脳の自律的プログラム合成機構のモデルに向けて: 2層ベイジアンネットによる 記号処理命令の獲得・実行機構

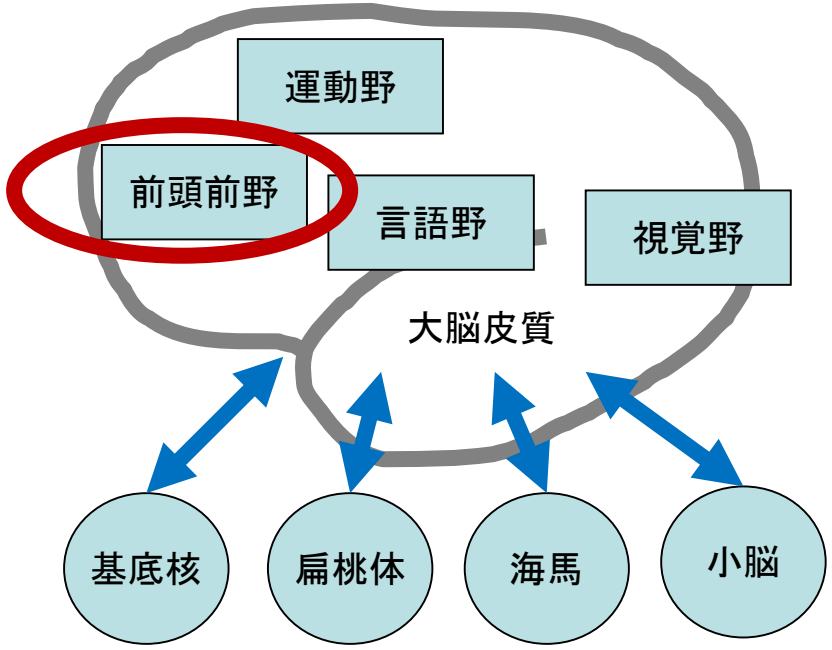
第15回 汎用人工知能研究会
2020-08-29

一杉裕志[1*] 中田秀基[1] 高橋直人[1] 佐野崇[2]
Yuuji Ichisugi[1] Hidemoto Nakada[1] Naoto Takahashi[1] Takashi Sano[2]

[1] 産業技術総合研究所 人工知能研究センター
[1] National Institute of Advanced Industrial Science and Technology (AIST), AIRC
[2] 東洋大学 情報連携学部情報連携学科
[2] Faculty of Information Networking for Innovation And Design, Toyo University

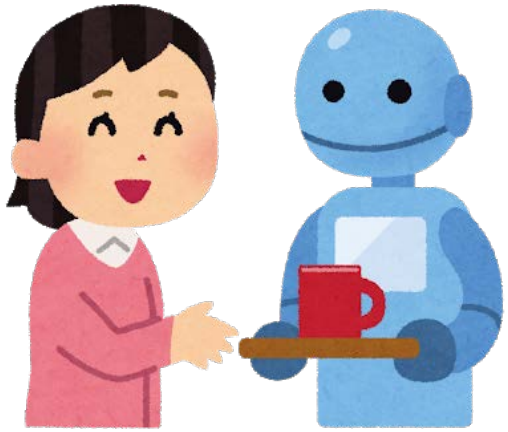
私の研究の中期的目標

- ヒトの脳の前頭前野周辺の計算論的モデルの構築
- 「前頭前野はヒトをヒトたらしめ，思考や創造性 を担う脳の最高中枢であると考えられている。」「前頭前野 - 脳科学辞典」
  http://bsd.neuroinf.jp/wiki/%E5%89%8D%E9%A 0%AD%E5%89%8D%E9%87%8E
- 作業仮説：前頭前野は強化学習を用いたプログラム合成・実行システム



目指しているもの

「お客さんにお茶を出しておいて。」



しかし、お茶を入れる複雑な手順を
人手でプログラムすることは不可能

「手伝ってくれたらあとで
ご褒美あげるね。」

報酬を目的とした
自律的なプログラムの獲得

関連研究

- 記号AIと強化学習の融合の試み:
  - RRL (Relational Reinforcement Learning), Dzeroski et al. 2001
    - 強化学習と帰納論理プログラミングの融合
    - 行動価値関数を論理決定木に圧縮
  - DNC(Differentiable Neural Computers), DeepMind 2016
    - NN でプログラムを表現、強化学習等で学習
- 強化学習・プログラム合成によるAGIの理論:
  - AIXI, Hutter 2000
  - UCAI, Katayama 2018
- **我々のアプローチの特徴:**
  - **神経科学、認知科学の知見との整合性を重視**
  - **ベイジアンネットを用いる**

階層型強化学習RGoalを用いた記号推論の実現手法[一杉 et al.第12回 汎用人工知能研究会 2019]

- 行動価値関数における「状態」を命題の集合で表現
- 「行動」は「状態」(ワーキングメモリ)を更新
- 命題の集合を固定長のベクトルで表現

P(a, b) ∧ Q(c) → R	: (P a b Q c 0 R 0 0)
P(a, b) → Q(c)	: (P a b 0 0 0 Q c 0)
P(a, b)	: (0 0 0 0 0 0 P a b)

prolog による記述:

```
exist(yesterday, snack).
exist(yesterday, chocolate).
notEat(brother, chocolate).
exist(today,X) :- exist(yesterday,X), notEat(brother,X).
canEat(X) :- exist(today, X).

?- canEat(X).
X = chocolate.
```

RGoal の行動価値関数による表現

```
Q((__,__,__,__,__,__,__,__,__), (__,__,__,__,__,__,__,__,__), Fail)

Q((__,__,__,__,__,__,__,__,__), (O,O,O,O,O,O,Exist,Yesterday,Snack), Set(O,O,O,O,O,O,Exist,Yesterday,Snack))

Q((__,__,__,__,__,__,__,__,__), (O,O,O,O,O,O,Exist,Yesterday,Chocolate), Set(O,O,O,O,O,O,Exist,Yesterday,Chocolate))

Q((__,__,__,__,__,__,__,__,__), (O,O,O,O,O,O,NotEat,Brother,Chocolate), Set(O,O,O,O,O,O,NotEat,Brother,Chocolate))

Q((__,__,__,__,__,__,__,__,__), (O,O,O,O,O,O,Exist,Today,x), Call(Exist,Yesterday,x,O,O,O,O,O,O))
Q((Exist,Yesterday,x,O,O,O,O,O,O), (O,O,O,O,O,O,Exist,Today,x), Call(Exist,Yesterday,x,NotEat,Brother,x,O,O,O))
Q((Exist,Yesterday,x,NotEat,Brother,x,O,O,O), (O,O,O,O,O,O,Exist,Today,x), Set(O,O,O,O,O,O,Exist,Today,x))
Q((__,__,__,__,__,__,__,__,__), (Exist,Yesterday,x,O,O,O,O,O,O), Call(O,O,O,O,O,O,Exist,Yesterday,x))
Q((O,O,O,O,O,O,Exist,Yesterday,x), (Exist,Yesterday,x,O,O,O,O,O,O), Set(Exist,Yesterday,x,O,O,O,O,O,O))
Q((Exist,Yesterday,x,O,O,O,O,O,O), (Exist,Yesterday,x,NotEat,Brother,x,O,O,O), Call(O,O,O,O,O,O,NotEat,Brother,x))
Q((O,O,O,O,O,O,NotEat,Brother,x), (Exist,Yesterday,x,NotEat,Brother,x,O,O,O),
Set(Exist,Yesterday,x,NotEat,Brother,x,O,O,O))

Q((__,__,__,__,__,__,__,__,__), (O,O,O,O,O,O,CanEat,x,O), Call(O,O,O,O,O,O,Exist,Today,x))
Q((O,O,O,O,O,O,Exist,Today,x), (O,O,O,O,O,O,CanEat,x,O), Set(O,O,O,O,O,O,CanEat,x,O))
```

パターンを用いた行動価値テーブルの圧縮

[一杉 et al.第10回 汎用人工知能研究会 2018]

X \ Y	0	1	2	3	4
0	2.0	1.0	1.0	3.0	1.0
1	1.0	2.0	1.0	3.0	1.0
2	1.0	1.0	2.0	3.0	1.0
3	1.0	1.0	1.0	4.0	1.0
4	1.0	1.0	1.0	3.0	2.0

パターン	値
(3,3)	4.0
(X,3)	3.0
(X,X)	2.0
(X,Y)	1.0

テーブルのインデックスの代わりにパターンを用いることで、
サイズ 5 x 5 = 25 のテーブルがサイズ 4 に圧縮

・ルールの順序には意味はないものとする。
・マッチするルールが複数ある場合、最も特殊なパターンをもつルールの方を選択する。
・「最も特殊なパターン」とは、用いる変数の種類が最も少ないパターンと定義。

前頭前野におけるパターンマッチ

前頭前野外側部で動作順序をパターンで抽象化した動作カテゴリとして表現

Keisetsu Shima, Masaki Isoda, Hajime Mushiake and Jun Tanji,
Categorization of behavioural sequences in the prefrontal cortex,
Nature, 445, 315-318, 2007.

参考：虫明 元「前頭葉のしくみ―からだ・心・社会をつなぐネットワーク― 」
共立出版, 2019.

言語野におけるパターンマッチ

- **理論言語学における文法記述の枠組みの多くが単一化文法**
  - 単一化文法：単一化を使って構文解析を進めていく文法枠組みの総称
  - 単一化(unification)はパターンマッチの一般化

$$\dfrac{\dfrac{\overset{\text{black}}{NP/NP : \lambda x.black(x)} \quad \overset{\text{cats}}{NP : cats}}{NP : black(cats)} > \quad \dfrac{\overset{\text{eat}}{(S\backslash NP)/NP : \lambda y.\lambda x.eat(x,y)} \quad \overset{\text{mice}}{NP : mice}}{S\backslash NP : \lambda x.eat(x, mice)} >}{S : eat(black(cats), mice)} <$$

自然言語の性質を表現する道具として洗練され到達した1つの結果が単一化文法であるという事実は、脳の言語野で何らかの形で単一化が行われていることを強く示唆している。

ヒト前頭前野が獲得する「プログラム」のイメージ

行動ルール：スパースな整数値ベクトルのパターン、
状態行動対を抽象化したもの

プログラム：
100万個程度の
行動ルールの集合

(_,_,_,29,_,_,_,_,_,X,_,X,_,_,...)
(_,_,_,48,_,_,_,_,_,X,_,X,_,_,...)
(_,_,_,62,_,_,_,_,_,X,_,X,_,_,...)
...
(29,_,X,_,_,_,Y,_,_,_,12,_,_,Y,...)

参照・更新

(0,0,0,29,0,0,0,0,94,0,0,0,0,0,0,0,...)

ワーキングメモリ：
100万次元程度のスパースな
整数値ベクトル

大脳皮質とベイジアンネットの類似点

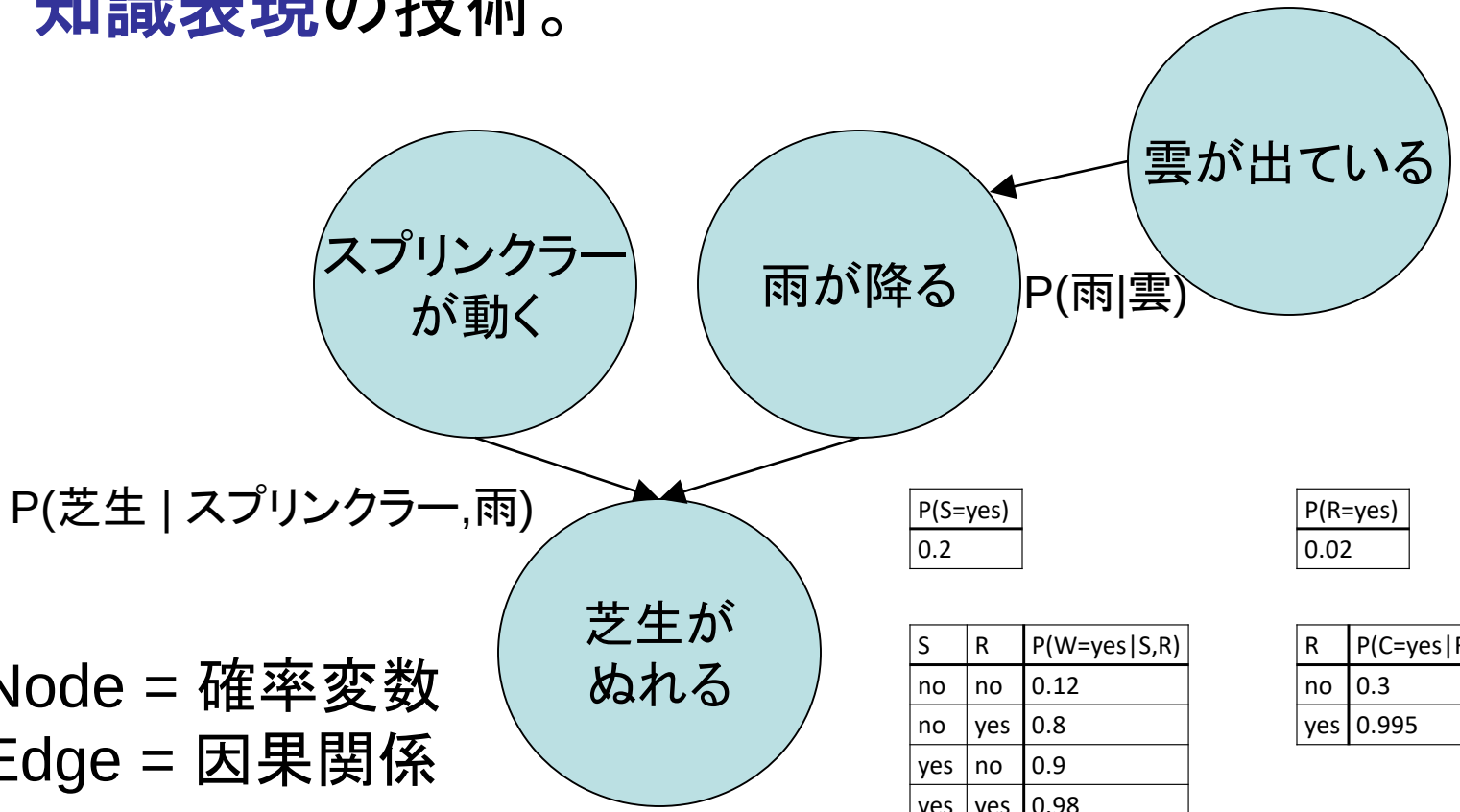
- トップダウンとボトムアップの非対称の接続
- 局所的かつ非同期な情報のやり取りだけで動作
- 値が非負
- 情報が正規化される
- ヘブ則学習
- 文脈や事前知識に依存した認識
- ベイズに基づく動作

一杉裕志, 解説：大脳皮質とベイジアンネット、
日本ロボット学会誌 Vol.29 No.5, pp.412--415, 2011.

Daniel J. Felleman and David C. Van Essen
Distributed Hierarchical Processing in the Primate Cerebral Cortex
Cerebral Cortex 1991 1: 1-47

ベイジアンネット[Pearl 1988]とは

- 脳の「直感・連想記憶」と似た働きをする。
- 確率変数の間の因果関係をグラフで**効率的**に表す**知識表現**の技術。



P(S=yes)
0.2

P(R=yes)
0.02

S	R	P(W=yes|S,R)
no	no	0.12
no	yes	0.8
yes	no	0.9
yes	yes	0.98

R	P(C=yes|R)
no	0.3
yes	0.995

ベイジアンネットでパターンを使った情報圧縮は可能か？

- 前提：
  - ヒトの大脳皮質はパターンを使って情報を圧縮する能力を持っているらしい。
  - 大脳皮質はおそらくベイジアンネットである。
- ベイジアンネットを使ってデータからのパターン獲得・パターンマッチができるはず。
  → 今回、実装して動作確認した。

- 強化学習を用いたプログラム合成の重要な要素技術
- 記号AIと統計的機械学習の橋渡し

提案手法

条件付確率モデル

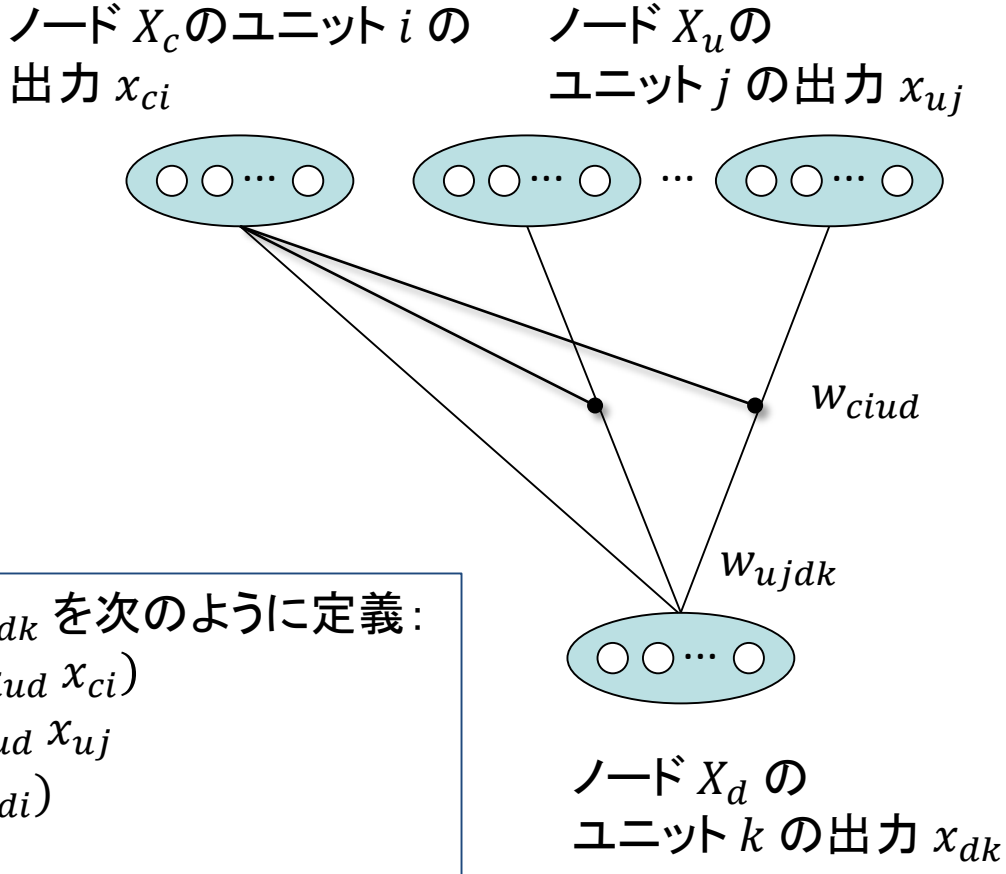
$$P\left(X_d = d_k \middle| X_u = u_j, \dots\right) = x_{dk}$$

・U, D はそれぞれ上の層・下の層のノードのインデックスの集合で $c, u \in U, d \in D$

・x_{ci}, x_{uj}, x_{dk} はそれぞれ変数 X_c, X_u, X_d の値のone hot vector 表現

・w_{ciud} は、X_c のユニット i が X_u と X_d の間の結合を制御する重み

・w_{ujdk} は、X_u のユニット j と、X_d のユニット k の間の重み



$u \in U, d \in D, k \neq 0$ に対して g_{ud}, s_{dk}, x_{dk} を次のように定義:

$$g_{ud} = \Pi_{c \in U} \Pi_{i \neq 0} (1 - w_{ciud}\, x_{ci})$$

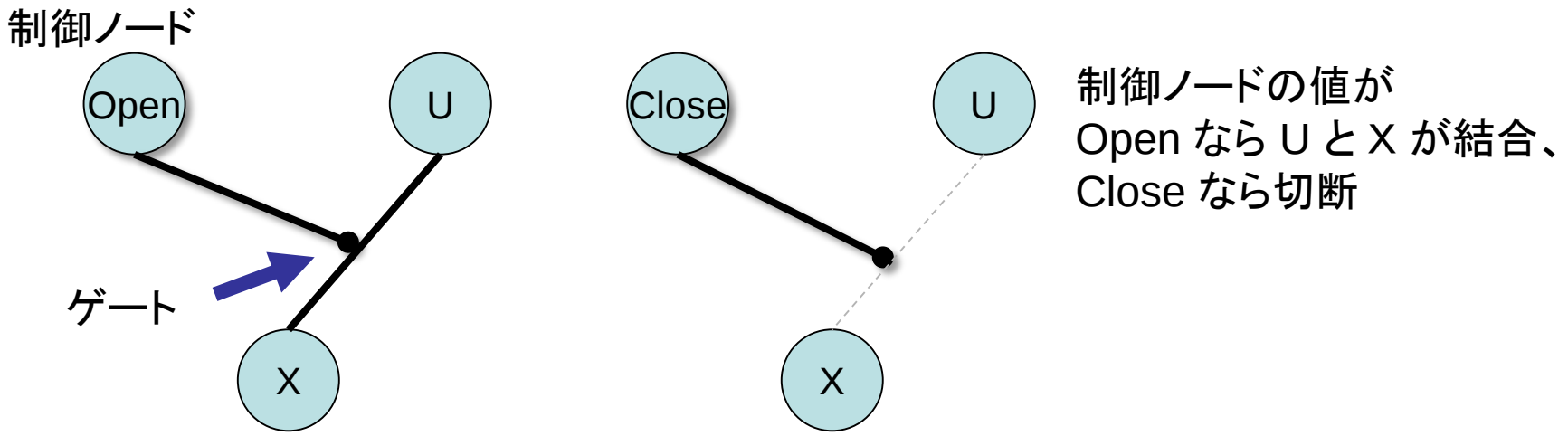
$$s_{dk} = \Sigma_{u \in U} \Sigma_{j \neq 0}\, w_{ujdk}\, g_{ud}\, x_{uj}$$

$$x_{dk} = s_{dk} / \max(1, \Sigma_{i \neq 0} s_{di})$$

x_{d0} を次のように定義:

$$x_{d0} = 1 - \Sigma_{k \neq 0} x_{dk}$$

ゲート

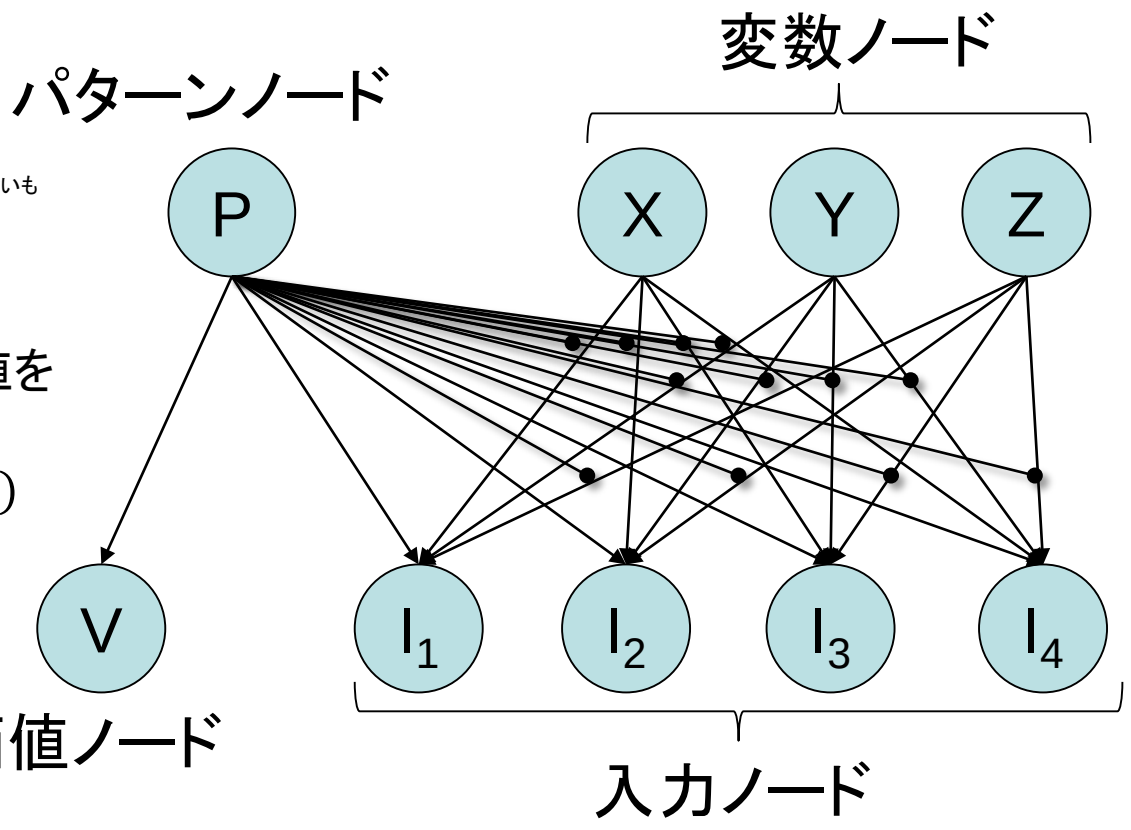


・**あくまでベイジアンネット**なので、出力（下流の値）から入力（上流の値）の推定もできる。

パターンマッチを行うベイジアンネット

[X, 1, 2]=>0.1
[X, X, 2]=>0.2
[1, X, Y]=>1
…

注：パターンノードは値0を取らないものとする



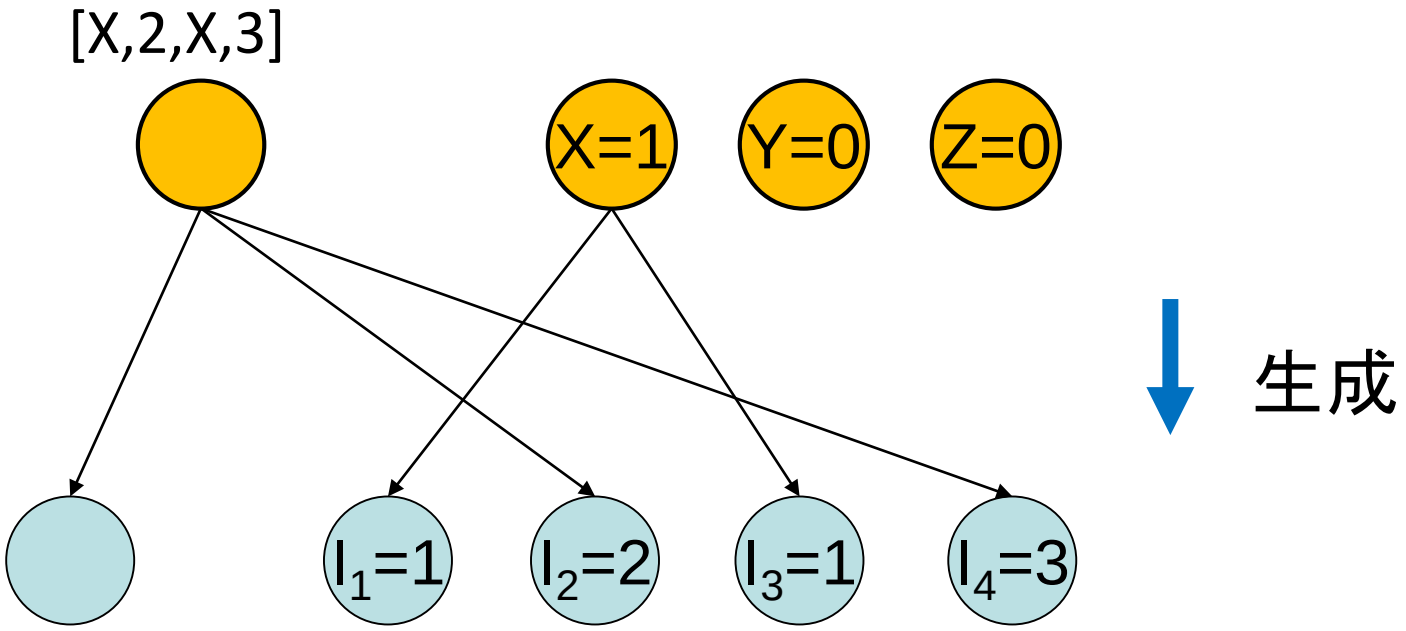
変数ノードの「発火」がスパースになるように事前分布を設定

パターン p の価値を r_p とすると

$P(V = 1|P = p) = r_p$

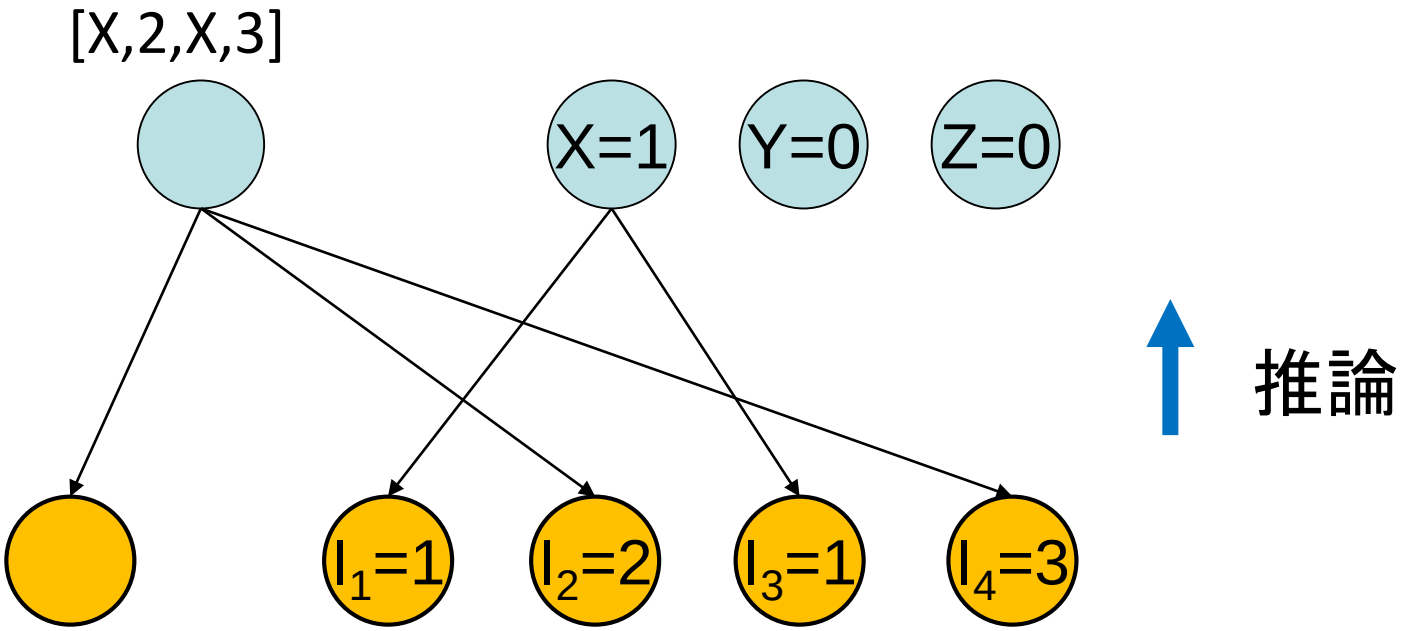
接続されている変数ノードと入力ノードは同じ値になるように重みを設定

下の層の値を生成する例



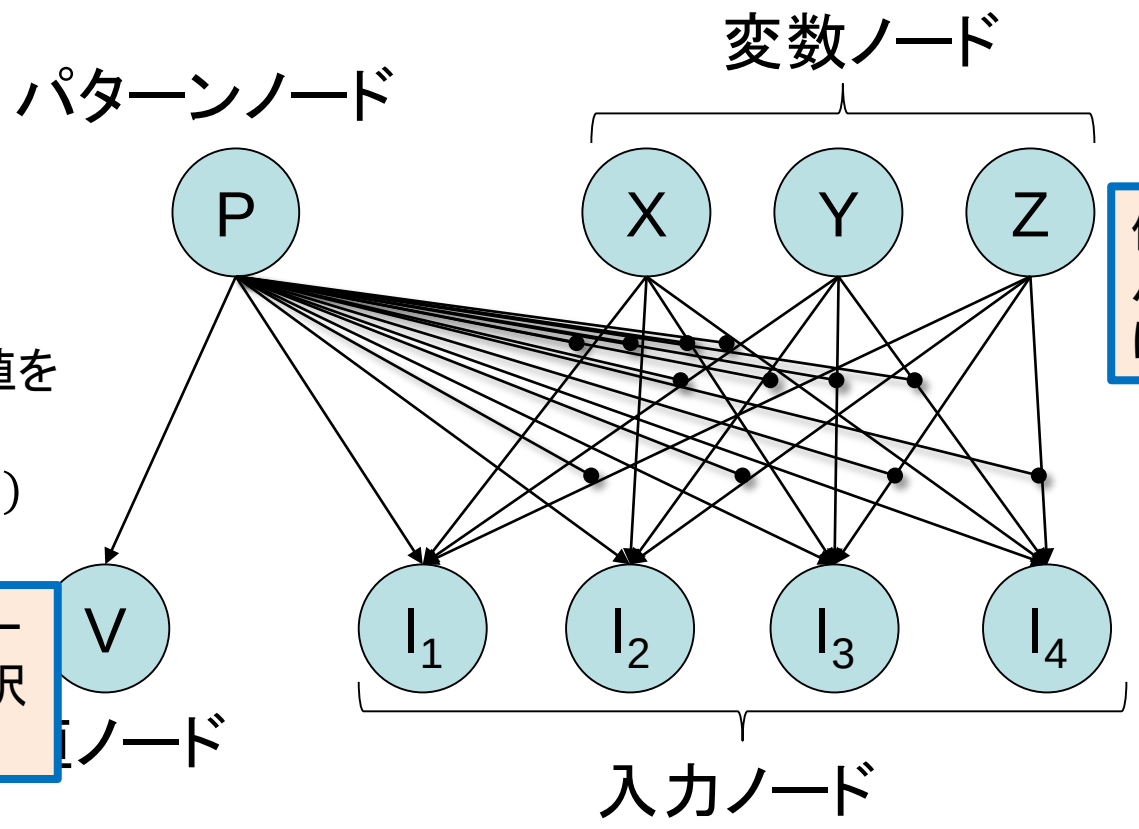
パターンノードが変数ノードと入力ノードの
間の結合を制御

パターンマッチの例



オートエンコーダーの形をしている。
→ 原理的に、下の層にデータを与えればパターンを教師なし学習可能
（今回は未実装、教師あり学習のみ実装）

最も特殊なパターンの選択
最も価値の高いパターンの選択



変数ノードの「発火」はスパース

使う変数が少ないパターンが優先的に選択される

パターン p の価値を r_p とすると

$P(V = 1|P = p) = r_p$

価値の高いパターンが優先的に選択される

実行例

パターンの定義からベイジアンネットを生成

Julia で実装した DSL (domain specific language) でパターンを定義

```
pat = defPattern(:[
    [X, 1, 2]=>1
    [X, X, 2]=>1
    [1, X, Y]=>1
], uNodes=2)
```

提案手法では重みの役割が決まっているのでパターンの定義から機械的にベイジアンネットを生成できる。

結合の重みテーブル

変数 X と入力の間の結合の制御

変数 Y と入力の間の結合の制御

```
w_ciud = [
 [
   [[0.0, 0.0, 0.0, 0.0, ], [0.0, 1.0, 1.0, 0.0, ], [1.0, 1.0, 1.0, 0.0, ], ],
   [[0.0, 0.0, 0.0, 0.0, ], [0.0, 0.0, 1.0, 0.0, ], [1.0, 1.0, 1.0, 0.0, ], ],
   [[0.0, 0.0, 0.0, 0.0, ], [1.0, 0.0, 1.0, 0.0, ], [1.0, 1.0, 0.0, 0.0, ], ],
],
 [
   [[0.0, 0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, 0.0, ], ],
   [[0.0, 0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, 0.0, ], ],
   [[0.0, 0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, 0.0, ], ],
],
 [
   [[0.0, 0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, 0.0, ], ],
   [[0.0, 0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, 0.0, ], ],
   [[0.0, 0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, 0.0, ], ],
],
]
w_ujdk = [
 [
   [[0.0, 0.0, 0.0, ], [1.0, 0.0, 0.0, ], [0.0, 1.0, 0.0, ], [1.0, ], ],
   [[0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, ], [0.0, 1.0, 0.0, ], [1.0, ], ],
   [[1.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, ], [0.0, 0.0, 0.0, ], [1.0, ], ],
 ],
 [
   [[1.0, 0.0, 0.0, ], [1.0, 0.0, 0.0, ], [1.0, 0.0, 0.0, ], [0.0, ], ],
   [[0.0, 1.0, 0.0, ], [0.0, 1.0, 0.0, ], [0.0, 1.0, 0.0, ], [0.0, ], ],
   [[0.0, 0.0, 1.0, ], [0.0, 0.0, 1.0, ], [0.0, 0.0, 1.0, ], [0.0, ], ],
 ],
 [
   [[1.0, 0.0, 0.0, ], [1.0, 0.0, 0.0, ], [1.0, 0.0, 0.0, ], [0.0, ], ],
   [[0.0, 1.0, 0.0, ], [0.0, 1.0, 0.0, ], [0.0, 1.0, 0.0, ], [0.0, ], ],
   [[0.0, 0.0, 1.0, ], [0.0, 0.0, 1.0, ], [0.0, 0.0, 1.0, ], [0.0, ], ],
 ],
]
```

パターンノードと入力ノードの間の結合

変数 X と入力ノードの間の結合

変数 Y と入力ノードの間の結合

パターンマッチの実行例

ネットワークが保持するパターンと価値:

[X, 1, 2]=>0.2
[X, X, 2]=>0.1
[X, Y, 2]=>1

与える入力:

[1,1,2]

同時確率の高い順にソートした結果:

P="[X, 1, 2] => 0.2", X=1, Y=0 : JP=0.000200
P="[X, X, 2] => 0.1", X=1, Y=0 : JP=0.000100
P="[X, Y, 2] => 1", X=1, Y=1 : JP=0.000037
...

最も特殊なパターンのうち
最も価値の高いパターンが選択されている。

直観に反する振る舞いをする例

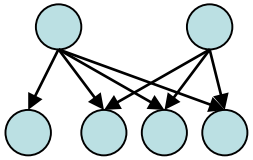
- 提案手法では、いくつか直観に反する振る舞いをする場合がある:
  - 価値の差が大きいとき
  - 値0が入力とパターンに含まれているとき
  - ワイルドカードが含まれているとき
- プログラム合成システムに用いるうえでは問題は生じないと現時点では考えている。今後、要検証。

人工データを使った教師あり学習の手順

パターンの記述

[1, X, X]=>1
[X, 2, X]=>1
[X, X, 3]=>1

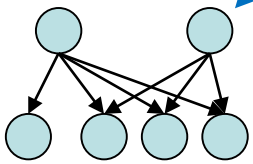
ベイジアンネットを生成



可能な値の組み合わせの
リストを生成 → 訓練データ

(1,1,1,1,1)
(1,2,1,2,2)
(1,3,1,3,3)
(1,0,1,0,0)
(2,1,1,2,1)
(2,2,2,2,2)
(2,3,3,2,3)
(2,0,0,2,3)
(3,1,1,1,3)
(3,2,2,2,3)
(3,3,3,3,3)
(3,0,0,0,3)

同じ構造を持ったネットワークを構築
パラメタの初期値はランダム



目的関数：負の対数尤度
モメンタム法で
1000エポック学習

学習結果

訓練データを生成した
ベイジアンネットの
条件付確率表

上の層の値を与えたときの
下の層の各ノードの条件付確率の表

Source network CPT:
[1, 0, 0, 0][1, 0, 0, 0]->[1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[1, 0, 0, 0][0, 1, 0, 0]->[1.0, 0.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[1, 0, 0, 0][0, 0, 1, 0]->[1.0, 0.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[1, 0, 0, 0][0, 0, 0, 1]->[1.0, 0.0, 0.0, 0.0,][0.0, 0.0, 0.0, 1.0,][0.0, 0.0, 0.0, 1.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][1, 0, 0, 0]->[1.0, 0.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][0, 1, 0, 0]->[0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][0, 0, 1, 0]->[0.0, 0.0, 1.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][0, 0, 0, 1]->[0.0, 0.0, 0.0, 1.0,][0.0, 1.0, 0.0, 0.0,][0.0, 0.0, 0.0, 1.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][1, 0, 0, 0]->[1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][0, 1, 0, 0]->[0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][0, 0, 1, 0]->[0.0, 0.0, 1.0, 0.0,][0.0, 0.0, 1.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][0, 0, 0, 1]->[0.0, 0.0, 0.0, 1.0,][0.0, 0.0, 0.0, 1.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]

学習後のベイジアンネットの
条件付確率表

Target network CPT:
[1, 0, 0, 0][1, 0, 0, 0]->[1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[1, 0, 0, 0][0, 1, 0, 0]->[1.0, 0.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[1, 0, 0, 0][0, 0, 1, 0]->[1.0, 0.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[1, 0, 0, 0][0, 0, 0, 1]->[1.0, 0.0, 0.0, 0.0,][0.0, 0.0, 0.0, 1.0,][0.0, 0.0, 0.0, 1.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][1, 0, 0, 0]->[1.0, 0.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][0, 1, 0, 0]->[0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][0, 0, 1, 0]->[0.0, 0.0, 1.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][0, 0, 0, 1]->[0.0, 0.0, 0.0, 1.0,][0.0, 1.0, 0.0, 0.0,][0.0, 0.0, 0.0, 1.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][1, 0, 0, 0]->[1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][0, 1, 0, 0]->[0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][0, 0, 1, 0]->[0.0, 0.0, 1.0, 0.0,][0.0, 0.0, 1.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][0, 0, 0, 1]->[0.0, 0.0, 0.0, 1.0,][0.0, 0.0, 0.0, 1.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]

安定して全く同一の条件付確率表が得られた。

訓練データを間引いた場合の学習結果

１２個の訓練データのうち
４個を間引いて学習

(1,1,1,1,1)
(1,2,1,2,2)
~~(1,3,1,3,3)~~
(1,0,1,0,0)
(2,1,1,2,1)
~~(2,2,2,2,2)~~
(2,3,3,2,3)
(2,0,0,2,3)
~~(3,1,1,1,3)~~
(3,2,2,2,3)
(3,3,3,3,3)
~~(3,0,0,0,3)~~

Source network CPT:
[1, 0, 0, 0][1, 0, 0, 0]->[1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[1, 0, 0, 0][0, 1, 0, 0]->[1.0, 0.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[1, 0, 0, 0][0, 0, 1, 0]->[1.0, 0.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[1, 0, 0, 0][0, 0, 0, 1]->[1.0, 0.0, 0.0, 0.0,][0.0, 0.0, 0.0, 1.0,][0.0, 0.0, 0.0, 1.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][1, 0, 0, 0]->[1.0, 0.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][0, 1, 0, 0]->[0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][0, 0, 1, 0]->[0.0, 0.0, 1.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][0, 0, 0, 1]->[0.0, 0.0, 0.0, 1.0,][0.0, 1.0, 0.0, 0.0,][0.0, 0.0, 0.0, 1.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][1, 0, 0, 0]->[1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][0, 1, 0, 0]->[0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][0, 0, 1, 0]->[0.0, 0.0, 1.0, 0.0,][0.0, 0.0, 1.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][0, 0, 0, 1]->[0.0, 0.0, 0.0, 1.0,][0.0, 0.0, 0.0, 1.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]

Target network CPT:
[1, 0, 0, 0][1, 0, 0, 0]->[1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[1, 0, 0, 0][0, 1, 0, 0]->[1.0, 0.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[1, 0, 0, 0][0, 0, 1, 0]->[0.9, 0.0, 0.0, 0.1,][0.0, 0.0, 1.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[1, 0, 0, 0][0, 0, 0, 1]->[1.0, 0.0, 0.0, 0.0,][0.0, 0.0, 0.0, 1.0,][0.0, 0.0, 0.0, 1.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][1, 0, 0, 0]->[1.0, 0.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][0, 1, 0, 0]->[0.0, 1.0, 0.0, 0.0,][0.0, 0.9, 0.0, 0.1,][0.0, 1.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][0, 0, 1, 0]->[0.0, 0.0, 1.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 1, 0, 0][0, 0, 0, 1]->[0.0, 0.0, 0.0, 1.0,][0.0, 1.0, 0.0, 0.0,][0.0, 0.0, 0.0, 1.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][1, 0, 0, 0]->[1.0, 0.0, 0.0, 0.0,][1.0, 0.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][0, 1, 0, 0]->[0.0, 1.0, 0.0, 0.0,][0.0, 1.0, 0.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][0, 0, 1, 0]->[0.0, 0.0, 1.0, 0.0,][0.0, 0.0, 1.0, 0.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]
[0, 0, 1, 0][0, 0, 0, 1]->[0.0, 0.0, 0.0, 1.0,][0.0, 0.0, 0.0, 1.0,][0.0, 0.0, 1.0, 0.0,][1.0, 0.0, 0.0, 0.0,]

安定してほぼ同一の条件付確率表が得られた。

訓練データを間引いた学習の意味

- (1,1,1) と (1,2,2) という2つの経験のみから
  パターン (1,X,X) という一般化された知識を獲得している。
  → 未経験のデータ (1,3,3) にも対処可能に。

- ニューラルネットワークのような値の「滑らかさ」を前提とした汎化とは違う種類の汎化。
  - 帰納論理プログラミングに近い、記号AIならではの強力な汎化能力。
  - 帰納論理プログラミングと違い、勾配法で逐次的に学習。
- 提案手法は、経験から汎用性の高い知識を自律的に獲得する知的エージェントの実現に役立つと考えられる。

まとめと今後

- ２層ベイジアンネットを用いて整数値ベクトル対するパターン獲得・パターンマッチを行う手法を提案
- 間引かれた訓練データだけから一般化された知識を獲得
- 今後の課題：
  - パターンの教師なし学習
    - 適切な正則化項を設定、EMアルゴリズムで学習
  - ネットワークの大規模化
    - 大規模化可能な確率伝搬アルゴリズムの開発
  - AGIのためのプログラム合成システムとして必要な機能の追加
    - 動的記憶域のように動作する連想記憶機構
    - 自然言語と知識の間の相互変換
    - 高階述語を用いたメタプログラミング
  - 同じ目的を持つ研究者を増やす
    - **脳型AGIアーキテクチャの最小構成モデルのデモにより、AGI実現が現実的であることを示すことが最優先課題**