人工知能覚え書

furuya 2008.3

人工知能覚え書

目次

人工知能とは

人工知能とは、コンピュータで実現する知的処理。

コンピュータ内の知識の分類（知識表現）

（1）規則(rule)的知識
If 赤　then 止まれ
If 青＆左折＆通行人あり　then 待て
If 青＆右折＆直進車あり＆交差点内　then 待て
２桁以上の掛け算、ルールに従って筆算を行う

（2）暗記的知識、図書館的知識
例：　頭に入る簡単なものでは、　鎌倉時代は 1192、円周率は 3.14 、3.3 が 9。
　　頭に入らない難しいものでは、　エジプトの第３王朝？、アンドラって国？
（人間は、おぼえらる物は丸暗記、覚えきれない時本にしたり規則化する）

（3）論理(推論)的知識（３段論法）
人は食べる
石塚は人である
-→故に石塚は食べる
推理小説

（1）規則（ルール）による知識表現

エキスパートシステとして最もよく用いられるプロダクションシステム（Production System）がその代表例で、人工知能の中で最も成功し普及している。

動機：人がやっていた知的処理をコンピュータに代行させる。
例：新幹線の運行管理、原発の管理、ネット検索、
利点：間違いがなくなる。高速判断。全貌が人間の頭に入りきれない大規模システム。製作者が去った後の管理。

プロダクションシステム
知識を If_Then_（　If　_条件部_　Then　_アクション部_　）　規則で表現
例：化学分析
If リトマス赤変　Then 酸
If リトマス青変　Then アルカリ
If 酸＋アンモニア→ 白煙　Then 塩酸

プロダクションシステム(エキスパートシステムの代名詞でもある)

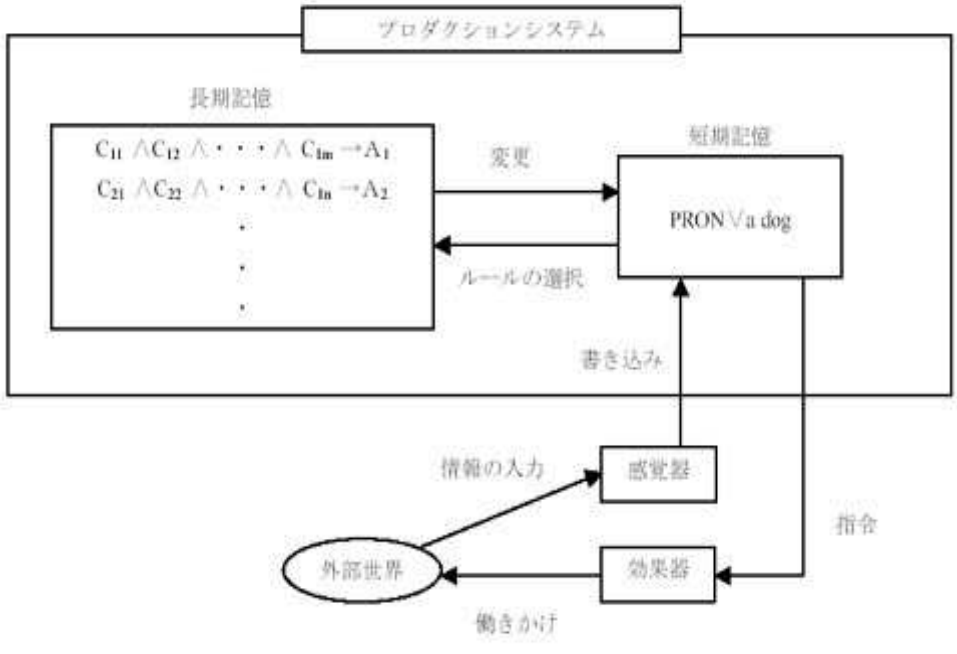
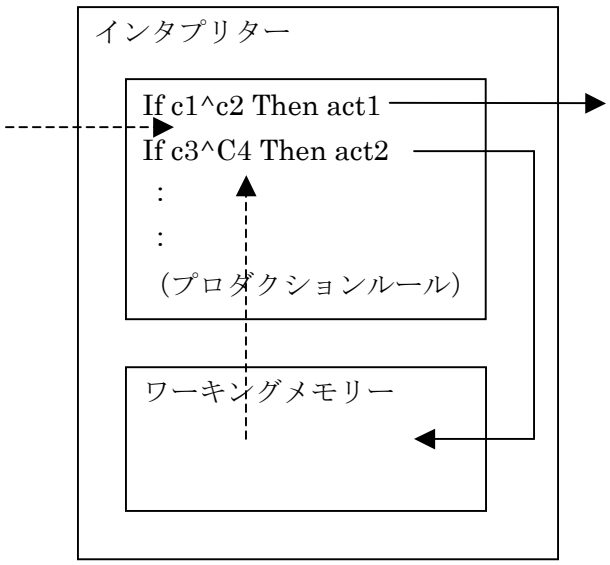
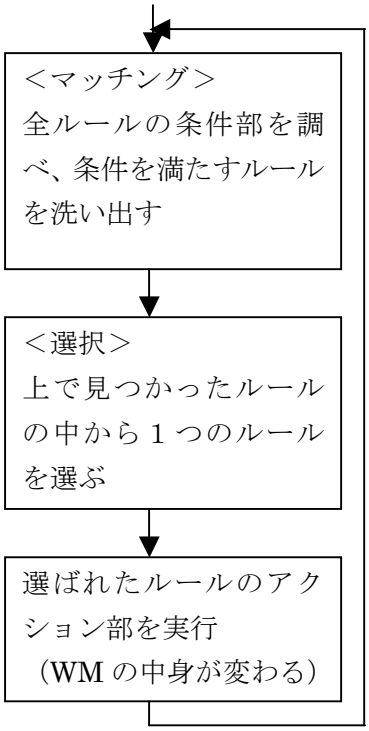


図1　プロダクションシステムの一般的な構成



プロダクションシステム

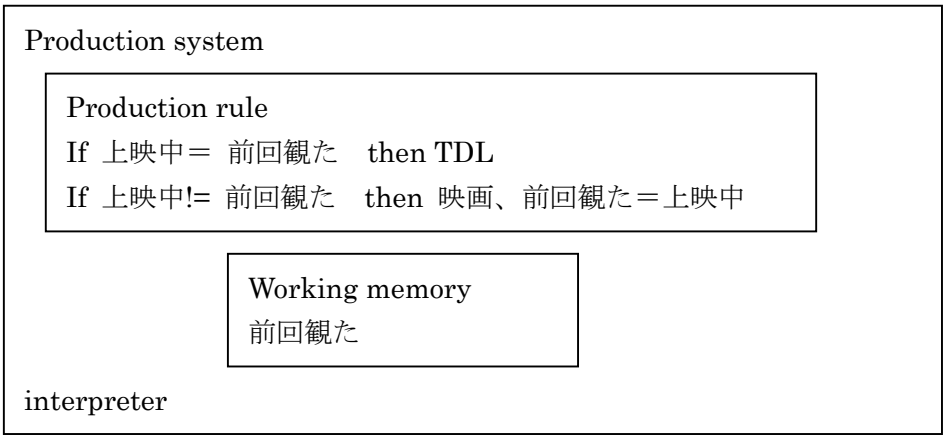


インタプリターの仕事

Production System の例

デート場所を決めるエキスパート

新しい映画をやっていたら映画を見に行き、前回と同じ映画ならディズニーランドへ行く。
現在上映中の映画名を入力して、何処へ行けばよいか決めてもらう。

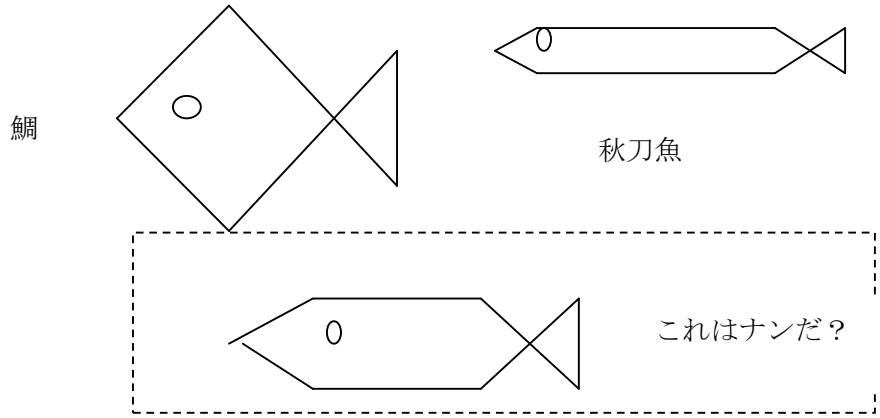


Vbscript によるプログラム

```
=======
Set Ag=Wscript.CreateObject("Agent.Control.1")
Ag.Connected=True
Ag.Characters.load "MSAg","Genie.acs"
set G=Ag.Characters("MSAg")
G.Show
G.MoveTo 200,200
mae_mita= "matrix"

For i=1 To 3
  G.Speak  "私は、デートの場所を決めるエキスパートです３回だけ相談に乗ります。いま、上映中の映画名を入力して下さい"
  joeityu=InputBox ("今上映中の映画名は？")

  If joeityu=mae_mita Then
    G.Speak ("上映中の映画は前に見た"&mae_mita&"なので今日はディズニーランドにしたら")
  End If
  If joeityu<>mae_mita Then
    G.Speak ("新しい映画やってるよ、、"&joeityu&"を見にいこう")
    mae_mita= joeityu
End If

  MsgBox "次の先相談"
Next
```

（２）　Fuzzy（曖昧）　System

予備知識１：Production System,　Fuzzy System,　Neural Network System でパターン認識をやった時の違い

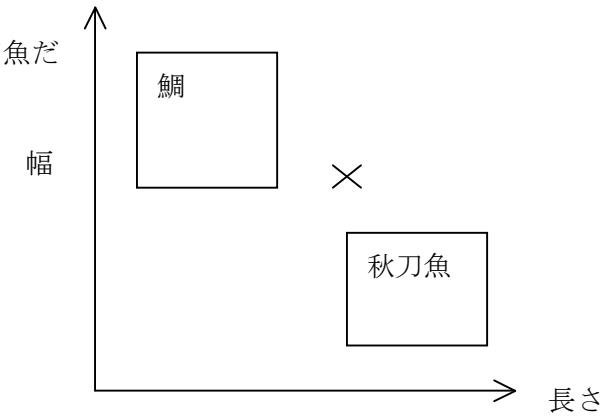


鯛と秋刀魚しか知らないシステムに新しい魚を見せた反応

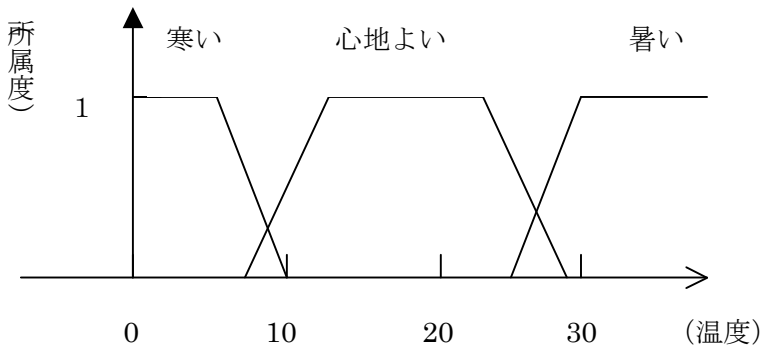
Production System 曰く：知らない

Fuzzy System 曰く：鯛のような秋刀魚だ

Neural Network System：秋刀魚だ



予備知識２：Fuzzy(曖昧さ)はメンバーシップ関数で表す

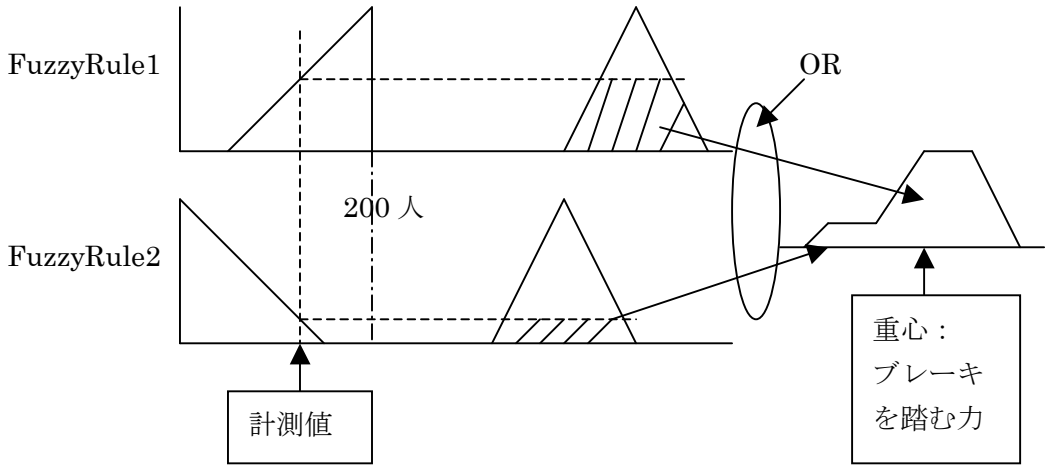


Menbership function とは、測定値が、あいまいな概念に属する割合を示す。

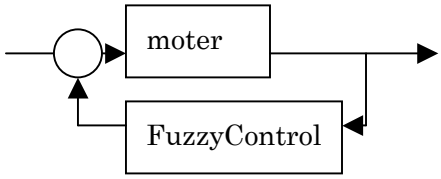
Production System と同様、知識を“If_条件_Then_行動_　ルール”の集合で表現。但し、

“条件”と“行動”に、曖昧な表現を数値化（メンバーシップ関数）したものを用いる。

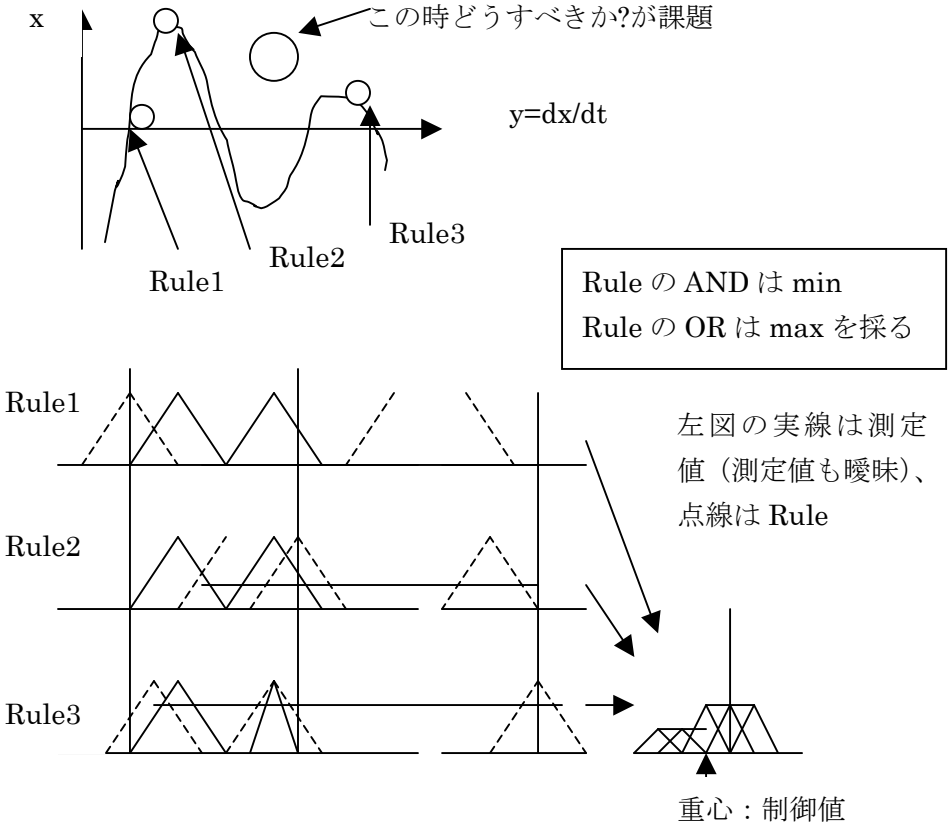
例１　（最も簡単な例）　Fuzzy 電車停止システム
Fuzzy Rule 1：　If　乗客多数　Then　ブレーキを強くかける
Fuzzy Rule 2：　If　乗客少数　Then　ブレーキを弱くかける



例２　（AND の入った例）　Fuzzy エアコン制御
x：温度、y：温度の時間変化分（dx/dt）、z：供給電力



Fuzzy Rule1:　If ((x is ZR) and (y is PL)) Then　z　is　NL
Fuzzy Rule2:　If ((x is PL) and (y is ZR)) Then　z　is　NM
Fuzzy Rule3:　If ((x is PS) and (y is NS)) Then　z　is　ZR
測定結果が　(x　is　PL)　and　(y　is　NS)　なら　z をどうすべきか？



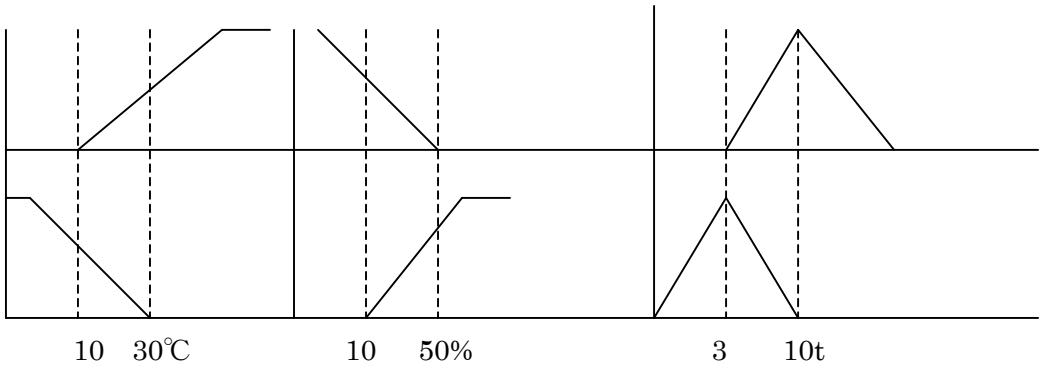
昨年の期末テストの問題

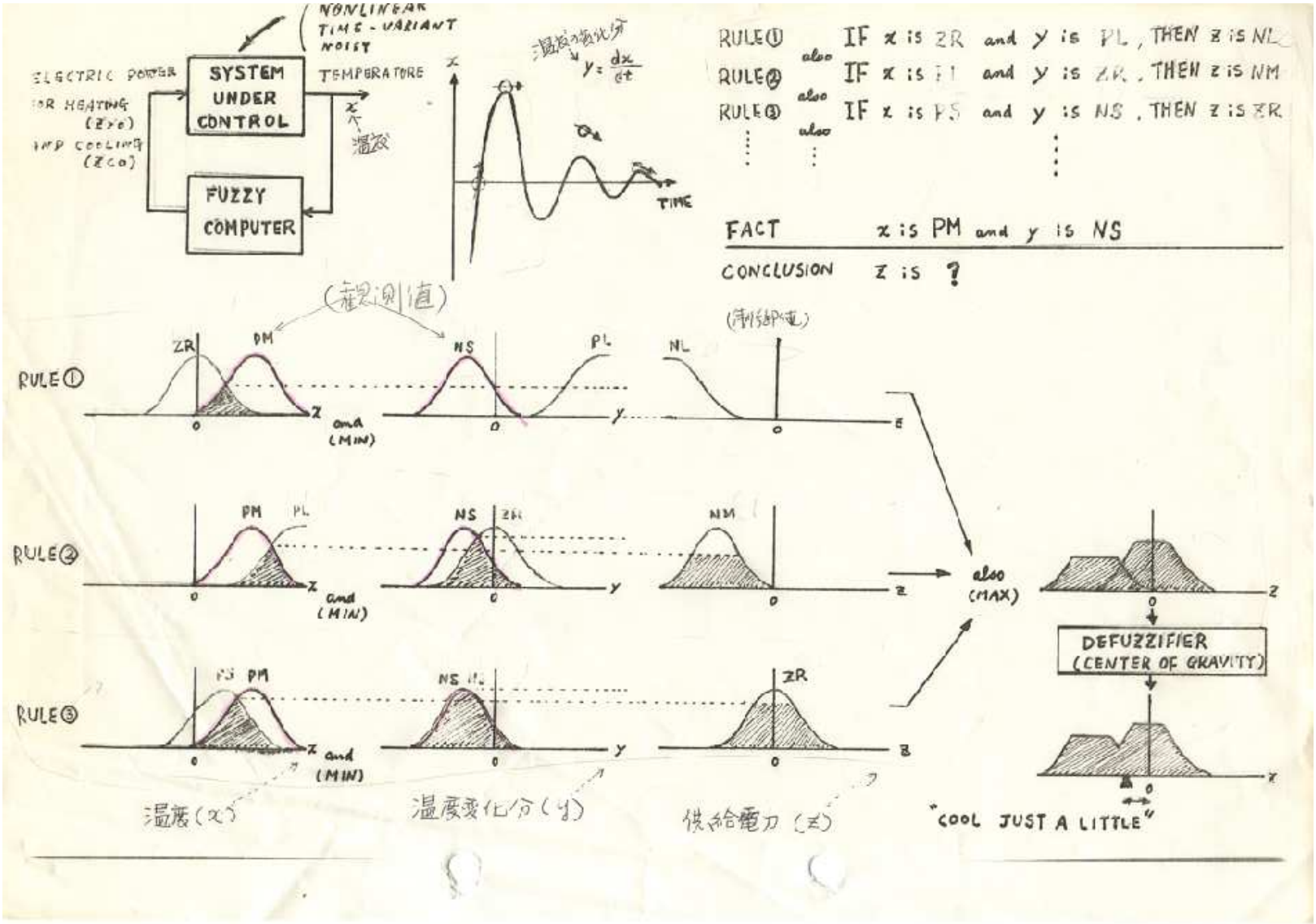
１．農場の散水機をファジー制御で自動運転させたい。ファジー規則は次の通りである。

（１） 気温が高く（約３０度以上）“かつ(and)”晴天（雲約１０％以下）なら水を沢山（１０トン前後）まく（下図参照）

（２） 気温が低（約１０度以下）“いか(or)”曇天（雲約５０％以上）なら水を少し（３トン前後）まく（下図参照）

気温が２５度、雲２０％の時は、どれくらい水をまくでしょう。




NONLINEAR
TIME-VARIANT
NOISY
ELECTRIC POWER
FOR HEATING
(Zxe)
AND COOLING
(Zco)
SYSTEM UNDER CONTROL
TEMPERATURE
FUZZY COMPUTER
TIME
RULE① IF x is ZR and y is PL, THEN z is NL
RULE② also IF x is PM and y is ZR, THEN z is NM
RULE③ also IF x is PS and y is NS, THEN z is ZR
FACT x is PM and y is NS
CONCLUSION z is ?
RULE①
RULE②
RULE③
and (MIN)
also (MAX)
DEFUZZIFIER (CENTER OF GRAVITY)
"COOL JUST A LITTLE"
温度(x)
温度変化分(y)
供給電力(z)

（3）知識をネットワークで表現

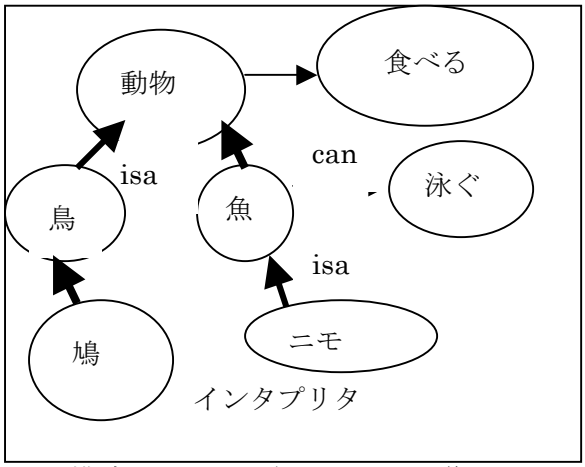
意味ネット（by Quillian）：

＊ 知識を概念間の関係（リンク）で表現

＊ isa リンクをたどって属性が継承する

＊ インタプリタが質問を受け、ネットをたどって答えを返す

＊ 右のネットに“ニモは泳げますか”と質問すると、ＹＥＳとなる。



フレーム：

ISA リンク付きの表と考えればよい。フレームの構造＝フレーム名＋スロット群

魚類フレーム

スロット名	value
類	魚
呼吸	エラ

各魚のフレーム　　ファセット

スロット名	value	ifneeded	ifadded	ifremoved
ISA	魚			
名前	マーリン			
誕生日	’01.5.1			
年齢		年齢計算		
子供	ニモ		フレーム追加	フレーム削除

フレームに陽に記されている（“value”に書かれている）ことに関する質問は表から答える。

陽に記されていない場合：

（1） ISA を辿る（例えば、マーリンはエラ呼吸か？といった質問には ISA リンクを辿る）

（2） ifneeded は“value”に書かれていない時質問されると年齢計算関数（デーモン）が呼ばれて計算する

（3） ifadded,ifremoved にはそれぞれ“value”が代入されたり削除されたりした時呼ばれる関数。上の例では、子供が生まれ、“ニモ”と書き込まれると ifadded という関数が動き出し、ニモのフレームを作る、と仮定している。

（４） 論理型言語 Prolog プログラム

知識を論理（ロジック）で表現する：(コンピュータによる創造の雰囲気を感じられる)
１．論理型言語 Prolog のプログラム構造と、それに対する質問
プログラム構造＝事実（データベース）＋　規則；　（注：規則は、なくてもよい）
例
（事実（データベース））
like(hanako, movie).　　「花子は映画が好き」
like(taro, football).　　「太郎はフットボールが好き」
like(taro, book).　　「太郎は本が好き」
like(ai, book).　　「アイは本が好き」
love(ai, taro).　　「アイは太郎を愛する」
（規則）
like(hanako, X) :- like(taro, X).　「太郎が好きなものは花子も好き」
like(M, T) :- love(M, N), like(N,T).　「愛する人が好きなものは全て好き」

以上のプログラムに質問を出す。
質問のタイプは３つある。
（質問タイプ１：事実に対する質問）
|?- like(taro, book).　　「太郎は本が好きですか？」
yes　　(prolog が返してくる答え)
|?- like(taro, sumo).　　「太郎は相撲が好きですか？」
no　　(prolog が返してくる答え)
（質問タイプ２：変数（大文字）を使う質問）
|?-like(taro, X).　　「太郎は何が好きですか？」
X=football　　(prolog が返してくる答え)
（注：enter を打つと次の質問に移るが、次の答えを欲しいときは；を打つ）
X=football の後で；を打つと X=book と答えてくる。
|?-like(taro, X), like(ai, X).　　「太郎もアイも好きなものは何ですか？」
X=book　　(prolog が返してくる答え)
（質問タイプ３：見かけは１，２と同じだが、解を得るため陰で規則が使われている質問）
|?-like(ai,football).　　「アイはフットボールが好きですか？」
yes（prolog は yes を返すために“愛する人が好きなものは、みんな好き”という規則を使っている）

２．prolog はどうやって答えを出すのか？
(1)事実だけで答えが出る場合：単に事実を探す。
(2)変数が有る場合：変数への穴埋めのイメージで事実のあてはめをやる。例えば、
like(taro, X). なら、X に football と book が入ることがすぐ分かる。
(3) like(taro, X), like(ai, X) の場合だと、まず、like(taro,X)の X にはまる football を見つけ、その X(football)で like(ai,football)が真か否かを調べる。これは、真ではないので football は捨てられ、次の X 候補 book を調べる。すると、like(taro,book),like(ai,book)は共に真なので、X=book となる。
(4)prolog は規則を使って質問を変形してまで答えを求めてくれる。
例えば、|?-like(ai,football).では、この質問を
|?-love(ai, N), like(N, football). と置き換えて答えを探すと N に taro がはまるので、yes となる。

３．Prolog プログラムヒント集
（１）家系
```
father(gonbe, chichiro).
father(chichiro, ichiro).
father(sasuke, haharo).
mother(haharo, ichiro).
grandpa(X,Y):-father(X,Z),father(Z,Y).
grandpa(X,Y):-father(X,Z),mother(Z,Y).
```
（２）大小関係
```
gt(cat, mouse).
gt(pig, cat).
gt(X,Z):-gt(X,Y),gt(Y,Z).
```
（３）c(…)の c は制約述語と呼ばれる計算
```
ha(a, 3).
ha(b, 4).
ha(c, 5).
gt(X,Y):- ha(X,XV), ha(Y,YV),ha(W,WV),c(XV > YV).
```
（４）階乗計算
```
k(0,1).
k(X,Y):- Z is X-1, Z>=0, k(Z,W), Y is X*W.
```
ここで、
|?-k(5,Y). なんて聞くと答えが返ってくる。

（５）英作文 [..,..]はリストを表す
```
syugo(haruko).
```

```
syugo(natsuko).
syugo(akio).
jidousi(smiles).
jidousi(laughs).
tadousi(plays).
tadousi(likes).
jutubu(V):- jidousi(V).
jutubu([V,O]):-tadousi(V),mokutekigo(O).
bun([S,P]):-syugo(S),jutubu(P).
```

ここで、

|?-bun([S,P]). なんて聞くと、文法規則上考えられる文が（；で）次々と返ってくる。

（６）俳句作り

```
jisu(furuike,4).
jisu(kawazu,3).
jisu(mizunooto,5).
jisu(tobikomu,4).
meisi(furuike).
meisi(kawazu).
meisi(mizunooto).
dousi(tobikomu).
josi(ya).
kaminoku(K):-meisi(K),jisu(K,5).
nakanoku([N1,N2]):-meisi(N1),jisu(N1,Y),dousi(N2,Z),jisu(N2,Z),c(Y+Z=7).
simonoku(S):-meisi(S),jisu(S,5).
haiku([K,N,S]):-kaminoku(K),nakanoku(N),simonoku(S),not(K=S).
```

（７）答えの重複を消す

```
oya(o1,a).
oya(o2,b).
oya(o2,c).
oya(g,o1).
oya(g,o2).
kyodai(X,Y):-oya(O,X),oya(O,Y), X ¥==Y.        （¥==は Bprolog ではないかも）
itoko(X,Y):-kyodai(O1,O2),oya(O1,X),oya(O2,Y).
```

（８）日英翻訳プログラム

```
es(jack).
es(betty).
js(hana).
js(taro).
eji(smile).
jji(warau).
eta(like).
```

```
jta(suki).
em(book).
jm(hon).
dic(book,hon).
dic(like,suki).
dic(jack,taro).
dic(betty,hana).
dic(smile,warau).
ejutubu(V):-eji(V).
ejutubu([V,O]):-eta(V),em(O).
ebun([S,P]):-es(S),ejutubu(P).
jjutubu(V):=jji(V).
jjutubu([O,V]):-jm(O),jta(V).
jbun([S,P]):-js(S),jjutubu(P).
yakus(ES,JS):-es(ES),js(JS),dic(ES,JS).
yakuj(EP,JP):-eji(EP),jji(JP),dic(EP,JP).
yakuj([EV,EO],[JO,JV]):-eta(EV),em(EO),jta(JV),jm(JO),dic(EV,JV),dic(EO,JO).
```

Ejyaku

＝＝＝

SWI-Prolog for MS-Windows

Ａ：　例

```
%% Demo Prog
%% Step1: Start メニュー->全てのプログラムから SWI-Prolog を選ぶ
%% Step2: 下のような PrologProgram（～.pl）を作る。
%%        この例は、demo の中の likes.pl というファイル。
%%        （File->New で作ってもよいし、エディター作ってもよい。）
%% Step3: プログラムをロード&コンサルト
%%        ?- [swi('demo/likes')].か、Flle->Consult（最後に.を加える）か、
%%        プログラムアイコンのWクリック。
%% 次のような質問が可能になる。
%% ?- likes(sam,dahl).
%% ?- likes(sam,chop_suey).
%% ?- likes(sam,pizza).
%% ?- likes(sam,chips).
%% ?- likes(sam,curry).

likes(sam,Food) :-
        indian(Food),
        mild(Food).
likes(sam,Food) :-
        chinese(Food).
likes(sam,Food) :-
        italian(Food).
likes(sam,chips).

indian(curry).
indian(dahl).
indian(tandoori).
indian(kurma).
```

```
mild(dahl).
mild(tandoori).
mild(kurma).

chinese(chow_mein).
chinese(chop_suey).
chinese(sweet_and_sour).

italian(pizza).
italian(spaghetti).
```

Ｂ：演算子と比較

演算子：　＋、－、＊、/、//（整数割り算）、X　mod Y、abs(X)。
算術比較：＞、＜、＞＝、＝＜、＝：＝（等しい）、＝¥＝（等しくない）。
項の比較：　＝＝（同一）、¥＝＝（同一でない）。

```
oya(o1,a).
oya(o2,b).
oya(o2,c).
kyodai(X,Y):-oya(O,X),oya(O,Y), X ¥==Y.
```
（最後の項が無いと自分も出てくる）

Ｃ：Bprolog との違い
（Bprolog の制約述語 c は、使わず、直接書く。加えて５になる文字を出すには）

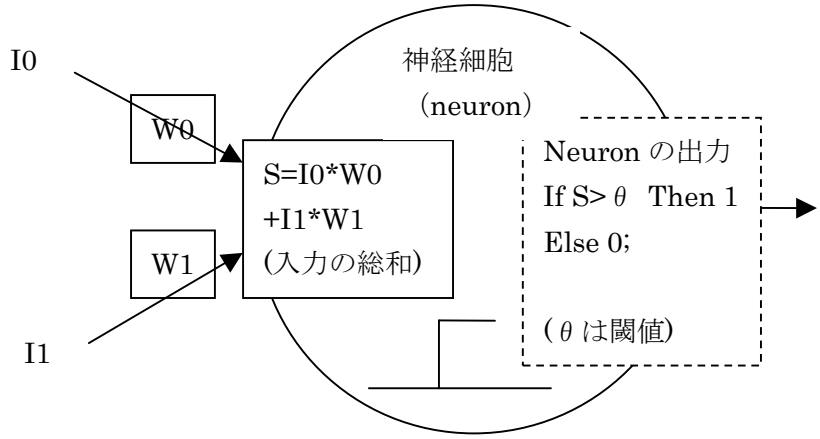
```
m(o1,1).
m(o2,2).
m(o3,3).
m(o4,4).
k5(X,Y):-m(X,M),m(Y,N), M+N =:= 5.
```

Ｄ：　is は代入
階乗計算は、

```
fact(0, 1).
fact(N, F) :- N1 is N-1,　fact(N1, F1),　F is N*F1.
```
で、fact(3,X).などと質問する。

（５）ニューラルネットと学習

人の脳のまねをすれば、柔軟な情報処理ができるんじゃない？って発想。



入力の総和が閾値を超えると発火して興奮が次に送られる(出力される)(神経は切れるヤツ)。

ニューラルネットの（教師あり）学習機能

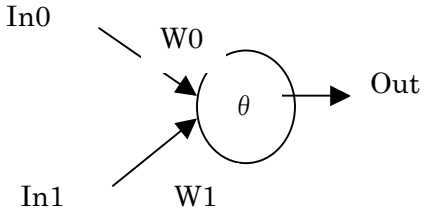
学習（入出力）パターンを提示し、入力が与えられたら所定の出力を出すように重みと閾値を調整していく。

例えば下のニューラルネットに（I0 と I1 の）ＡＮＤを学習させるには、教師データ（training data）は下の表になる。

In0	In1	Out
0	1	0
1	1	1
0	0	0
1	0	0

教師パターン０

教師パターン３

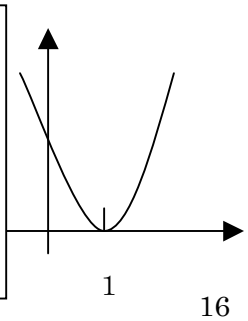


学習手順は誤差逆伝播（backpropagation）と呼ばれるもので、教師入力パターン０を入れては教師出力が出るように W0,W1, θを微調整し、次に教師入力パターン１を入れては正しい出力が出るように W1,W2, θを微調整し、、、、ってのを根気よく繰り返していきます。教師パターン０を覚えたら１に進むのではなく、０，１，２，３，０，１，２，３、、、って（英数国英数国と）繰り返し学びます。

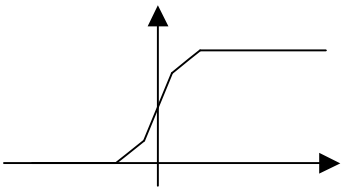
ここで、数学的準備

(1)　Ｅ＝ｆ（ｗ）を考える：Ｅがｗの関数の時、
Ｅを最小（極小）に導くにはｗを
$-\partial E/\partial w$　の方向に少し移動する。

$y=(x-1)^2$
$y'=2x-2$
を考えれば、
x=2 の時は
左に行くとよい

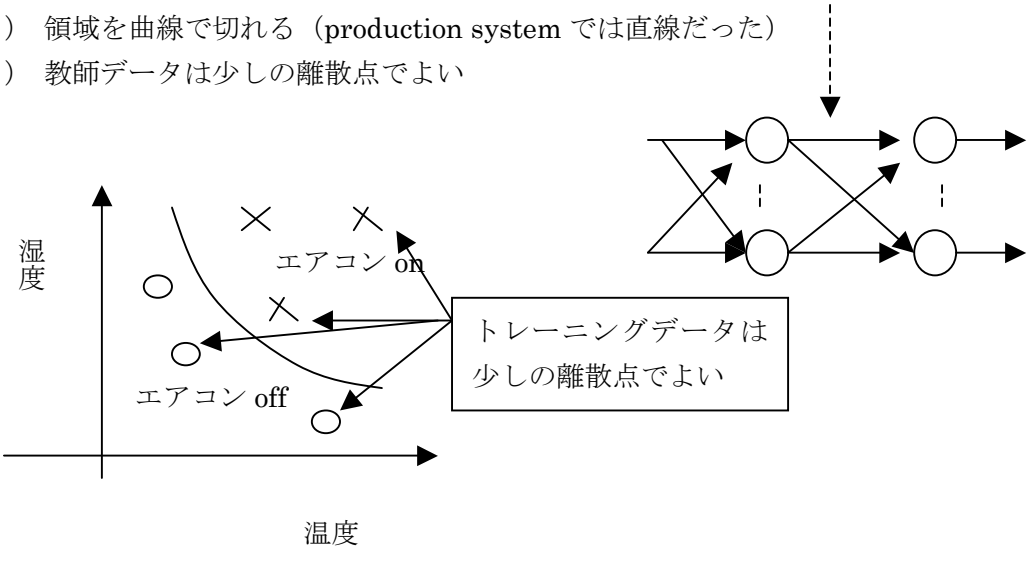


（最急降下法）



(2)　$y=1/(1+e^{-x})$ の微分は　$y'=y(1-y)$　である。
（切れる階段関数の変わりに、微分可能な
右のシグモイド関数を使う）
(3)　$\partial E/\partial w = (\partial E/\partial x) \cdot (\partial x/\partial w)$　である。

ＮＮのＢＰ学習
教師出力をｙ（正解)、教師入力を入れた時出てしまう出力を out とする。学習の目的は正解 y と出てしまう出力 out の差を０に近づけることである。式で書くと、自乗誤差Ｅ
Ｅ＝　Σ（y-out）²　を最小にしたい（心の中でＥはｗの関数だから最急降下法と思う）
（このΣは全教師パターンに対してっていうΣ)。
じゃあ、教師入力０を与えて、w_0、w_1をどう変えるかって話。最急降下法で考えれば、
$-\partial E/\partial w_0 = -(\partial E/\partial S) \cdot (\partial S/\partial w_1)$　　---(式１)
ここで、$S = w_0 * I_0 + w_1 * I_1$
式１は、$-\partial E/\partial w_0 = -(\partial E/\partial out) \cdot (\partial out/\partial S) \cdot (\partial S/\partial w_0)$　（式２）となる。
ここで、　$out=1/(1-e^{-S})$　である。となると式２は、
$-\partial E/\partial w_0 = 2 \cdot (y - out) \cdot (out * (1 - out)) \cdot (I_0)$　ってなるよね？
w_0はこの値（$-\partial E/\partial w_0$）に比例した小さな値だけ変化させる。
これをw_1、θについても同じように変更し、それを全教師パターンに関して繰り返していくと、やがて、ニューラルネットは全ての入力に正しく答えられるようになる。すなわち学習が終わる。
注１）　閾値は上のシグモイド曲線の左右へのシフトと考える
注２）　学習率（learning_rate）は、どの位の速度で重みを変化させるかであり、大きすぎると値が変化しすぎ、小さすぎると学習がなかなか進まない。
注３）　上の例は最も簡単なネットだが、普通は次の多層ネットを用いる
注４）　領域を曲線で切れる（production system では直線だった）
注５）　教師データは少しの離散点でよい



（６）ベイジアン（Bayesian Network）ネット

確率のネットあるいは、風が吹けば桶屋が儲かる？

１、確率の復習

標本空間　Ω：　起こりうる全ての集合。

事象 A　：標本空間の部分集合。

確率　P(A)：A の起こる確率。

例１：さいころ

Ω＝｛1,2,3,4,5,6｝

P({1})=P({2})=・・・ =P({6})= 1/6

例２：天気

Ω＝｛晴、曇、雨｝

P(晴)=0.6、P(曇)＝0.3、　P(雨)＝0.1

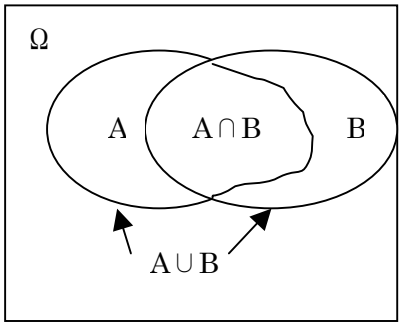


図１

２、条件付確率

事象Aが起こっている時、事象Bが起こる確率、を条件付確率と言う

$$P_A(B) = P(B \mid A) = \frac{P(A \cap B)}{P(A)}$$

確率の積の公式

$$P(A \cap B) = P(A \mid B)P(B)$$

３、ベイズの法則

積の公式より、

$$P(A \cap B) = P(A \mid B)P(B)$$

$$P(A \cap B) = P(B \mid A)P(A)$$

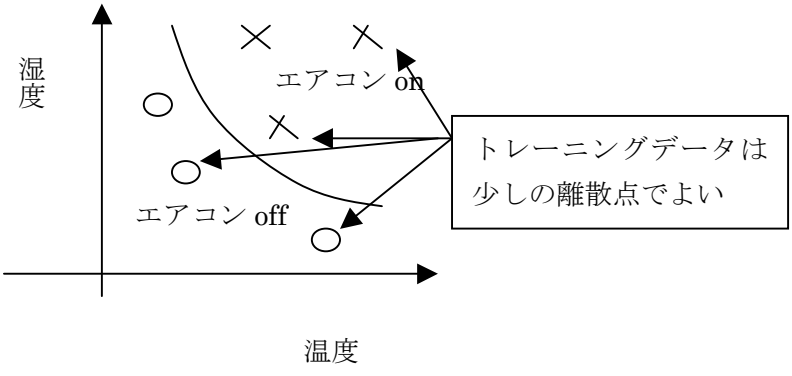
この２式より

$$P(B \mid A)P(A) = P(A \mid B)P(B)$$

両辺を $P(A)$ で割って、

$$P(B \mid A) = \frac{P(A \mid B)P(B)}{P(A)}$$　　（この関係式をベイズの法則と言う）

この式から、事前確率２つ（P(A),P(B)）と条件付確率１つ（P(A|B)）から、条件付確率１つ（P(B|A)）を求められることが分かる。



４、ベイジアンネット

２つの事象Ａ，Ｂに、次のような確率があったとする。

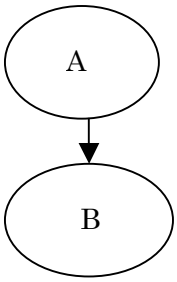
$P(A) = 0.3$

$P(B \mid A) = 0.1$

$P(B \mid \overline{A}) = 0.4$

例えば、Ａは晴、Ｂは風、と考えれば、晴に風が吹く確率は 0.1、晴ていないとき風が吹く確率は 0.4 って感じ。

この関係を、図で表したのがベイジアンネットである。



	A が起こった時	A が起こらないとき
Ｂが起こる	0.1	0.4
Ｂが起こらない	0.9	0.6

右の表は、左図の矢印に沿った関係を示している。が、ベイズの公式を使うと矢印を遡る関係（P(B)や P(A|B)）を求めることができる。

$$P(B) = P(B \mid A)P(A) + P(B \mid \overline{A})P(\overline{A}) = 0.1 \times 0.3 + 0.4 \times 0.7 \cdots (式\ 1)$$

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} = \frac{0.1 \times 0.3}{0.31} = 0.096$$

５、もう少し複雑な例

熱、咳、風邪の関係をベイズネットで表現

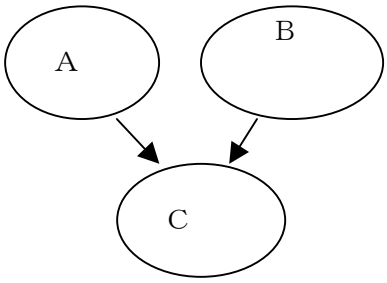
Ａ：熱がある、

Ｂ：咳が出る、

Ｃ：風邪、

P(A)=0.2

P(B)=0.1



右の表は、矢印に沿った関係を示している。

ベイズの公式を使うと矢印を遡れる。

C が真の時Ａが真の確率

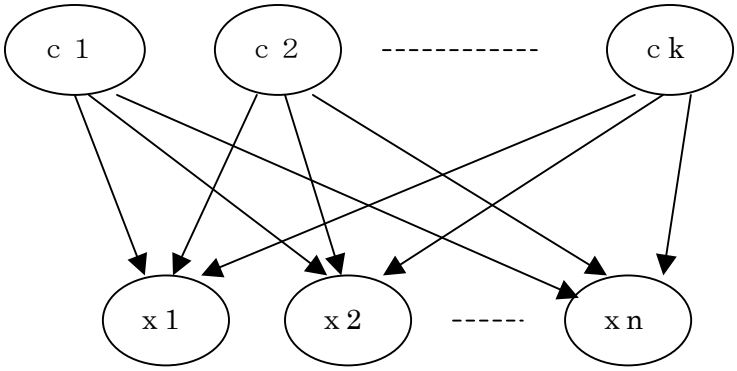
$$P(A \mid C) = \frac{P(C \mid A)P(A)}{P(C)}$$

$$P(C \mid A) = P(C \mid A, B)P(B) + P(C \mid A, \overline{B})P(\overline{B}) \cdots (式２)$$

$$P(C) = P(C \mid A, B)P(A)P(B) + P(C \mid A, \overline{B})P(A)P(\overline{B})$$
$$+ P(C \mid \overline{A}, B)P(\overline{A})P(B) + P(C \mid \overline{A}, \overline{B})P(\overline{A})P(\overline{B})$$

(A,B	(T,T	(T,F	(F,T	(F,F
C=T	0.8	0.5	0.4	0.1
C=F	0.2	0.5	0.6	0.9

６．ベイズの分類



ｃは分類
(例えば、
麻疹、風疹)

ｘは特徴
(例えば、
熱、咳)

ベイズの法則の｜の右側をｘのベクトル$\vec{x}$と見ると式10が出る、

$$P(C=c_k \mid \vec{X}=\vec{x})=\frac{P(\vec{X}=\vec{x} \mid C=c_k)P(C=c_k)}{P(\vec{X}=\vec{x})} \qquad (式10)$$

$$P(\vec{X}=\vec{x} \mid C=c_k)=\prod_i P(X_i=x_i \mid C=c_k) \qquad (式11)$$

式 11 は、図の矢印に沿った方向なので、ベイズネットの確率の表から計算できる（左辺の意味を考えれば分かる)。

式10の右辺の分母は、特定のメール$P(\vec{X}=\vec{x})$が来る確率。特定のメールとは、

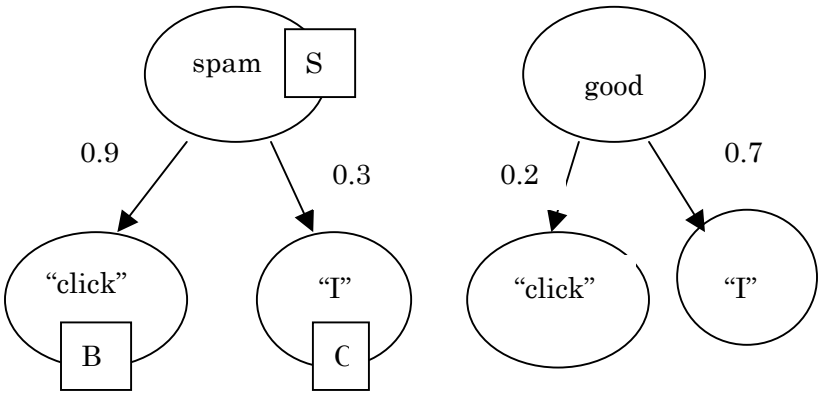
”click”と”I”があるメールはx1=’1’,x2=’1’、
”click”があって”I”がないメールはx1=’1’,x2=’0’、
”click”がなくて”I”があるメールはx1=’0’,x2=’1’、
”click”も”I”もないメールはx1=’0’,x2=’0’、

$P(\vec{X}=\vec{x})$

=ｃ１が起こる確率＊ｃ１が起こったとき特定の$\vec{x}$（例えばx1=1,x2=0）が起こる確率
+ｃ２が起こる確率＊ｃ１が起こったとき特定の$\vec{x}$（例えばx1=1,x2=0）が起こる確率
+…
+ｃｋが起こる確率＊ｃ１が起こったとき特定の$\vec{x}$（例えばx1=1,x2=0）が起こる確率
となる。
この右辺の各項は、ベイズネットの確率表から求めることができる。
でも、これ（右辺の各項）って式10の分子と同じ。

それでは、ってことで、麻疹、百日咳、風疹、の例で考えると。
“熱あり”“咳なし”“発疹なし”の場合、
P(x1=1,x2=0,x3=0)
＝P(麻疹)＊P(x1=1,x2=0,x3=0|麻疹)
＋P(百日)＊P(x1=1,x2=0,x3=0|百日)
＋P(風疹)＊P(x1=1,x2=0,x3=0|風疹)
＝P(麻疹)＊P(x1=1|麻疹)＊P(x2=0|麻疹)＊P(x3=0|麻疹)
＋...
＋...
って具合。

例：スパムメール発見
メール全体の内、spam
メールの割合は 0.1 と
仮定すると。



メールが来て”click”と”I”が入って
いる確率は、

$$P(E) = P(B \mid S)P(C \mid S)P(S)$$
$$+ P(B \mid \bar{S})P(C \mid \bar{S})P(\bar{S})$$

"*click*"と"*I*"が入っている時
スパムの確率

$$P(S \mid E) = \frac{P(B \mid S)P(C \mid S)P(S)}{P(E)}$$

スパムでない確率

$$P(\bar{S} \mid E) = \frac{P(B \mid \bar{S})P(C \mid \bar{S})P(\bar{S})}{P(E)}$$

	S(T)	S(F)
B_T	0.9	0.2
B_F	0.1	0.8
C_T	0.3	0.7
C_F	0.7	0.3

このような計算を、good メールに対しても行い、確率の高い方を選ぶ。

EXCEL で人工知能

以下の病気が発症する確率は、はしか 0.2、百日咳 0.1、風疹 0.15 とする。

熱があり、咳がなく、発疹がない時の各病気の確率を EXCEL で求めよう。

	はしか	百日咳	風疹
熱	0.8	0.1	0.6
咳	0.4	0.9	0.1
発疹	0.7	0.1	0.6

付録： 式２の導出

（１） 確認： 式２以降に出てくる P(C|A,B)は、A と B の AND が成り立った時 C が起こる確率を表す。分かりやすく書くと P(C|(A∩B))となる。以下では P(C|A∩B)と書く。

（２） 準備１： 式１の確認

$$P(B) = P(B \mid A)P(A) + P(B \mid \overline{A})P(\overline{A}) = P(B \cap A) + P(B \cap \overline{A}) \cdots (式３)$$

ベイズの法則で上のようになる。この最右辺を図１に照らし合わせれば納得できると思う。

（３） 準備２： P(B∩C∩A)の変換

A∩B を組にして考えてベイズの法則を適用すると、

P(B∩C∩A)= P(A∩B)・P(C|A∩B)‥‥（式４）

この右辺の P(A∩B)をベイスの法則で展開すると、P(A∩B)=P(B)・P(A|B)なので、（式４）は、

P(B∩C∩A)= P(B)・P(A|B)・P(C|A∩B) ‥‥（式５）

となる。

（４） これより本番

式３のアナロジーで、

$$P(C \cap A) = P((B \cap C \cap A) \cup (\overline{B} \cap C \cap A))$$

$$= P(B \cap C \cap A) + P(\overline{B} \cap C \cap A)$$

(ここで式５を用いると、)

$$= P(B) \cdot P(A \mid B) \cdot P(C \mid A \cap B) + P(\overline{B}) \cdot P(A \mid \overline{B}) \cdot P(C \mid A \cap \overline{B}) \cdots (式6)$$

ＡとＢは独立なので、$P(A \mid B)$と$P(A \mid \overline{B})$は、いずれも P(A)となる。

式６をさらに整理すると。

$$P(C \cap A) = P(A) \cdot P(C \mid A)$$
$$= P(B) \cdot P(A) \cdot P(C \mid A \cap B) + P(\overline{B}) \cdot P(A) \cdot P(C \mid A \cap \overline{B})$$

となる。この右の２辺の両辺を P(A)で割ると、式２がでてくる。

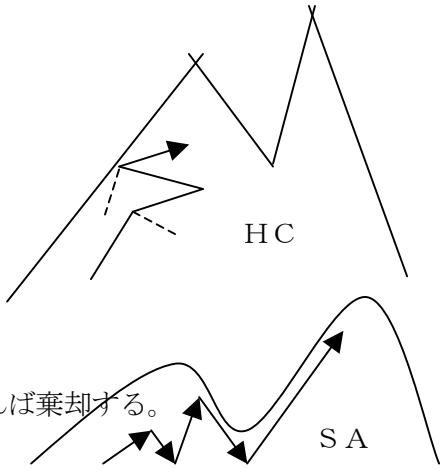
$$P(C \mid A) = P(B) \cdot P(C \mid A \cap B) + P(\overline{B}) \cdot P(C \mid A \cap \overline{B}) \cdots (\text{式２と同じ})$$

（７）組み合わせ最適化問題を解く

組み合わせ最適化問題：たくさんの組み合わせの中から最も望ましい組み合わせを選び出す問題のことである。一般に、組み合わせは膨大で、全てを調べ尽くすことはできない場合を対象にしている。

（１）山登り法（Hill Climbing）

現地点から足元だけを見て頂上を目指す



例：色々な成分を混ぜて薬品を作る場合を考えよう。

Step1:Ａを混ぜると少し効果があった、A

Step2:B を加えると少し効果が上がった、A.B

Step3:C を加えても効果がなかった、A.B

Step4:D を加えたら少し効果が上がった、A.B.D

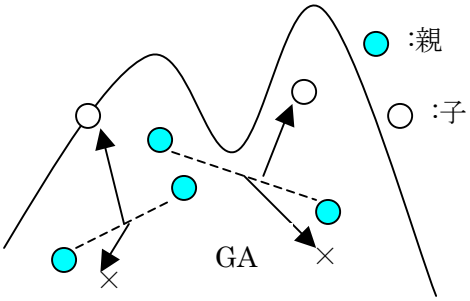
って感じで、よい効果が出たら採用し、効果が無ければ棄却する。

常に、よい方へ進むので、山を登っている感じがある。そこで、この方法を“山登り法”と呼ぶ。

この問題点は、小山に登ってしまうと大山の方へいけなくなる。

（２）焼き鈍し法（Simulated　Annealing）

基本的には山登りだが、最初の内は、時々低い方へも行ってみる。終わりに近づくほど完全な山登りになる。



（３）遺伝的アルゴリズム（GA: Genetic Algorithm）

GA は生物の進化の仕組み（交配、突然変異、淘汰）を模して問題を解く方法である。

イメージ的には右図の様に、複数の登山者が子供を作り（交配＋突然変異）、出来の良い子だけを生き残らせる（淘汰×）。

具体的には、問題の答の候補を（生物）個体（染色体）と考え、これを多数用意する、次にこれらの個体の交配、突然変異、淘汰を繰り返し、正解に近づける。

例題：ｘ＋ｙ＝５０、を満たすｘ、ｙを求めることを考える（これは組み合わせ最適化問題ではないが）。ｘ、ｙを要素とする１次元配列が１個体（ｘ、ｙは遺伝子）。

x	y

← １個体（染色体）

アルゴリズム：

ステップ１(初期個体の生成)：解の候補を（今回は）１０個用意する：２ｘ１０の整数配

列に乱数 rand()%100 を使って１００以下の整数を作成。

例えば

x	y
３１	１５
２２	６６
:	:
８１	４４

個体０（染色体 0）
個体１（染色体１）
：
個体９（染色体９）

のようなものができる。

ステップ２：上の個体（解の候補）群の中から２個体（父と母）を乱数で選ぶ。

ステップ３（交配）：選んだ個体から子供（父の左半分と母の右半分からなる）を作る。
（ここで２個体から３個体ができたことになる）
上の図で、父が個体０、母が個体１だとすると子は

３１	６６

ステップ４（個体の適応度評価）：３個体（父、母、子）について、正解にどれくらい近いかを調べる。
（ｘ＋ｙの値と３０との差の絶対値）を適応度とする（０に近いほどよい個体)。(絶対値の計算には abs()を使ってよい)

ステップ５（淘汰）：３個体（父、母、子）の適応度を求めた結果、子が親より適応度がよい（０に近い）時は、個体群から悪いほうの親を消し（上の２ｘ１０の配列から悪い親を消し)、消して空いたところに子を入れる。

ステップ６（突然変異）：乱数で１個体を選び、その遺伝子の一つ（ｘまたはｙ）を少し変える（乱数で＋１または－１する）

ステップ７（世代の繰り返し）：ステップ２へ戻る（この繰り返しを５００回くらいやると正解に近い解が残ってくるかも）

（８）強化学習（Reinforcement Learning）

試行錯誤を繰り返しながら（誰にも教わることなく）、現状（各状態）において、どういう行動（長期的視野に立ち）を採れば最大報酬が得られるかを学ぶ方法。但し、報酬は目的を達成した時のみ、得られる。成功しないと報酬が得られないので“飴と鞭の学習”とも言われる。すなわち、強化学習では、現在の自分の置かれている状況（状態）は既知とし、そこでどの行動を採るのがよいかを学習する。エージェントは行動を起こすことにより、次の状態に移る。もっと具体的例としては、カメラを持った自律ロボットがカメラからの情報を基にできるだけ短時間で目的地に到着するよう、各分岐点で進路を決定するための学習法と考えればよい。但し、ロボットが分岐点で判断を行いながら歩いている最中には教師の指示はなく、目的を達成した時のみご褒美がもらえる。

強化学習をもう少しフォーマルに整理すると、外部から与えられる成否情報を基に行う学習であり、成否情報は行動をした直後ではなく時間がたってから与えられる。

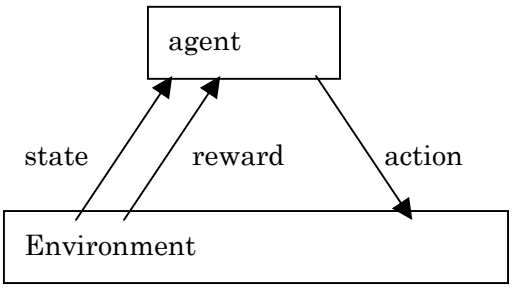
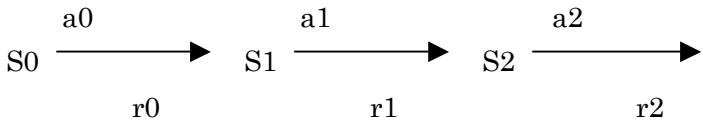


Figure 1 強化学習

エージェントは行動を行うと環境の状態が変化する。目的を達成すると環境から褒美がもらえる。

もう少し数式的に書くと、行動と状態遷移を書くと次のようになる。



この図は s0 状態にいた時に a0 という行動を行い r0 という報酬をもらい、s1 という状態に移るということ示している。強化学習の目標は

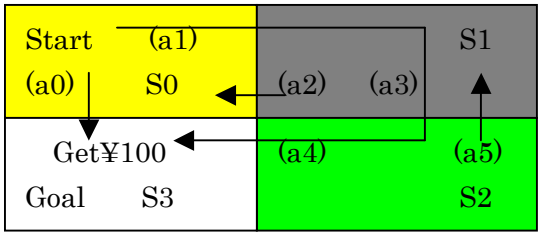
$$r0+\gamma r1+\gamma^2 r2+....,where 0<\gamma<1$$

を最大にする action を選ぶことである。

＊コメント：上の式は r0 が現在で、rn は未来。言い換えると、近い将来もらえる報酬を

大切にすることになり、できるだけ速くゴールに近づくことを促している。

簡単な例を考える。色の異なる 4 部屋があり、各部屋には隣の部屋へ入るドアがある。



Start 室から出発し、100 円もらえる goal 室へ向かう経路を学習する。あちこち歩くとエネルギーを使うので、できるだけ効率よく goal に到達する道を学習したい。S0 から直接下に行った時の報酬は上の式でいうと r0=100 で 100 円ゲットでき最大である。S0 から S1,S2,S3 と辿った場合は(ガンマを 0.9 とすると) r0+0.9*r1+0.81*r2 となるが r2=100,r0=0,r1=0（ゴールでしか報酬をもらえない）なので 81 円となり報酬が減る。

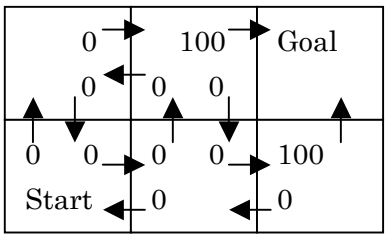
それでは、どうやったら最適経路を学習できるのか？

１．Ｑラーニング：各部屋の出口に黒板を用意し、そこを通るといくらもらえそうかを書いていく。

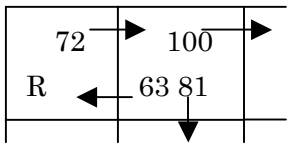
Ｑラーニングの前提：各状態（部屋）の出口ドアにＱ値を記した札がぶら下がっている。ロボットは次の部屋に入る際、先ず次の部屋に頭を突っ込み、次の部屋の出口にぶら下がっているＱ値を眺め、それを使って（元の部屋の）出口ドアのＱ値を計算し書き換えてから次の部屋に入る。

（もし、最後にしか御褒美をもらえない場合に、私達はどうするかを考えてみると、先ず褒美をもらった時の出口にしるしをつけ、次にその部屋へ入る出口に印をつけるという作業を繰り返すはず。Ｑラーニングの基本思想はまさにそれ）

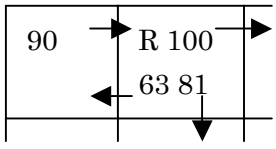
もう少し複雑な例：



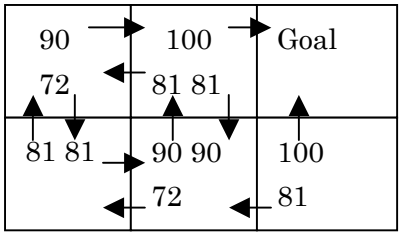
報酬表 r(s,a): s 状態で a を選んだ時の報酬



R 移動前のＱ値



R が右に移動後のＱ値



学習が終わった時のQ値

学習終了後はQ値の大きいほうに進めばよい。

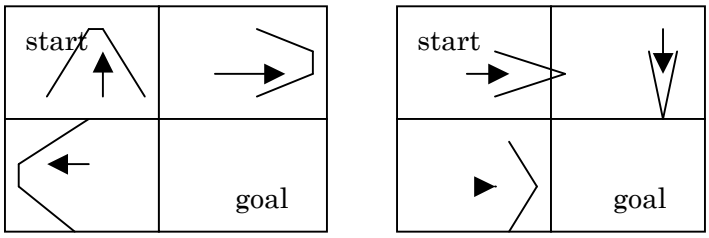
プログラム例：

```
#include <stdio.h>
#include <stdlib.h>
#define goal 2
double q[6][4];
double   gamma=0.9;
int now=0, before=0;
int status[6][4] =    /*    環境      （数は部屋番号）      */
      {{0,1,3,0},{1,2,4,0},{2,2,5,1},{0,4,3,3},{1,5,4,3},{0,0,0,0}};
  //次の状態を現在の状態とアクションから導く
          int nextStatus(){
               int action;
               while(before==now) {
                    action= rand()%4;
                    now= status[before][action];
               }
              return action;
          }
     //Q 値の変更
          void changeQ(int action){
                    int i;
                    double maxq, reward;
                    maxq=-999;
                    if(now==goal) reward=100.0;
                    else reward=0.0;
                    for(i=0; i<4; i++)
                         if(maxq< q[now][i]) maxq= q[now][i];
                    q[before][action]= reward + gamma*maxq;
                    return;
```

```
        }

    void report() {
        int i;
        for( i=0; i<6; i++)
            printf("%f,%f,%f,%f¥n",q[i][0],q[i][1],q[i][2],q[i][3]);
    }
/* メイン関数　*/
  void main() {
        int count=0, step=0, action;
        double reward=0.0;
        while(count<10) {
          while (now!=goal){
                before= now;
                action= nextStatus();
                changeQ(action);
            }
        count++; now=0; reward=0.0;
        }
        report();
  }
```

おまけ

２、Actor-Critic：出口ドアに連続した数値を割り当て（例えば a0=0,a1=1 って感じ)、どの方向に進むと有利かを、正規分布（平均が進むべき方向、分散がその方向の確からしさ）として記憶していく。



学習前　　学習後


は正規分布のつもり、頂上が進むべき方向

進む方向への数値割り当て：上＝０，右＝１，下＝２、左＝３。初期値をランダムにセッ

トすると進むべき方向がバラバラだが、やがて学習が進むと分散の小さな正規分布になり、進路が正確に示されるようになる。
学習の原理：Ｑラーニングで各部屋の最大Q値は、その部屋からゴールに達する最大報酬（最も効率よくゴールにいける）を示している。これ（部屋の最大Q）をＶと置く。エージェントは部屋 before(Ｖbefore)から部屋 now(Ｖnow)へ乱数の結果に従って移動する。Ｖbefore とＶnow の間にはＶbefore＝Ｖnow*γという関係がある。ということは、部屋 before で（乱数で）行き先を決めて移動してみて、Ｖnow*γ＋reward ＞ Ｖbefore の関係が成り立つ時、わざわざ移動（γ分のコストをかけて）して良い結果が得られたので before での乱数の示した方向は正しかった、ということになる。そこで、乱数の示した方向に正規分布の中心を少し移動する、という学習操作を繰り返す。同時に正規分布の分散も狭め、一方向に進みやすくする。
プログラム例（2x3 の迷路）（同志社大三木研究室の HP を参考にさせていただきました）

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define statenum 6
#define goal 5
#define defmu 2.0
#define defsigma 2.0
#define defval 0.0

double val[6]={0,0,0,0,0,0}, mu[6]={2,2,2,2,2,2}, sigma[6]={2,2,2,2,2,2};
int actnum=4;
double min=0, max=4;
double alpha=0.5, gamma=0.9, r, r1=100, r2=0;
double oldact;
int i,j,k;


/*    環境      （数は部屋番号）      */
int status[6][4] =
      {{0,1,3,0},{1,2,4,0},{2,2,5,1},{0,4,3,3},{1,5,4,3},{0,0,0,0}};

/*    乱数発生    */
double NumR() {      /*  0 ～ 1 一様乱数  */
          return (double)rand()/((double)RAND_MAX+1);
}
```

```
double BoxMuller(double mean, double var) {/*正規 mean:平均,var:分散*/
        double r1, r2, z1, z2;

        r1 = rand()/(double)RAND_MAX;
        r2 = rand()/(double)RAND_MAX;
        z1 = sqrt(-2.0*log(r1));
        z2 = sin(6.283185307179586*r2);
        return var*z1*z2 + mean;
        }

/* アクター */
        int Aaction(int state){
                double act;
                act = BoxMuller(mu[state],sigma[state]);
                oldact = act;
                if(act<min){
                        do{
                                act = act + actnum;
                        }while(act<min);
                }
                if(act>=max){
                        do{
                                act = act-actnum;
                        }while(act>=max);
                }
                return (int)act;
        }

    //次の状態を現在の状態とアクションから導く
        int nextStatus(int now,int action){
                int next;
                if(now == goal){ //***
                        next=0;
                }
                else{
                        next = status[now][action];
                }
```

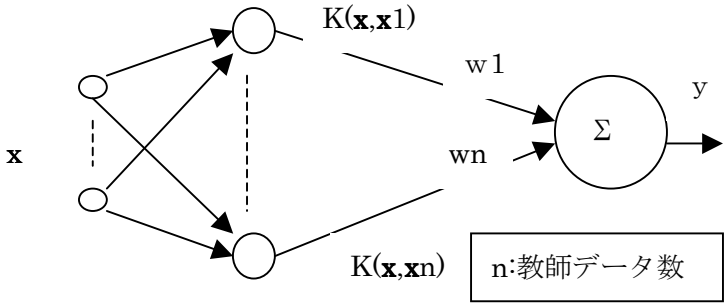
```
	return(next);
}


void Arenew(int before){
	sigma[before] = (sigma[before]+fabs(oldact-mu[before]))/2;
	mu[before] = (mu[before]+oldact)/2;
	if(fabs(oldact-mu[before])<sigma[before]){
	}
	if(oldact<mu[before]){
		do{
			mu[before] = mu[before] + actnum;
		}while(mu[before]<min);
	}
	if(mu[before]>=max){
		do{
			mu[before] = mu[before]-actnum;
		}while(mu[before]>=max);
	}
}


void Grenew(int now,int before){
	double tderror;
	if(now == goal) r = 100.0;
	else r = 0.0;
	//TD-error を求める
	tderror = r+gamma*val[now]-val[before];
	//状態評価値の更新
	val[before] = val[before]+alpha*tderror;
	//TD 誤差を返す
	if(tderror>0){
		Arenew(before);
	}
}

void report() {
int i;
```

```
            for(i=0; i<statenum; i++) {
                printf("%d:%f==%f_%f¥n",i,val[i],mu[i],sigma[i]);
            }
        }

/*  メイン関数    */
    void main() {
        int before=0, now=0;
        int count=0, nowt, ran;
        report();
        while(count<3000) {
            while (now != goal){
                before=now;
                nowt= Aaction(before);
                now=nextStatus(now, nowt);
                Grenew(now,before);
            }
          count++;
          now=0;
        }
        report();
    }
終わり
```

（９）サポートベクターマシン（SVM）

原理は、栗田氏、津田氏、平井氏等のＨＰに詳しく解説されている。ここでは、“使い方”という立場から、排他的論理和（XOR）例を使って説明する。

SVM ネットワークの構成は次のようになっている。



ネットワークは、入力 **x**（図の x は一寸曲者でベクトル、m 入力なら(x1,x2,…,xm)）が中間ユニットに送られる。各中間ユニットは各教師データに対応して作成するもので、当然中間ユニット数は教師データの数と同じである。各中間ユニットの出力はK(**x**,**x**s)はスカラーである（s は s 番目の中間ユニットを表す）。中間ユニットの出力には重みｗs が掛けられ出力ユニットで加算されて、最終出力ｙが得られる。

式で書くと、$y=\sum_{i=1}^{n}\alpha_i y_i K(x,x_i)+b$　となる。

SVM の課題は中間ユニットの出力Ｋの求め方とｗ（上の式のα*y）の求め方である。

（理論的には線形分離できない空間をＫ（中間ユニット）で別の超平面に写像し、写像後の空間をｗで分離している）

Ｋはカーネル（核）関数と呼ばれ、いくつかの関数が提案されている。例えば、p 次の多項式カーネル $K(\mathbf{x},\mathbf{x}')=(\mathbf{x}^T\mathbf{x}'+1)^P$、やガウシアンカーネルである。以下では２次多項式カーネルを用いた例を示す。

ｗを求めるにはαを求めて、ｙを掛ければよい。αを求めるにはニューラルネットの学習のような反復法と、固有値問題を解くような方法がある。以下では簡単な繰り返し法を示す。

排他的論理和（XOR）の実現

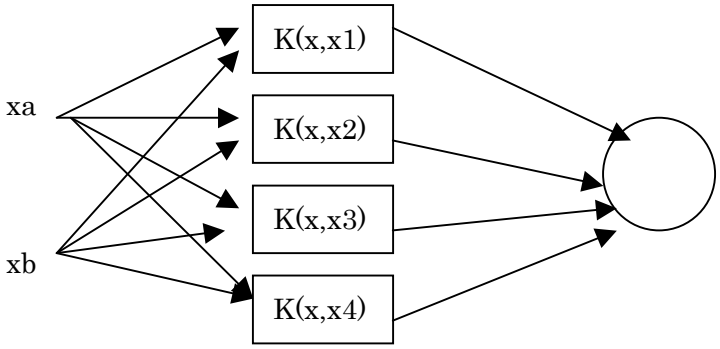
学習データは $(\mathbf{x}_1=(-1,-1)^T, y_1=-1),(\mathbf{x}_2=(-1,+1)^T, y_2=+1)$
$(\mathbf{x}_3=(+1,-1)^T, y_3=+1),(\mathbf{x}_4=(+1,+1)^T, y_4=-1)$ である。

２次多項式カーネルの式は次の通りである。

$$K(\mathbf{x},\mathbf{x}_i)=(1+\mathbf{x}^T\mathbf{x}_i)^2=1+x_1^2x_{i1}^2+2x_1x_2x_{i1}x_{i2}+x_2^2x_{i2}^2+2x_1x_{i1}+2x_2x_{i2}$$

この式でiはi番目の中間ユニット、すなわちi番目の学習パターンに対応したユニットを示している。

XORのネットの図は以下の通りである。



この図では入力をxa,xbとした。各中間ユニットの出力は、教師データ（x_{11}は教師データ１の１番目の要素（－１）を示す）を代入すると次のようになる。

$$K(\mathbf{x},\mathbf{x}_1)=(1+\mathbf{x}^T\mathbf{x}_1)^2=1+x_a^2x_{11}^2+2x_ax_bx_{11}x_{12}+x_b^2x_{12}^2+2x_ax_{11}+2x_bx_{12}$$
$$=1+x_a^2+2x_ax_b+x_b^2-2x_a-2x_b$$

$$K(\mathbf{x},\mathbf{x}_2)=(1+\mathbf{x}^T\mathbf{x}_2)^2=1+x_a^2x_{21}^2+2x_ax_bx_{21}x_{22}+x_b^2x_{22}^2+2x_ax_{21}+2x_bx_{22}$$
$$=1+x_a^2-2x_ax_b+x_b^2-2x_a+2x_b$$

$$K(\mathbf{x},\mathbf{x}_3)=(1+\mathbf{x}^T\mathbf{x}_3)^2=1+x_a^2x_{31}^2+2x_ax_bx_{31}x_{32}+x_b^2x_{32}^2+2x_ax_{31}+2x_bx_{32}$$
$$=1+x_a^2-2x_ax_b+x_b^2+2x_a-2x_b$$

$$K(\mathbf{x},\mathbf{x}_4)=(1+\mathbf{x}^T\mathbf{x}_4)^2=1+x_a^2x_{41}^2+2x_ax_bx_{41}x_{42}+x_b^2x_{42}^2+2x_ax_{41}+2x_bx_{42}$$
$$=1+x_a^2+2x_ax_b+x_b^2+2x_a+2x_b$$

KとYをまとめると次のようになる。

$$\mathbf{K}=(K(\mathbf{x}_i,\mathbf{x}_j))_{i,j}=\begin{pmatrix}9&1&1&1\\1&9&1&1\\1&1&9&1\\1&1&1&9\end{pmatrix},\mathbf{Y}=(y_iy_j)=\begin{pmatrix}1&-1&-1&1\\-1&1&1&-1\\-1&1&1&-1\\1&-1&-1&1\end{pmatrix}$$

ここで、$K(\mathbf{x}_i,\mathbf{x}_j)$とは、教師データiが入力された時のj番目のユニットの出力、y_iy_jは

教師データの出力 i と j の積を表す。
次に、中間層と出力層間の重みを反復計算で求める方法を示す。
w を求めるには α を求め y を掛ければよい。 α を反復計算で求める際は次の手順を行う。

Step1: α_i（i=1~4）に初期値 0 を入れる。また、次の行列 Dij を用意する。

$$D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) = \begin{pmatrix} 9 & -1 & -1 & 1 \\ -1 & 9 & 1 & -1 \\ -1 & 1 & 9 & -1 \\ 1 & -1 & -1 & 9 \end{pmatrix}$$

Step2:各教師データ（i=1,..,4）に対して次の処理を行う。

$$(a) E_i = \sum_{j=1}^{4} \alpha_j D_{ij}$$

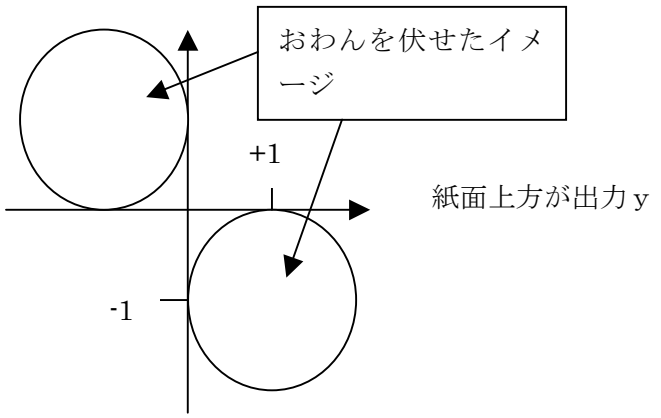
$$(b) \delta\alpha_i = \min(\max(\gamma(1 - E_i), -\alpha_i), C - \alpha_i)$$

$$(c) \alpha_i = \alpha_i + \delta\alpha_i$$

Step2 を α が収束するまで繰り返す
ここで、 γ は学習率、C は α の上限を決める正の数。

これをプログラムで実行した結果は、α 1~α 4＝0.125（全て同じ）であった。w は次の通りである。w1=0.125*-1=-0.25, w2=0.125, w3=0.125, w4=-0.125。
学習済みのネットでは出力 y の正負で入力パターンを 2 つのクラスに分類する。

上の例はカーネル関数に多項式を用いた、多項式では K のイメージがつかみにくいが、カーネルをガウシアンにすると RBF というモデルになりイメージがつかみやすくなる。RBF では、教師データを正規分布で表すので、入力が 2 つの場合、平面におわんを逆さに置いていくイメージになる。排他的論理和では以下のイメージとなる。



```
//αの学習プログラム
#include <stdio.h>
        float alpha[4]= {0,0,0,0};
        void report() {
        int i;
        for(i=0; i<4; i++)
                printf("%f,",alpha[i]);
            printf("¥n");
            }

    void main() {
            int i,count;
            float D[4][4] =   {{9,-1,-1,1},{-1,9,1,-1},{-1,1,9,-1},{1,-1,-1,9}};
            float E[4];
            float C=10.0, gamma=0.01, dalpha, mi, ma;

        for(count=0; count<1000; count++) {
            for(i=0; i<4; i++){

E[i]=alpha[0]*D[i][0]+alpha[1]*D[i][1]+alpha[2]*D[i][2]+alpha[3]*D[i][3];
                if ((gamma*(1.0-E[i]))> -alpha[i]) ma=gamma*(1.0-E[i]);
                else ma= -alpha[i];
                if( ma< (C-alpha[i])) dalpha= ma;
                else dalpha= C- alpha[i];
                alpha[i]+= dalpha;
                }
        }
        report();
    }
```

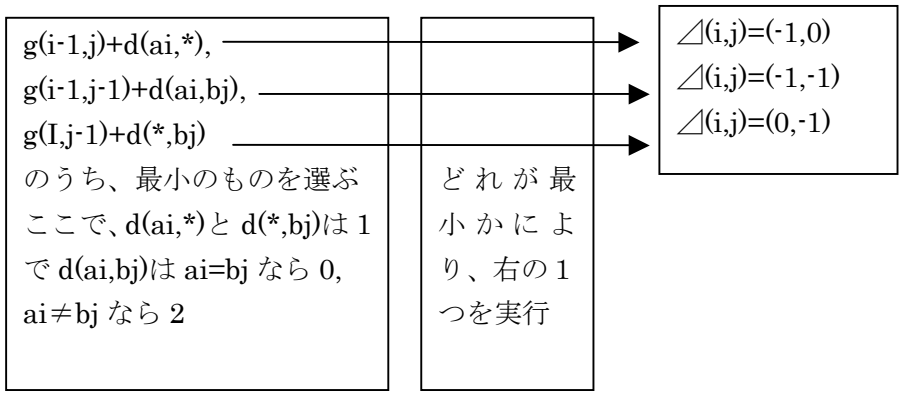
（１０）DP マッチング
DP マッチングを用いた生物間の距離（人と猿は、人と魚より距離が近い）の算出

前提：
１．距離をアミノ酸の配列で計る
２．長い時間の間にアミノ酸が置き換わる（例えば cdefg から cdsfg に）距離２とする
３．長い時間の間に脱落が起こる（例えば cdefg が cd*fg になる）距離１とする（*は脱落）
（言葉で言うと、置換より脱落が起こりやすい、ってこと）
課題：
２つのアミノ酸配列の距離を出すとき２つを並べて比較しますが、並べ方が色々考えられます。
例えば、abcc　と　aebc を並べて比較する時、
(1)　　abcc　　aebc　　と並べると b と e、c と b が置換されたことになり２か所置換なの距離２＊２＝４）
(2)　　a*bcc　　aebc*　と並べることもできます（距離は脱落１＋１で距離２）。
このように並べ方で距離が違ってきます。
長い間にどのような置換や脱落が起こったかは不明ですが、たぶん“距離が最小の変化が起きた”と考えるのが常識的です(この例でいえば(2))

そこで、今回の課題は、２つの生物間の距離最小になるような並べ替えを行い、その距離とします。
文字列が長くなるとこの“最小距離の文字配列”をみつけるのは容易ではありません。
そこで、DP を使って計算量を最小限にしようというわけです。

＝＝
DP マッチングのプログラム
パターン A(a1,a2,..,ai)とパターン B(b1,b2,…,bj)の Dynamic Pattern Alignment
I：パターン A の長さ
J：パターン B の長さ
p0～pk：(0,0)から(I,J)への最適経路
A’：A の整列化パターン
B’：B の整列化パターン
//初期設定
g(0,0)=0;
for(i=1; i<=I;i++){
　g(i,0)=g(i-1,0)+d(ai,*);
　⊿(i,0)=(-1,0);
}

```
for(j=1; j<=J+1;j++){
    g(0,j)=g(0,j-1)+d(*,bj);
    ⊿(0,j)=(0,-1);
}
//D(A,B)の計算
for(i=1; i<=I; i++){
    for(j=1;j<J;j++){
```




```
    }
}
D(A,B)=g(I,J)
```

最適経路は⊿をたどるころによって分かる。⊿は 2 次元配列で、配列要素に入る値｛(-1,0)または(-1,-1)または(0,-1)｝は、経路をたどる方向を示している。

ヒトとミドリムシのアミノ酸配列

```
H                              /
C                             /
Q                            /
A                           |
R                           |
S                          /
E                         /
F                ------
L              /
K             /
K            /
G           /
R          /
E        /
```

A　/
D　/
G /
　GDVEKGKKIFIMKCSQCH

D(A,B)=14
Aligned Sequence
A'= GDVEKGKKIFIMKCS***QCH
B'= GDAERGKKLF***ESRAAQCH

＝＝
課題：
abcdef　と　abcddf との並べ方と最短距離を示しなさい。並べ方は複数あると思います。
また、
abcdef　と　abcef　に関しても上と同様のことをしてみましょう。
＝＝
プログラム：

```
#include <stdio.h>
#define das　1
#define dsb　1
#define I 5
#define J 6
int dab, i, j, sum0, sum1, sum2, D, IJ, Ix, Jx;
char A[I]={'o','c','d','e','f'};
char Ad[I+J];
char B[J]={'o','a','b','c','d','e'};
char Bd[I+J];
int g[I][J], del[I][J];

main(){
    g[0][0]=0;
    for( i=1; i<I; i++){
        g[i][0]= g[i-1][0]+ das;
        del[i][0]=-1;
    }
    for( j=1; j<J; j++){
        g[0][j]= g[0][j-1]+ dsb;
```

```
            del[0][j]=-3;
    }
    for(i=1; i<I; i++){
        for(j=1; j<J; j++){
            if(A[i]==B[j]) dab=0;
            else dab=2;
            sum0= g[i-1][j]+das;
            sum1= g[i-1][j-1]+ dab;
            sum2= g[i][j-1] +dsb;

if(((sum0<sum1)&&(sum0<sum2))||((sum0==sum1)&&(sum0<sum2))||((sum0==su
m2)&&(sum0<sum1))) {
                g[i][j]= sum0;
                del[i][j]= -1;
            }

if(((sum1<sum0)&&(sum1<sum2))||((sum1==sum0)&&(sum1<sum2))||((sum1==su
m0)&&(sum1<sum2))) {
                g[i][j]= sum1;
                del[i][j]= -2;
            }

if(((sum2<sum0)&&(sum2<sum1))||((sum2==sum1)&&(sum2<sum0))||((sum2==su
m0)&&(sum2<sum1))) {
                g[i][j]= sum2;
                del[i][j]= -3;
             }
            if((sum1==sum0)&&(sum1==sum2)) {
                g[i][j]= sum1;
                del[i][j]= -2;
            }
         }
     }
//   類似度表示
    printf("D=%d¥n",g[I-1][J-1]);
// g[][]の表示
    for(i=0; i<I; i++){
        for(j=0;j<J;j++)    printf("%d,",g[i][j]);
```

```
        printf("¥n");
    }
// delta の表示
    for(i=0; i<I; i++){
        for(j=0;j<J;j++)    printf("%d,",del[i][j]);
        printf("¥n");
    }
//  最短経路決定
    for(i=0; i<I+J-1; i++){
        Ad[i]='o';
        Bd[i]='o';
    }
    Ix=I-1;
    Jx=J-1;
        for(i=I+J-1; i>0 ;i--){
            if(del[Ix][Jx]==-2){
                Ad[i]= A[Ix];
                Bd[i]= B[Jx];
                Ix--;
                Jx--;
            }
            else if(del[Ix][Jx]==-3){
                Ad[i]= '*';
                Bd[i]= B[Jx];
                Jx--;
            }
            else if(del[Ix][Jx]==-1){
                Ad[i]= A[Ix];
                Bd[i]= '*';
                Ix--;
            }
        }
//   最短経路表示
    for(i=0; i<I+J ;i++)
        if(Ad[i]!='o')    printf("%c",Ad[i]);
    printf("¥n");
    for(i=0; i<I+J ;i++)
      if(Bd[i]!='o')     printf("%c",Bd[i]);
```

```
    printf("¥n");
}
```

（１１）隠れマルコフモデル（HMM）

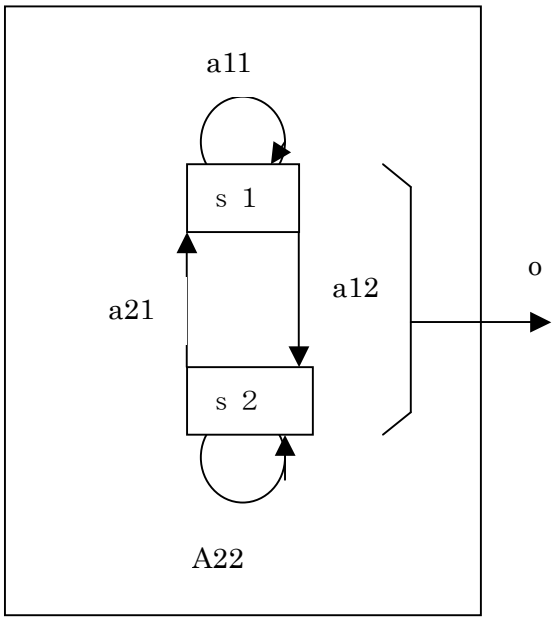
HMMを特徴付けるパラメータＳ，Ｏ，Π、Ａ、Ｂ

HMモデルは内部状態の遷移を繰り返す（例えば s1,,s2,s2,s1,....）。状態は常にどれか１つの状態にある。その状態遷移に伴って出力Ｏを出す。

＜やりたいこと＞HMMを複数用意しておき（HMM1,HMM2,....）、出力系列が得られた時、どの HMM から出力されたものかを推定する。

＜例１＞　生物のＤＮＡ系列をHMMで表し、ＤＮＡから種を推定する。

＜例２＞　音声系列から単語を推定する



S={s1,s2}：状態は s1 と s2 がある。

O={o1,o2}：出力は o1 と o2 がある（例えば o1 が０で、o2 が１）。

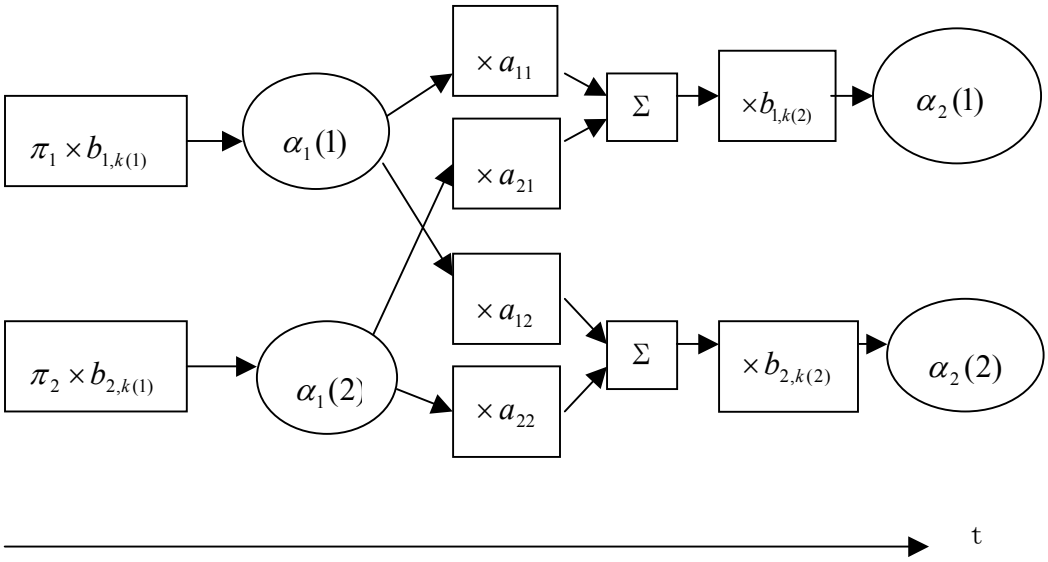
Π=(π1,π2)：π１、π２は初期分布の確率。例えば（0.3,0.7）。

Ａ：状態遷移確率（上図参照）

Ｂ：出力確率（b11 は s1 の時 o1 を出力する確率、b12 は s1 の時 o2 を出力する確率、
b21 は s2 の時 o1 を出力する確率、b22 は、、、）。

出力系列から、マルコフモデルがその出力系列を出す確率を計算する。この計算法に Forward 法がある。
参考文献（上坂、尾関：パターン認識と学習アルゴリズム（文一総合出版））

Forward Algorithm



k(1)は時刻 1 の出力　　k(2)は時刻 2 の出力

$\alpha_t(i)$の t は時刻

（１２）ヒューリスティック（発見的）探索

片っ端から闇雲に探すのではなく、知恵を使って探そう、という話。最高峰をめざす探検を例に考える。

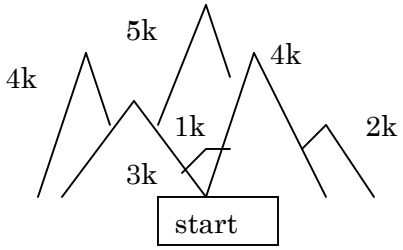


Figure 1 最高峰を目指せ

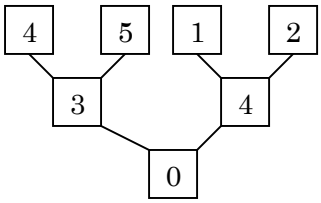


Figure 2　探索木

（１）　山登り法

とりあえず、今いける最高点へ進む。

その後は、そこから最高点へ移る。

上の例では、０－＞４－＞２と進む。

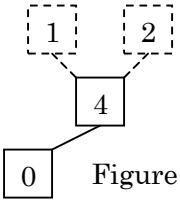


Figure 3 山登り法

（２）　最良優先探索

端末の全ノードの中から最高の方向に進む。右のように、とりあえず４に進み、１、２，３を比較し３に進む。

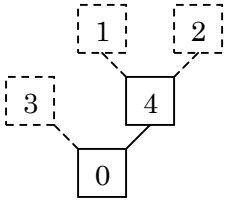


Figure 4 最良優先探索

一般に最良優先探索では、評価関数によっては探索コストが最小になる解を見つけ出せる保障はない。最良優先探索のひとつであるＡ＊法（後述）は、探索コストの点で必ず最適解を求めることができるという特徴をもつ。

（13） 最短経路探索（ダイクストラ法）

ダイクストラ法　（http://prog.usamimi.info/algo/graph/dijkstra.php）

ダイクストラ法のプログラムを、図１の地図を例に示す。 都市間の接続関係を「隣接行列」という、２次元配列で表す。 ２次元配列には、都市間が接続していれば距離の数字を、接続されていない場合は -1 を入れる。ちなみに自分自身の距離は 0 を入れる。

ダイクストラ法の基本的な考え方

ダイクストラ法とは各都市への最短経路を始点の周辺から１つずつ確定し、徐々に範囲を拡大していき、最終的にすべての節点への最短経路を求める。 アルゴリズムは、次の通りである。

＊　始点と接続している都市のなかで、始点との距離が最小のものを探してこの都市を確定する。

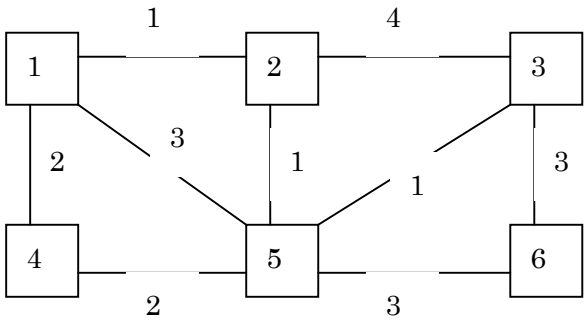
＊　確定した都市に接続している都市と、始点との距離を求め、先ほど確定されなかった都市と 合わせた中で始点との距離が最小のものを探してその都市を確定する。

＊　これをすべての都市（あるいは終点の都市）が確定するまで繰り返すと、最短経路路が求まる。

ダイクストラ法の詳細

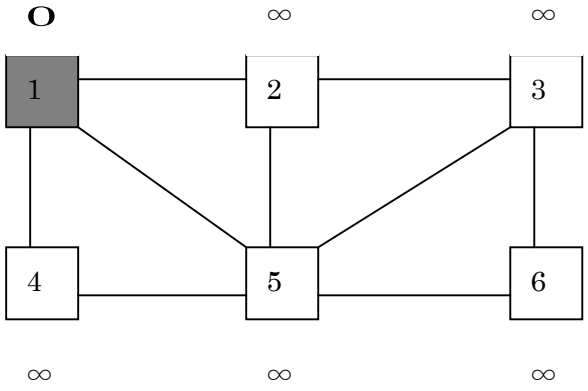
（http://www.me.sophia.ac.jp/or/lab/ishizuka/OC/spath_00.html）

ダイクストラ法では，目的地への最短経路だけでなく，その他の全ての都市への最短経路を同時に求めることができる.次の都市配置と都市間距離を例にアルゴリズムを紹介する。都市１から都市６への距離を求めていく。

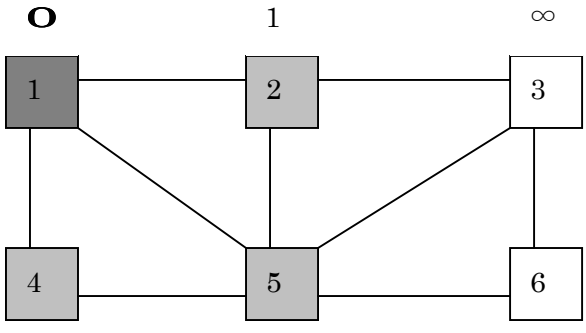


以下で，順次，都市 1 からの最短距離が求まった都市は濃灰色で塗り潰し，その上に都市 1 からの最短距離を大数字で記入していく．また，すでに最短距離が確定している都市に直接つながっている都市を薄灰色に塗り潰し，その上に小数字で「仮の最短距離」を書き込む．この「仮の最短距離」は，それに至る直前の(最短距離が確定している）都市にもとづいて更新する．

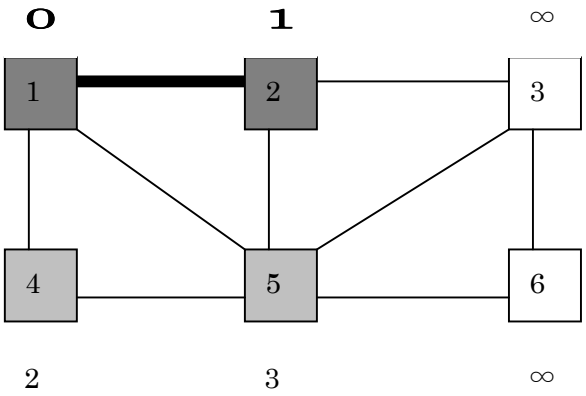
まず，出発都市 1 から都市 1 への距離は当然 0 で，確定しているので次のようになる．



今最短距離が確定した都市 1 から直接行ける都市(2，4，5）に 仮の最短距離 をつける．今「仮の最短距離」は，今確定した都市(都市 1)からの距離になる．



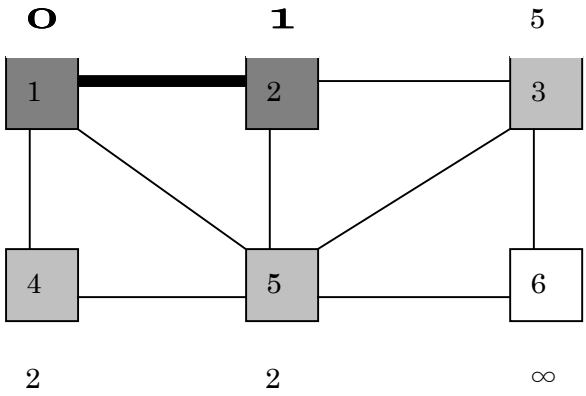
次に、仮の最短距離が 2 小さい都市を 3 それは，都市 ∞ 直接行ける都市の中での最小値なので，その の仮の最短距 都市 1 からの最 離として確定する．今の場合，それは都市 2 となる．ここで，太線は，確定した最短経路 を示す．



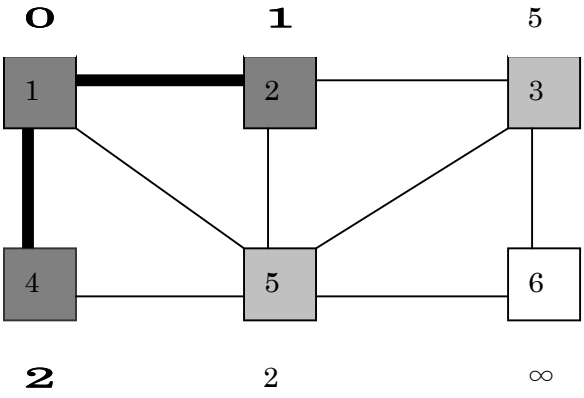
あらたに確定した都市(都市 2)から直接行ける都市の仮の最短距離を更新する．都市５の仮の最短距離は 3 たっだが，都市 2 を経由すれば，

都市 2 までの最短距離(= 1) + 2 と 3 の間の距離 (= 1) = 2

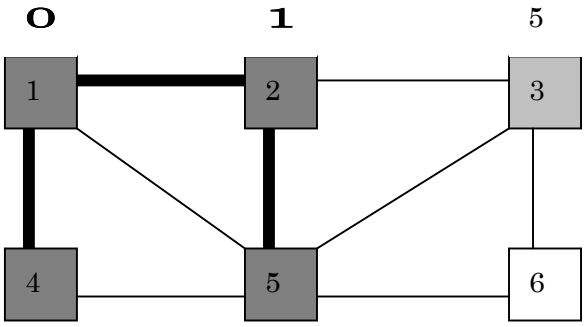
で行けるので，都市５の仮の最短距離は 2 に更新する．また都市 3 の仮の最短距離も 5 に更新される．



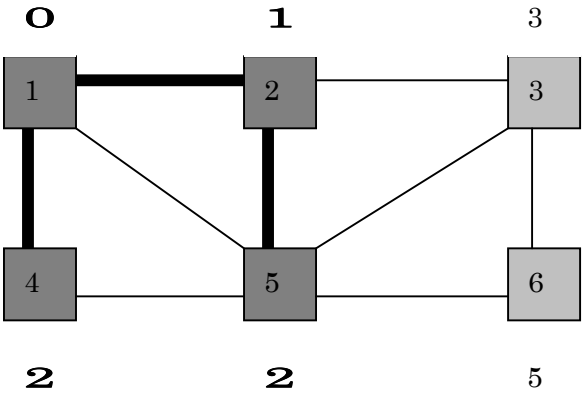
以下，同じ操作を続ける．仮の最短距離が最小の都市は、今度は都市 4 と都市 5 の 2 個所あり，どちらを選んでも良いが，例えば都市 4 を選ぶ．すると，その仮の最短距離の 2 は確定となる(もし，都市 4 に至るもっと短い経路があるなら，その経路は，都市 2 か 5 を経由していることになるが，今，都市 2 は最小の仮の最短距離を持っていたわけで，そういうことはあり得ない)．



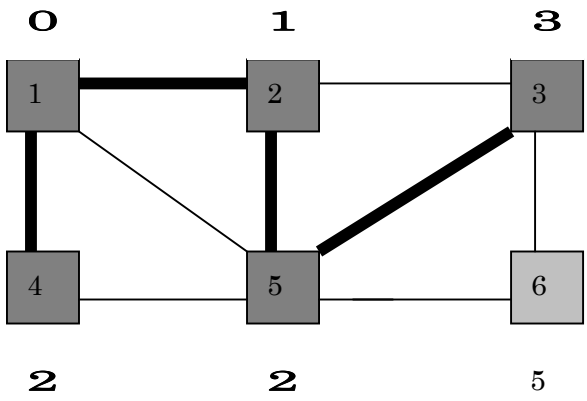
最小の仮の最短距離を持っている都市は都市 5 となり,



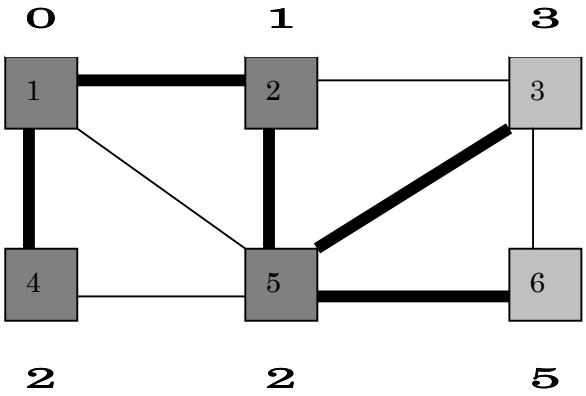
それにつながっている者 2 仮の最短距離 2 する. ∞



次に都市 3 が確定し，



最終的に，

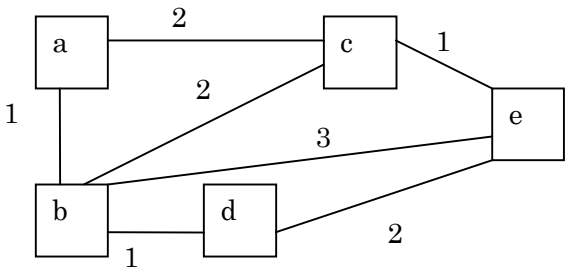


となり，都市 1 からその他全ての都市への最短距離と最短経路が求まる．

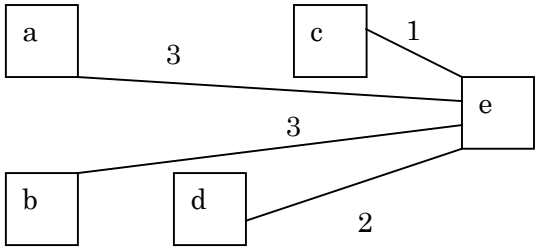
（１４）ダイクストラ法と A*法のプログラム

ダイクストラ法では「出発点からその点までの実合計距離」が短い点から順に探索していくのに対して、A*法では「出発点からその点までの実合計距離」＋「その点から到着点までの直線距離」が短い点から順に評価をおこなう。

A*法は、最初に終着点に到着した時が最短距離となる（ダイクストラ同じかも)。



都市間経路（数値は距離）



上の例で、目的地までの直線距離

この時の最短経路を求めるダイクストラ法とＡ＊法のプログラム

都市間距離には次の隣接行列を用いる

	a	b	c	d	e
a	0	1	2	-1	-1
b	1	0	2	1	3
c	2	2	0	-1	1
d	-1	1	-1	0	2
e	-1	3	1	2	0

隣接行列

プログラム（ダイクストラ法）

```
import java.io.*;
import java.util.*;

public class Dijkstra2 {
  static int[] path = new int[5];
  static char[] name = {'a', 'b', 'c', 'd', 'e'};
  static int start, end, p, min;

  public static void main (String[] args) throws Exception {
        //  地図情報
    int
map[][]={{0,1,2,-1,-1},{1,0,2,1,3},{2,2,0,-1,1},{-1,1,-1,0,2},{-1,3,1,2,0}};
        int[] len = new int[5];          // 始点から各節点までの最短距離保持用
    boolean[] visit = new boolean[5];  // 確定用
    start = 0;  // a を始点にする
    end = 4;    // e を終点にする
    p = 0;

    // 初期化
    for (int i=0; i<5; i++) {
      len[i] = -1;
      path[i] = -1;
      visit[i] = false;
    }
    len[start] = 0;

    for (int i=0; i<5; i++) {
      // 最小の節点を探す
      min = -1;
      for (int j=0; j<5; j++) {
        // まだ確定していなく、接続されているかどうか
        if (!visit[j] && len[j] != -1) {
          // 最小値よりも小さい節点が見つかった場合
          if (min == -1 || len[j] < min) {
            p = j;
            min = len[j];
```

```
        }
      }
    }

    visit[p] = true;
    if (p == end) break;

    // p を経由した j (len[p] + map[p][j]) と、
    // 始点との距離がそれまでの最短路よりも小さければ更新
    for (int j=0; j<5; j++) {
      // p と j が接続されていているかどうか
      if (map[p][j] != -1) {
        if (len[j] == -1 || (len[p] + map[p][j]) < len[j]) {
          len[j] = len[p] + map[p][j];
          path[j] = p;
        }
      }
    }
  }
  // 終点から最短距離をスタック (再帰) を用いて逆順で表示
  Result(end);
  System.out.println("¥nLength " + len[end]);
 }

 static void Result (int p) {
  if (path[p] != -1) Result(path[p]);
  System.out.print(name[p]);
  if (p != end) System.out.print(" -> ");
 }
}
```

プログラム (A*法)

```
import java.io.*;
import java.util.*;

public class Astar1 {
 static int[] path = new int[5];
 static char[] name = {'a', 'b', 'c', 'd', 'e'};
```

```
  static int start, end, p, min;

  public static void main (String[] args) throws Exception {
        //  地図情報
    int
map[][]={{0,1,2,-1,-1},{1,0,2,1,3},{2,2,0,-1,1},{-1,1,-1,0,2},{-1,3,1,2,0}};
        int toe[]={3,3,1,2,0};
        int[] len = new int[5];            // 始点から各節点までの最短距離保持用
    boolean[] visit = new boolean[5];  // 確定用
    start = 0;  // a を始点にする
    end = 4;    // e を終点にする
    p = 0;

    // 初期化
    for (int i=0; i<5; i++) {
      len[i] = -1;
      path[i] = -1;
      visit[i] = false;
    }
    len[start] = 0;

    for (int i=0; i<5; i++) {
      // 最小の節点を探す
      min = -1;
      for (int j=0; j<5; j++) {
        // まだ確定していなく、接続されているかどうか
        if (!visit[j] && len[j] != -1) {
          // 最小値よりも小さい節点が見つかった場合
          if (min == -1 || (len[j]+toe[j]) < min) {
            p = j;
            min = len[j]+toe[j];
          }
        }
      }

      visit[p] = true;
      if (p == end) break;
```

```
      // p を経由した j (len[p] + map[p][j]) と、
      // 始点との距離がそれまでの最短路よりも小さければ更新
      for (int j=0; j<5; j++) {
        // p と j が接続されていているかどうか
        if (map[p][j] != -1) {
          if (len[j] == -1 || (len[p] + map[p][j]) < len[j]) {
            len[j] = len[p] + map[p][j];
            path[j] = p;
          }
        }
      }
    }
    // 終点から最短距離をスタック (再帰) を用いて逆順で表示
    Result(end);
    System.out.println("¥nLength " + len[end]);
  }

  static void Result (int p) {
    if (path[p] != -1) Result(path[p]);
    System.out.print(name[p]);
    if (p != end) System.out.print(" -> ");
  }
}
```

終わり