัวอย่าง d ตามลำดับ

- ให้ $net_j = \sum_i w_{ji} x_{ji}$ เราจะได้ว่า $\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji}$

- พิจารณา $\frac{\partial E_d}{\partial net_j}$ จากสมการที่แล้วเป็น 2 กรณี:
  1 กรณีที่ unit j เป็น output unit และ 2 กรณีที่ unit j เป็น hidden unit
- กรณีที่ 1: training rule สำหรับน้ำหนักของ output unit
  เนื่องจาก wji มีผลต่อเน็ตเวิร์กโดยผ่านทาง *netj*
  และ *netj* มีผลต่อเน็ตเวิร์กโดยผ่านทาง *oj* เท่านั้น

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

- คำนวณ $\frac{\partial E_d}{\partial o_j}$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2$$

เพราะว่า อนุพันธ์ของพจน์อื่นๆที่ k ที่ไม่เท่ากับ j เป็น 0

$$\frac{\partial E_d}{\partial o_j} = -(t_j - o_j)$$

– คำนวณ $\frac{\partial o_j}{\partial net_j}$,

จากคุณสมบัติของ sigmoid function

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j}$$

$$= o_j(1-o_j)$$

∴ จะได้ว่า $\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)o_j(1-o_j)$

- ดังนั้น gradient descent rule สำหรับ output unit จึงเป็น

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

$$= \eta(t_j - o_j)o_j(1-o_j)x_{ji}$$

- กรณีที่ 2: training rule สำหรับน้ำหนักของ hidden unit

  ในการคำนวณกฎสำหรับ wji ซึ่ง j เป็น hidden unit นั้น เราต้องพิจารณาผลทางอ้อมของ wji ที่มีต่อเอ๊าท์พุตของเน็ตเวิร์กและ Ed

  - ให้ Downstream(j) เป็น เซตของ units ทั้งหมดที่ได้รับ input จาก output ของ unit j
  - netj สามารถส่งผลต่อเอ๊าท์พุตของเน็ตเวิร์ก (และ Ed) ได้โดยผ่านทาง unit ใน Downstream(j) เท่านั้น

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k w_{kj} o_j (1 - o_j)$$

– ให้ δ_j แสดง $-\frac{\partial E_d}{\partial net_j}$

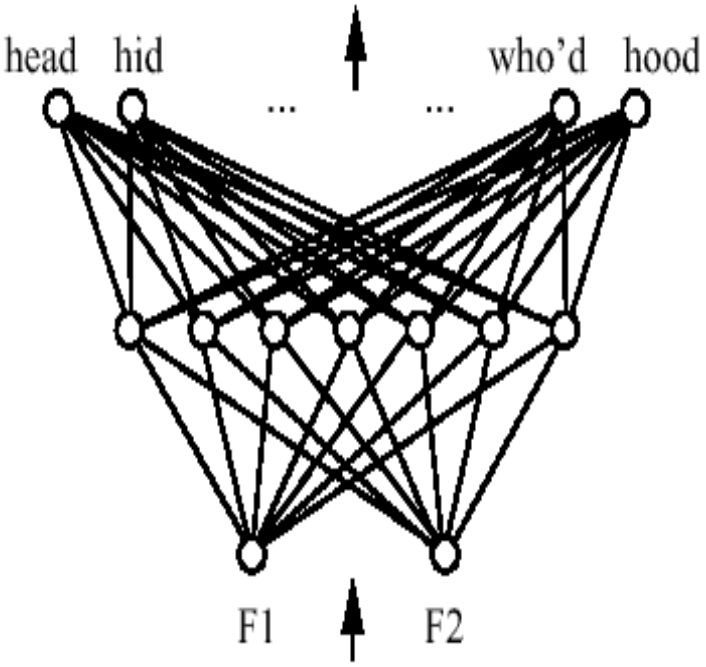
$$\delta_j = o_j (1 - o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj}$$

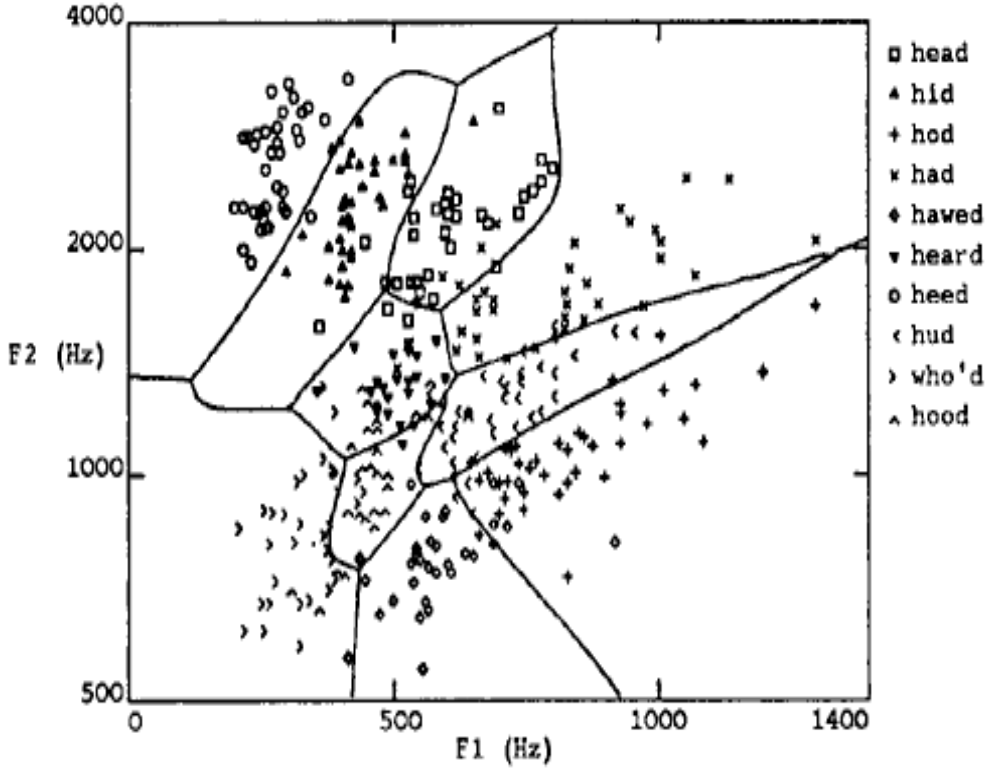
และ $\Delta w_{ji} = \eta \delta_j x_{ji}$

- ในอัลกอริทึมเป็นกรณีที่ Downstream(j) = outputs

An Example of Multilayer Networks

- ตัวอย่างของ multilayer networks ที่ใช้รู้จำเสียงพูด





5.10 Bayesian Learning

- Bayesian learning เป็นเทคนิคที่ใช้ทฤษฎีความน่าจะเป็น (Bayes theorem) เพื่อหาว่าสมมติฐาน (hypothesis)ใด น่าจะถูกต้องที่สุด
- ใช้ความรู้ก่อนหน้า (prior knowledge) ร่วมกับข้อมูล (data) เพื่อหาสมมติฐานที่ดีที่สุด
  - ความน่าจะเป็นก่อนหน้าสำหรับสมมติฐานหนึ่ง ๆ
  - ความน่าจะเป็นของข้อมูลที่สังเกตุได้สำหรับสมมติฐานหนึ่ง ๆ
- เป็นเทคนิคที่ใช้งานจริงได้ดี เช่น naive Bayes learning, Bayesian belief network learning

Bayes Theorem

- $P(h \mid D) = \dfrac{P(D \mid h) * P(h)}{P(D)}$

  โดยที่ P(h | D) คือ ความน่าจะเป็นของ h เมื่อรู้ D

  P(D | h) คือ ความน่าจะเป็นของ D เมื่อรู้ h

  P(h) คือ ความน่าจะเป็นก่อนหน้าของสมมติฐาน h

  P(D) คือ ความน่าจะเป็นก่อนหน้าของเซตของตัวอย่าง D

- ใช้ Bayes theorem ในการหาสมมติฐานที่น่าจะเป็นที่สุดเมื่อรู้ D

  Maximum a posteriori hypothesis : h_{MAP}

$$h_{MAP} = \arg\max_{h \in H} P(h \mid D) = \arg\max_{h \in H} \frac{P(D \mid h) * P(h)}{P(D)}$$

$$= \arg\max_{h \in H} P(D \mid h) * P(h)$$

- Maximum Likelihood (h_{ML}) ($P(h_i)=P(h_j)$) $= \arg\max_{h \in H} P(D \mid h)$

ตัวอย่างการเลือกสมมติฐานโดย Bayes Theorem

- คนไข้คนหนึ่งไปตรวจหามะเร็ง ผลการตรวจเป็นบวก
  - ผลการตรวจเมื่อเป็นบวกจะให้ความถูกต้อง 98% ของกรณีที่มีโรคนั้นอยู่จริง
  - ผลการตรวจเมื่อเป็นลบจะให้ความถูกต้อง 97% ของกรณีที่ไม่มีโรคนั้น
  - นอกจากนั้น 0.008 ของประชากรทั้งหมดเป็นโรคมะเร็ง
- คนไข้คนนี้เป็นมะเร็งจริงหรือไม่?

P(cancer) = P(¬cancer) =

P(+ | cancer) = P(– | cancer) =

P(+ | ¬cancer) = P(– | ¬cancer) =

h_{MAP} =

สูตรพื้นฐานเกี่ยวกับความน่าจะเป็น

- Product Rule : ความน่าจะเป็น $P(A \wedge B)$ ที่สองเหตุการณ์ A และ B จะเกิดพร้อมกัน

$$P(A \wedge B) = P(A|B)P(B) = P(B|A)P(A)$$

- Sum Rule : ความน่าจะเป็น $P(A \vee B)$ ที่เหตุการณ์ A หรือ B จะเกิด

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

- Theorem of total probability : ถ้าเหตุการณ์ $A_1,...,A_n$ ไม่เกิดร่วมกัน และ $\sum_{i=1}^{n} P(A_i) = 1$ แล้ว

$$P(B) = \sum_{i=1}^{n} P(B \mid A_i)P(A_i)$$

- Chain Rule : $A_1, A_2, ..., A_n$ เป็นเหตุการณ์ n เหตุการณ์

$$P(A_1, A_2, ..., A_n) = \sum_{i=1}^{n} P(A_i \mid A_{i-1}, ..., A_1)$$

การแยกแยะที่น่าจะเป็นที่สุดสำหรับตัวอย่าง

- h_{MAP} เป็นสมมติฐานที่น่าจะเป็นที่สุด แต่ไม่เป็นการแยกแยกตัวอย่างที่น่าจะเป็นที่สุด (most probable classification)
- พิจารณาสมมติฐานทั้งสามต่อไปนี้

$P(h_1 | D) = 0.4$ $P(h_2 | D) = 0.3$ $P(h_3 | D) = 0.3$

- เมื่อให้ตัวอย่าง x ผลการแยกแยะของสมมติฐานเป็นดังนี้

$h_1(x) = +$ $h_2(x) = -$ $h_3(x) = -$

- การแยกแยะตัวอย่าง x ที่น่าจะเป็นที่สุดคือ?

Bayes Optimal Classifier

- **ตัวแยกแยะที่ดีที่สุดแบบเบย์ :**

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j \mid h_i)\, P(h_i \mid D)$$

- **ตัวอย่าง**

$P(h_1 \mid D) = 0.4 \qquad P(- \mid h_1) = 0 \qquad P(+ \mid h_1) = 1$

$P(h_2 \mid D) = 0.3 \qquad P(- \mid h_2) = 1 \qquad P(+ \mid h_2) = 0$

$P(h_3 \mid D) = 0.3 \qquad P(- \mid h_3) = 1 \qquad P(+ \mid h_3) = 0$

$$\Rightarrow \sum_{h_i \in H} P(+ \mid h_i)\, P(h_i \mid D) = 0.4$$

$$\sum_{h_i \in H} P(- \mid h_i)\, P(h_i \mid D) = 0.6$$

- **ดังนั้น** $\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j \mid h_i)\, P(h_i \mid D) = -$

Naive Bayes Classifier

- ตัวแยกแยะเบย์อย่างง่าย (naive Bayes classifier) เป็นตัวแยก-แยะประเภทหนึ่งที่ใช้งานได้ดี เหมาะกับกรณีของ
  - เซตตัวอย่างมีจำนวนมาก
  - คุณสมบัติ (attribute) ของตัวอย่างไม่ขึ้นต่อกัน
- ประยุกต์ใช้งานในด้าน
  - การแยกแยะข้อความ (text classification)
  - การวินิจฉัย (diagnosis)

Naive Bayes Classifier

- ให้ $a_1, a_2, ..., a_n$ เป็นคุณสมบัติของตัวอย่าง
- ค่าที่น่าจะเป็นที่สุดของตัวอย่าง x คือ

$$v_{MAP} = \arg\max_{v_j \in V} P(v_j \mid a_1, a_2, ..., a_n)$$

$$v_{MAP} = \arg\max_{v_j \in V} \frac{P(a_1, a_2, ..., a_n \mid v_j) * P(v_j)}{P(a_1, a_2, ..., a_n)}$$

$$v_{MAP} = \arg\max_{v_j \in V} P(a_1, a_2, ..., a_n \mid v_j) * P(v_j)$$

- Naive Bayes assumption :

$$P(a_1, a_2, ..., a_n \mid v_j) = \prod_i P(a_i \mid v_j)$$

- Naive Bayes classifier : $v_{NB} = \arg\max_{v_j \in V} P(v_j) * \prod_i P(a_i \mid v_j)$

Naive Bayes Algorithm

- Naive_Bayes_Learn(examples)

  For each target value v_j

  $\hat{P}(v_j) \leftarrow$ estimate $P(v_j)$

  For each attribute value a_i of each attribute a

  $\hat{P}(a_i \mid v_j) \leftarrow$ estimate $P(a_i \mid v_j)$

- Classify_New_Example(x)

$$v_{NB} = \arg\max_{v_j \in V} \hat{P}(v_j) * \prod_i \hat{P}(a_i \mid v_j)$$

Name	Hair	Height	Weight	Lotion	Result
Sarah	blonde	average	light	no	+
Dana	blonde	tall	average	yes	–
Alex	brown	short	average	yes	–
Annie	blonde	short	average	no	+
Emily	red	average	heavy	no	+
Pete	brown	tall	heavy	no	–
John	brown	average	heavy	no	–
Katie	blonde	short	light	yes	–

- **สมมติว่าตัวอย่างที่ต้องการแยกแยะคือ**

  Judy blonde average heavy no class = ?

- **คำนวณ** $v_{NB} = \arg\max_{v_j \in V} \hat{P}(v_j) * \prod_i \hat{P}(a_i \mid v_j)$

  $P(+)P(\text{blonde}|+)P(\text{average}|+)P(\text{heavy}|+)P(\text{no}|+) = 1/18$

  $P(-)P(\text{blonde}|-)P(\text{average}|-)P(\text{heavy}|-)P(\text{no}|-) = 1/125$

  $\Rightarrow \quad v_{NB} = +$

Learning To Classify Text

Target concept *Interest?* : Document $\rightarrow$ {+, –}

1. Represent each document by vector of words
   - one attribute per word position in document
2. Learning: Use training examples to estimate
   - P(+)
   - P(–)
   - P(doc | +)
   - P(doc | –)

Naive Bayes conditional independence assumption

$$P(doc \mid v_j) = \prod_{i=1}^{length(doc)} P(a_i = w_k \mid v_j)$$

- $P(a_i = w_k \mid v_j)$: **ความน่าจะเป็นที่คำในตำแหน่งที่** i **เป็น** w_k **เมื่อรู้** v_j
- **สมมติฐานเพิ่มเติม** $P(a_i = w_k \mid v_j) = P(a_m = w_k \mid v_j), \forall i,m$

Learn_naive_Bayes_text

Learn_naive_Bayes_text(*Examples,V*)

1. Collect all words and other tokens that occur in *Examples*

- *Vocabulary* ← all distinct words and other tokens in *Examples*

2. Calculate the required $P(v_j)$ and $P(w_k \mid v_j)$

- For each target value v_j in V do
  - $docs_j$ ← subset of *Examples* for which the target value is v_j
  - $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
  - $Text_j$ ← a single document created by concatenating all members of $docs_j$
  - n ← total number of words in $Text_j$ (counting duplicate words multiple times)

Learn_naive_Bayes_text

-- for each word w_k in *Vocabulary*

$n_k \leftarrow$ number of times word w_k occurs in $Text_j$

$$P(w_k \mid v_j) \leftarrow \frac{n_k + 1}{n + |Vocabulary|}$$

Classify_naive_Bayes_text(*Doc*)

- *positions* ← all word positions in *Doc* that contain tokens found in *Vocabulary*
- Return v_{NB} where

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) * \prod_{i \in positions} P(a_i \mid v_j)$$

Bayesian Belief Networks

- Naive Bayes assumption มีข้อจำกัดมากเกินไป
- Bayesian belief network ใช้อธิบายความไม่ขึ้นต่อกันอย่างมีเงื่อนไข (conditional independent) ระหว่างตัวแปร
- ทำให้เราสามารถใช้

  (1) ความรู้ก่อนหน้า (prior knowledge) เกี่ยวกับความ(ไม่)ขึ้นต่อกันระหว่างตัวแปร,

  ร่วมกับ

  (2) ตัวอย่าง
- บางครั้งเราเรียกว่า Bayes Net

Conditional Independence

- Definition: X is conditionally independent of Y given Z

  if the probability distribution governing X is independent

  of the value of Y given the value of Z; that is, if

  $$(\forall\, x_i, y_j, z_k)\ P(X=x_i \mid Y=y_j, Z=z_k) = P(X=x_i \mid Z=z_k)$$

  or simply,

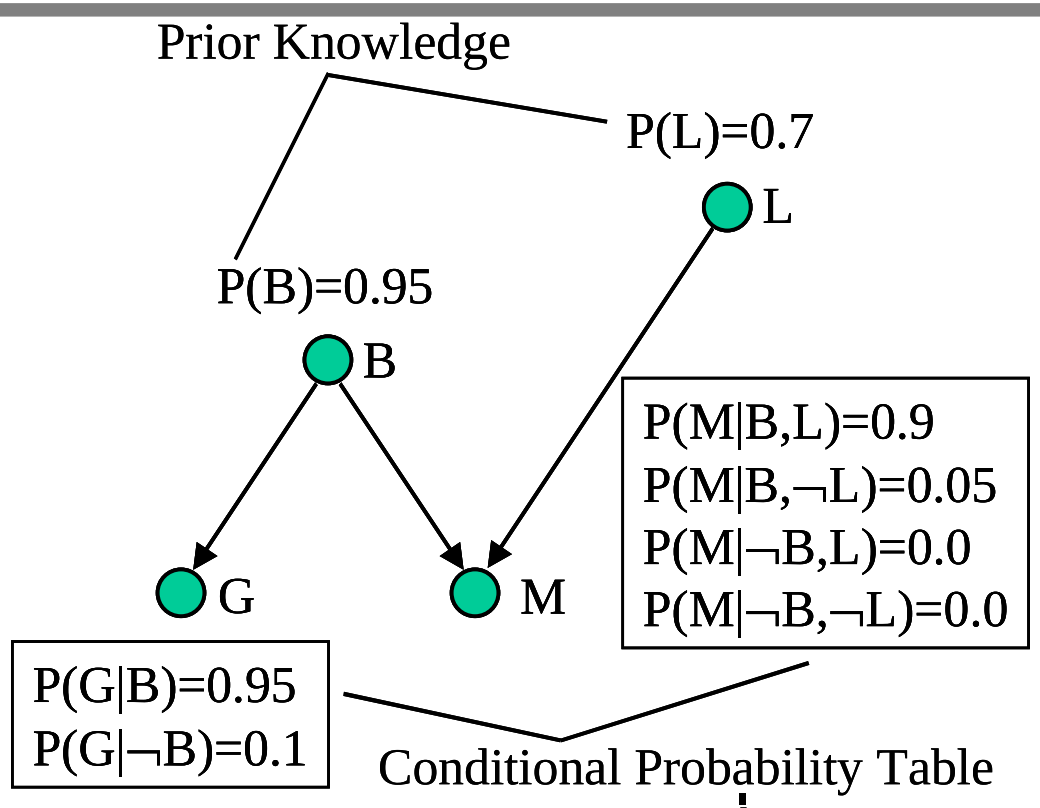
  $$P(X \mid Y,Z) = P(X \mid Z)$$

- Example: Thunder is conditionally independent of Rain,

  given Lightning

  P(Thunder | Rain, Lightning) = P(Thunder | Lightning)

Bayesian Belief Network



B : Battery is charged

L : block is Lifetable

M : arm Moves

G : Gauge indicates that battery is charged

รูปที่ 5.10.1

- Bayes net **แสดงเซตของความ (ไม่) ขึ้นต่อกันของตัวแปร**
- **แสดงโดย** Directed Acyclic Graph (DAG)
- **แต่ละโนดไม่ขึ้นต่อกันอย่างมีเงื่อนไข** (conditional independent) **กับโนดอื่น เมื่อรู้โนดพ่อแม่โดยตรง** (immediate predecessors)

Bayesian Belief Network

- **แสดงการกระจายความน่าจะเป็นร่วม** (joint probability distribution) **ระหว่างตัวแปร**
  - **เช่น** P(Battery,Liftable,Gauge,Move)
- **โดยทั่วไป**

$$P(y_1,....,y_n) = \prod_{i=1}^{n} P(y_i \mid Parents(Y_i))$$

โดยที่ Parents(Y_i) **หมายถึงโนดพ่อแม่โดยตรงของโนด** Y_i

- **การกระจายความน่าจะเป็นร่วมนิยามโดยกราฟและ** $P(y_i \mid Parents(Y_i))$
- **จากตัวอย่าง** P(G,M,B,L) = P(G|B,M,L)P(M|B,L)P(B|L)P(L)
  = P(G|B)P(M|B,L)P(B)P(L)

รูปแบบการอนุมานใน Bayes Net (1)

1. Causal Reasoning : การอนุมานจากเหตุ

   เช่น ต้องการคำนวณ P(M|L) -- หาความน่าจะเป็นที่แขนจะเคลื่อนได้เมื่อรู้ว่ากล่องยกได้ (กล่องยกได้เป็นสาเหตุหนึ่งของการที่แขนจะเคลื่อนได้)

   - กระจาย P(M|L) ให้อยู่ในรูปของผลรวมของความน่าจะเป็นร่วมระหว่าง M กับโนดพ่อแม่อื่นนอกจาก L (M กับ B)

     $$P(M|L) = P(M,B|L) + P(M,\neg B|L)$$

   - จัดรูปให้ M ขึ้นกับโนดพ่อแม่ (B,L) โดยใช้ chain rule

     $$P(M|L) = P(M|B,L)P(B|L) + P(M|\neg B,L)P(\neg B|L)$$

     $$\Rightarrow P(M|L) = P(M|B,L)P(B) + P(M|\neg B,L)P(\neg B)$$

     $$= 0.855$$

รูปแบบการอนุมานใน Bayes Net (2)

2. Diagnosis Reasoning : การอนุมานจากผล

เช่น ต้องการคำนวณ $P(\neg L|\neg M)$ -- ความน่าจะเป็นที่กล่องยกไม่ได้เมื่อรู้ว่าแขนไม่ได้เคลื่อน (ใช้ผล -- อาการ เพื่อหาสาเหตุ)

$$P(\neg L|\neg M) = \frac{P(\neg M|\neg L)P(\neg L)}{P(\neg M)} \qquad \text{(Bayes' rule)}$$

- คำนวณ $P(\neg M|\neg L)$ ได้ 0.9525 (ใช้ causal reasoning)
- คำนวณ $P(\neg L|\neg M) = \frac{0.9525 * 0.3}{P(\neg M)} = \frac{0.28575}{P(\neg M)}$
- ในทำนองเดียวกัน $P(L|\neg M) = \frac{P(\neg M|L)P(L)}{P(\neg M)}$

$$= \frac{0.0595*0.7}{P(\neg M)} = \frac{0.03665}{P(\neg M)}$$

- $P(\neg L|\neg M) + P(L|\neg M) = 1 \Rightarrow P(\neg L|\neg M) = 0.88632$

รูปแบบการอนุมานใน Bayes Net (3)

3. Explaining away : ทำ causal reasoning ภายใน diagnostic reasoning

- เช่น เมื่อรู้ $\neg$ M (แขนไม่เคลื่อน) เราสามารถคำนวณ $\neg$ L (ความน่าจะเป็นที่กล่องไม่สามารถยกได้)
- แต่ถ้าเรารู้ $\neg$ B (แบตเตอรีไม่ได้ชาร์จ) แล้ว $\neg$ L ควรจะมีค่าความน่าจะเป็นน้อยลง
- ในกรณีนี้เราเรียกว่า $\neg$ B explain $\neg$ M, making $\neg$ L less certain

$$P(\neg L|\neg B,\neg M) = \frac{P(\neg M,\neg B|\neg L)P(\neg L)}{P(\neg B,\neg M)} \quad \text{(Bayes' rule)}$$

$$= \frac{P(\neg M|\neg B,\neg L)P(\neg B|\neg L)P(\neg L)}{P(\neg B,\neg M)}$$

$$= \frac{P(\neg M|\neg B,\neg L)P(\neg B)P(\neg L)}{P(\neg B,\neg M)}$$

- หลังคำนวณ $P(\neg B,\neg M)$ เราได้ $P(\neg L|\neg B,\neg M) = 0.030$

การเรียนรู้ Bayes Net

- การเรียนรู้ Bayes Net คือการหาเน็ตเวิร์ก (structure และ conditional probability tables) ที่สอดคล้องกับตัวอย่างมากที่สุด
- ปัจหาการเรียนรู้แบ่งเป็น

  (1) structure unknown

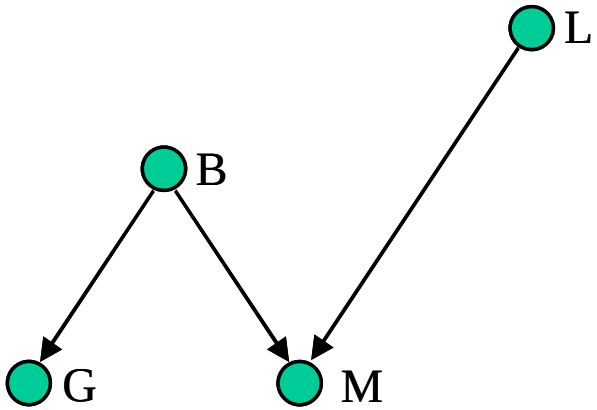
  (2) structure known

  (2.1) no missing value data

  (2.2) with missing value data
- กรณีของ structure known + no missing value data เป็นกรณีที่ง่ายที่สุด สามารถทำการเรียนรู้ได้ในลักษณะเดียวกับ naive Bayes classifer

Structure Known with No Missing Value Data (Learn CPTs)

- คำนวณ CPT สำหรับแต่ละโนด
- ต้องใช้ตัวอย่างจำนวนมากเพื่อให้ค่าทางสถิติถูกต้อง



รูปที่ 5.10.2

G	M	B	L	no. of instances
True	True	True	True	54
True	True	True	False	1
True	False	True	True	7
True	False	True	False	27
False	True	True	True	3
False	False	True	False	2
False	False	False	True	4
False	False	False	False	2
				100

- $\hat{P}(V_i = v_i \mid \text{Parents}(V_i) = \mathbf{P}_i) = \dfrac{\text{จำนวนตัวอย่างที่มี } V_i = v_i}{\text{จำนวนตัวอย่างที่มี Parents}(V_i) = \mathbf{P}_i}$
- P(B=true) = (54+1+7+27+3+2)/100 = 0.94
- P(L=true) = (54+7+3+4)/100 = 0.68
- P(M|B,¬ L) = อัตราส่วนที่ M=true เมื่อ B=true, L=false
  = 1/(1+27+2) = 0.03
- เราสามารถคำนวณ CPT สำหรับโนด G ได้ในทำนองเดียวกัน

Structure Known with Missing Value Data (Learn CPTs)

- ตัวอย่างเช่นข้อมูลต่อไปนี้ โดยที่ * แทน missing value

G	M	B	L	no. of instances
True	True	True	True	54
True	True	True	False	1
*	*	True	True	7
True	False	True	False	27
False	True	*	True	3
False	False	True	False	2
False	False	False	True	4
False	False	False	False	2

รูปที่ 5.10.3

- พิจารณากรณีของ 3 ตัวอย่างที่มีค่า G=false, M=true, L=true ในกรณีนี้เราไม่รู้ค่าของ B แต่อาจคำนวณ $P(B|\neg G,M,L)$ หรือ $P(\neg B|\neg G,M,L)$ ได้ ถ้าเรารู้ CPT (แต่เรายังไม่รู้)
- จากนั้น เราจะแทนที่ตัวอย่างทั้งสามด้วยตัวอย่างมีน้ำหนัก (weighted examples) 2 ตัว
  - ตัวแรกคือตัวอย่างที่ B=True มีน้ำหนัก= $P(B|\neg G,M,L)$
  - ตัวที่สองคือตัวอย่างที่ B=False มีน้ำหนัก= $P(\neg B|\neg G,M,L)$

Structure Known with Missing Value Data (Learn CPTs)

- ในทำนองเดียวกัน กรณีของ 7 ตัวอย่างที่มีค่า B=true, L=true และ G และ M ไม่รู้ค่านั้น เราสามารถแทนที่ตัวอย่างทั้งเจ็ดด้วยตัวอย่างมีน้ำหนัก (weighted examples) 4 ตัว
  - (1) ตัวอย่าง G=true, M=true มีน้ำหนัก = P(G,M|B,L)
  - (2) ตัวอย่าง G=true, M=false มีน้ำหนัก = P(G,¬ M|B,L)
  - (3) ตัวอย่าง G=false, M=true มีน้ำหนัก = P(¬ G,M|B,L)
  - (4) ตัวอย่าง G=false, M=false มีน้ำหนัก = P(¬ G,¬ M|B,L)
- ในทำนองเดียวกัน เราสามารถค่าความน่าจะเป็นเหล่านี้ได้ถ้าเรารู้ CPTs และ structure ของ Bayes Net
- จากนั้นเราจะใช้ตัวอย่างมีน้ำหนักเหล่านี้ ร่วมกับตัวอย่าง-ที่เหลือเพื่อคำนวณ CPTs ได้ (ตัวอย่างที่ไม่รู้ค่าถูกแทนที่ด้วยตัวอย่างมีน้ำหนัก)

Expectation-Maximization (EM) Algorithm

- อัลกอริทึม EM สามารถใช้เพื่อคำนวณค่าน้ำหนักของตัวอย่างที่ไม่รู้ค่าได้ (คำนวณ CPTs)
  (1) เริ่มต้นโดย เราสุ่มค่าให้กับพารามิเตอร์ทั้งหมดของ CPTs
  (2) เราใช้ CPTs เพื่อคำนวณน้ำหนัก (ความน่าจะเป็นของค่าต่าง ๆของตัวอย่างที่ไม่รู้ค่า เมื่อรู้ตัวอย่างอื่น ๆ)
  (3) เราใช้น้ำหนักที่คำนวณได้ เพื่อประมาณ CPTs ใหม่
  (4) ทำซ้ำ (2) - (3) จนกระทั่ง CPTs ลู่เข้า
- โดยทั่วไปอัลกอริทึม EM จะใช้เวลาในการลู่เข้าไม่มาก

ตัวอย่างของอัลกอริทึม EM (1)

พิจารณาตัวอย่างในรูปที่ 5.10.3

(1) **สุ่มค่าสำหรับตาราง** CPTs

- P(L) = 0.5 (P(¬ L) = 1– P(L))
- P(B) = 0.5 (P(¬ B) = 1– P(B))
- P(M|B,L) = 0.5 (P(¬ M|B,L) = 1 – P(M|B,L))

  P(M|B,¬ L) = 0.5 (P(¬ M|B,¬ L) = 1 – P(M|B,¬ L))

  P(M|¬ B,L) = 0.5 (P(¬ M|¬ B,L) = 1 – P(M|¬ B,L))

  P(M|¬ B,¬ L) = 0.5 (P(¬ M|¬ B,¬ L) = 1 – P(M|¬ B,¬ L))
- P(G|B) = 0.5 (P(¬ G|B) = 1 – P(G|B))

  P(G|¬ B) = 0.5 (P(¬ G|¬ B) = 1 – P(G|¬ B))

ตัวอย่างของอัลกอริทึม EM (2)

(2) ใช้ CPTs เพื่อคำนวณน้ำหนัก

- ตัวอย่างที่ไม่รู้ค่าคือ

G	M	B	L	no. of instances
*	*	True	True	7
False	True	*	True	3

- ในกรณีของ 7 ตัวอย่างแรกเราต้องการหา P(G,M|B,L), P(G,¬ M|B,L), P(¬ G,M|B,L) และ P(¬ G,¬ M|B,L)
  - P(G,M|B,L) = P(G|B)*P(M|B,L) = 0.5*0.5
  - P(G,¬ M|B,L) = P(G|B)*P(¬ M|B,L) = 0.5*0.5
  - P(¬ G,M|B,L) = P(¬ G|B)*P(M|B,L) = 0.5*0.5
  - P(¬ G,¬ M|B,L) = P(¬ G|B)*P(¬ M|B,L) = 0.5*0.5

ตัวอย่างของอัลกอริทึม EM (3)

- แสดงว่า 7 ตัวอย่างแรก เราสามารถใส่น้ำหนักให้เป็นตัวอย่างดังต่อไปนี้

G	M	B	L	no. of instances
True	True	True	True	7*0.5*0.5=1.75
True	False	True	True	7*0.5*0.5=1.75
False	True	True	True	7*0.5*0.5=1.75
False	False	True	True	7*0.5*0.5=1.75

- ในกรณีของ 3 ตัวอย่าง

G	M	B	L	no. of instances
False	True	*	True	3

เราต้องการหา $P(B|\neg G,M,L)$ และ $P(\neg B|\neg G,M,L)$

- $P(B|\neg G,M,L) = P(B|\neg G,M) = \frac{P(B,\neg G,M)}{P(\neg G,M)}$

$$= \frac{P(\neg G|B)*P(M|B)*P(B)}{P(\neg G)*P(M)} \qquad \text{(E1)}$$

ตัวอย่างของอัลกอริทึม EM (4)

- P(M|B) ของสมการ E1 คำนวณได้ดังนี้

  $P(M|B) = P(M,L|B) + P(M,\neg L|B)$

  $= P(M|B,L)*P(L|B) + P(M|B,\neg L)*P(\neg L|B)$

  $= P(M|B,L)*P(L) + P(M|B,\neg L)*P(\neg L)$

- P(¬ G) ของสมการ E1 คำนวณได้ดังนี้

  $P(\neg G) = P(\neg G,B) + P(\neg G,\neg B)$

  $= P(\neg G|B)*P(B) + P(\neg G|\neg B)*P(\neg B)$

- P(M) ของสมการ E1 คำนวณได้ดังนี้

  $P(M) = P(M,B,L) + P(M,B,\neg L) + P(M,\neg B,L) + P(M,\neg B,\neg L)$

  $= P(M|B,L)*P(B)*P(L) + P(M|B,\neg L)*P(B)*P(\neg L) +$

  $P(M|\neg B,L)*P(\neg B)*P(L) + P(M|\neg B,\neg L)*P(\neg B)*P(\neg L)$

ตัวอย่างของอัลกอริทึม EM (5)

- สมการ E1 มีค่าดังนี้

$$P(B|\neg G,M,L) = \frac{0.5*(0.5*0.5+0.5*0.5)*0.5}{(0.5*0.5+0.5*0.5)* 4*(0.5*0.5*0.5)} = 0.5$$

$$P(\neg B|\neg G,M,L) = 0.5$$

- แสดงว่า 3 ตัวอย่าง เราสามารถใส่น้ำหนักให้เป็นตัวอย่างดังต่อไปนี้

G	M	B	L	no. of instances
False	True	True	True	3*0.5=1.5
False	True	False	True	3*0.5=1.5

- ตัวอย่างทั้งหมด

G	M	B	L	no. of instances
True	True	True	True	54
True	True	True	False	1
True	*True*	*True*	*True*	*1.75*
True	*False*	*True*	*True*	*1.75*
False	*True*	*True*	*True*	*1.75*
False	*False*	*True*	*True*	*1.75*
True	False	True	False	27
False	*True*	*True*	*True*	*1.5*
False	*True*	*False*	*True*	*1.5*
False	False	True	False	2
False	False	False	True	4
False	False	False	False	2

ตัวอย่างของอัลกอริทึม EM (6)

(3) ใช้ตัวอย่างมีน้ำหนักที่คำนวณได้ เพื่อประมาณ CPTs ใหม่

- P(L) = 68/100 = 0.680 (P(¬ L) = 1– P(L))
- P(B) = 92.5/100 = 0.925 (P(¬ B) = 1– P(B))
- P(M|B,L) = 59/62.5 = 0.944 (P(¬ M|B,L) = 1 – P(M|B,L))

  P(M|B,¬ L) = 1/30 = 0.033 (P(¬ M|B,¬ L) = 1 – P(M|B,¬ L))

  P(M|¬ B,L) = 1.5/5.5 = 0.273 (P(¬ M|¬ B,L) = 1 – P(M|¬ B,L))

  P(M|¬ B,¬ L) = 0/2 = 0.000 (P(¬ M|¬ B,¬ L) = 1 – P(M|¬ B,¬ L))
- P(G|B) = 85.5/92.5 = 0.924 (P(¬ G|B) = 1 – P(G|B))

  P(G|¬ B) = 0/7.5 = 0.000 (P(¬ G|¬ B) = 1 – P(G|¬ B))

ตัวอย่างของอัลกอริทึม EM (7)

(2) ใช้ CPTs เพื่อคำนวณน้ำหนักของตัวอย่างไม่รู้ค่าใหม่

- ในกรณีของ 7 ตัวอย่างแรก
  - P(G,M|B,L) = P(G|B)*P(M|B,L) = 0.924*0.944 = 0.872
  - P(G,¬ M|B,L) = P(G|B)*P(¬ M|B,L) = 0.924*0.056 = 0.052
  - P(¬ G,M|B,L) = P(¬ G|B)*P(M|B,L) = 0.076*0.944 = 0.072
  - P(¬ G,¬ M|B,L) = P(¬ G|B)*P(¬ M|B,L) = 0.076*0.056 =0.004

- ได้ตัวอย่างมีน้ำหนักเป็น

G	M	B	L	no. of instances
True	True	True	True	7*0.872=6.11
True	False	True	True	7*0.052=0.36
False	True	True	True	7*0.072=0.50
False	False	True	True	7*0.004=0.03

ตัวอย่างของอัลกอริทึม EM (8)

- ในกรณีของ 3 ตัวอย่าง
  - $P(B|\neg G,M,L) = \frac{0.076*0.652*0.925}{0.145*0.618} = 0.510$
  - $P(\neg B|\neg G,M,L) = 1 - 0.510 = 0.490$
- แสดงว่า 3 ตัวอย่าง เราสามารถใส่น้ำหนักให้เป็นตัวอย่างดังต่อไปนี้

G	M	B	L	no. of instances
False	True	True	True	3*0.510=1.53
False	True	False	True	3*0.490=1.47

- เมื่อทำซ้ำขั้นตอน (2), (3) จนครบ 5 รอบ CPTs ลู่เข้าดังนี้

$P(L) = 0.680$ $P(B) = 0.919$

$P(M|B,L) = 0.999$ $P(M|B,\neg L) = 0.033$

$P(M|\neg B,L) = 0.344$ $P(M|\neg B,\neg L) = 0.000$

$P(G|B) = 0.966$ $P(G|\neg B) = 0/7.5 = 0.000$