Neural Programs: Towards Adaptive Control in Cyber-Physical Systems

Konstantin Selyunin[1], Denise Ratasich[1]
Ezio Bartocci[1], M.A. Islam[2]
Scott A. Smolka[2], Radu Grosu[1]

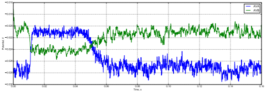
[1]Vienna University of Technology
Cyber-Physical Systems Group E182-1
Institut of Computer Engineering

[2]Stony Brook University, NY, USA

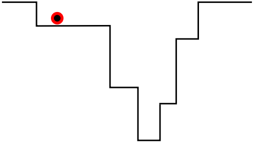
Motivation

- Programs are not robust

- Case studies: neural circuit simulation & parallel parking



- Parameter synthesis: plateaus are bad for optimizations



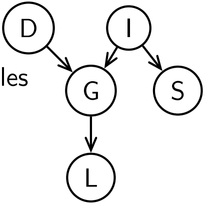
Motivation II

This presentation:

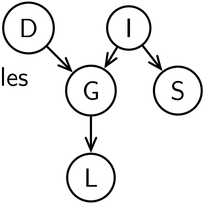
- How to incorporate “smooth” decisions in CPS to make systems more robust using neural circuits and GBN
- Technique to learn parameters of a model
- Application to two case studies and the relation between them

Background

- Bayesian Networks
  - express probabilistic dependencies between variables
  - are represented as DAGs
  - allow compact representation using CPDs



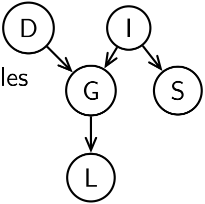
Background



- Bayesian Networks
  - express probabilistic dependencies between variables
  - are represented as DAGs
  - allow compact representation using CPDs
- Gaussian Distributions
  - Univariate and Multivariate Gaussian distributions

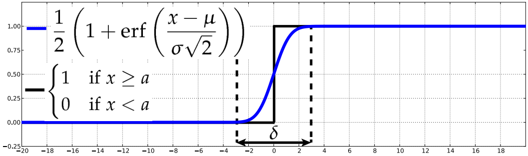
Background

- Bayesian Networks
  - express probabilistic dependencies between variables
  - are represented as DAGs
  - allow compact representation using CPDs



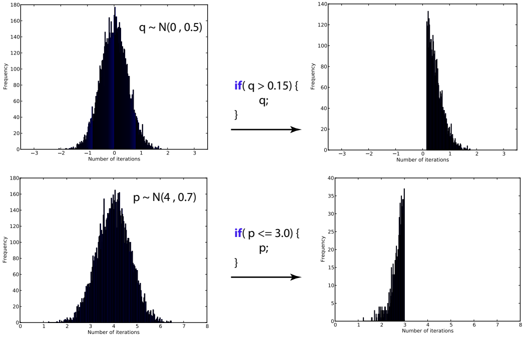
- Gaussian Distributions
  - Univariate and Multivariate Gaussian distributions

- Step function vs. sigmoid



Background

- Passing random variables through conditions



Towards the `nif` statement

Our setting:

- Program operates random variables (RVs)
- RVs are mutually dependent Gaussians

Towards the `nif` statement

Our setting:

- Program operates random variables (RVs)
- RVs are mutually dependent Gaussians

Questions:

- How to incorporate uncertainty of making a decision and make decisions "smooth"?
- How to avoid cutting distributions when passing a variable through a condition or a loop?

Towards the `nif` statement

Our setting:

- Program operates random variables (RVs)
- RVs are mutually dependent Gaussians

Questions:

- How to incorporate uncertainty of making a decision and make decisions “smooth”?
- How to avoid cutting distributions when passing a variable through a condition or a loop?

We propose to use `nif`s instead of traditional `if` statements.

Neural if

The nif statement: nif(x # y, σ^2)

- Inequality relation $\{\geq, >, <, \leq\}$
- Variance (represents our confidence of making a decision)

Example:

```
nif( x >= a, σ²) S1 else S2
```

$\texttt{nif(x \# a, } \sigma^2 \texttt{)}$: Evaluation

1. Compute the difference between $\texttt{x, a}$

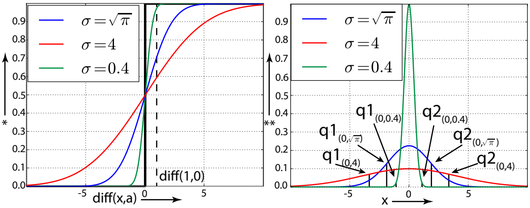
$$\texttt{diff(x,a)} = \begin{cases} \texttt{x - a} - \epsilon & \text{if \# is } >, \\ \texttt{x - a} & \text{if \# is } \geq, \\ \texttt{a - x} - \epsilon & \text{if \# is } <, \\ \texttt{a - x} & \text{if \# is } \leq . \end{cases}$$

nif(x # a, σ^2) : Evaluation

1. Compute the difference between x, a

$$\texttt{diff(x,a)} = \begin{cases} \texttt{x - a} - \epsilon & \text{if \# is } >, \\ \texttt{x - a} & \text{if \# is } \geq, \\ \texttt{a - x} - \epsilon & \text{if \# is } <, \\ \texttt{a - x} & \text{if \# is } \leq. \end{cases}$$

2. Compute quantiles of the probability density function

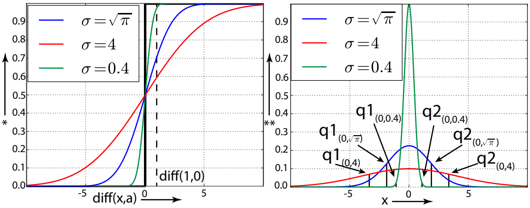


nif(x # a, σ^2) : Evaluation

1. Compute the difference between x, a

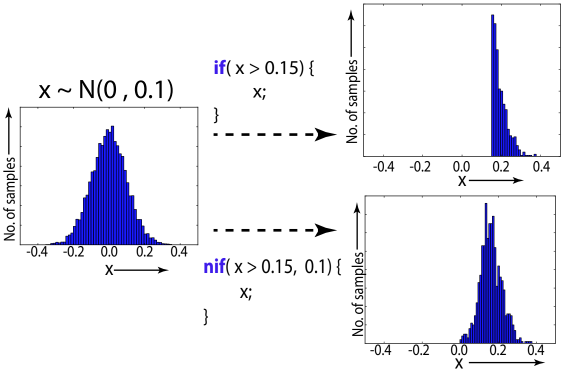
$$\texttt{diff(x,a)} = \begin{cases} \texttt{x - a} - \epsilon & \text{if } \# \text{ is } >, \\ \texttt{x - a} & \text{if } \# \text{ is } \geq, \\ \texttt{a - x} - \epsilon & \text{if } \# \text{ is } <, \\ \texttt{a - x} & \text{if } \# \text{ is } \leq . \end{cases}$$

2. Compute quantiles of the probability density function



3. Check if a random sample is within the interval

nif: Example


x ~ N(0 , 0.1)
No. of samples
-0.4 -0.2 0.0 0.2 0.4
x
if(x > 0.15) {
x;
}
nif(x > 0.15, 0.1) {
x;
}
No. of samples
-0.4 -0.2 0.0 0.2 0.4
x
No. of samples
-0.4 -0.2 0.0 0.2 0.4
x

Limit case $\sigma^2 \to 0$

```
nif( x >= a, σ²) S1 else S2
```

- For the case with "no uncertainty" ($\sigma^2 \to 0$) the PDF is expressed as the Dirac function:
  - $\delta(x) = +\infty$ if $x = 0$ else 0
  - $\int_{-\infty}^{\infty} \delta(x)\, dx = 1$

Limit case $\sigma^2 \to 0$

```
nif( x >= a, σ²) S1 else S2
```

- For the case with "no uncertainty" ($\sigma^2 \to 0$) the PDF is expressed as the Dirac function:
  - $\delta(x) = +\infty$ if $x = 0$ else 0
  - $\int_{-\infty}^{\infty} \delta(x)\, dx = 1$

- $\sigma^2 \to 0$: the `nif` statement is equivalent to the `if` condition

nwhile

Extension of a traditional `while` statement that incorporates uncertainty

$\texttt{nwhile}(\ \texttt{x} \# \texttt{a}, \sigma^2)\{P_1\}$

nwhile

Extension of a traditional `while` statement that incorporates uncertainty

`nwhile(` x $\#$ a, σ^2)$\{P_1\}$

Evaluation:

1. Compute `diff(x,a)`, obtain quantiles $q1$ and $q2$
2. Check if a random sample is within the interval
3. If sample within the interval, execute `P1` and go to 1, else exit

Case study 1: C.elegans

C.elegans

- a 1-mm round worm
- each adult individual has exactly 302 neurons
- extensively studied in evolutional- and neurobiology

Tap withdrawal response

- apply stimulus to mechanosensory (input) neurons
- observe the behavior: forward / backward movement

Goal

- express the behavior using neural program

Case study 1: C.elegans

C.elegans

- a 1-mm round worm
- each adult individual has exactly 302 neurons
- extensively studied in evolutional- and neurobiology

Tap withdrawal response

- apply stimulus to mechanosensory (input) neurons
- observe the behavior: forward / backward movement

Goal

- express the behavior using neural program

Case study 1: C.elegans

C.elegans

- a 1-mm round worm
- each adult individual has exactly 302 neurons
- extensively studied in evolutional- and neurobiology

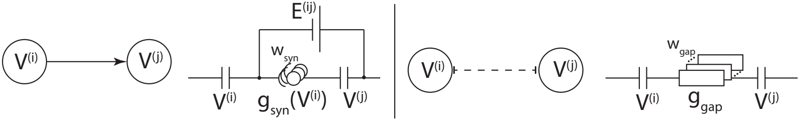
Tap withdrawal response

- apply stimulus to mechanosensory (input) neurons
- observe the behavior: forward / backward movement

Goal

- express the behavior using neural program

Neural connections 101



Synaptic connection

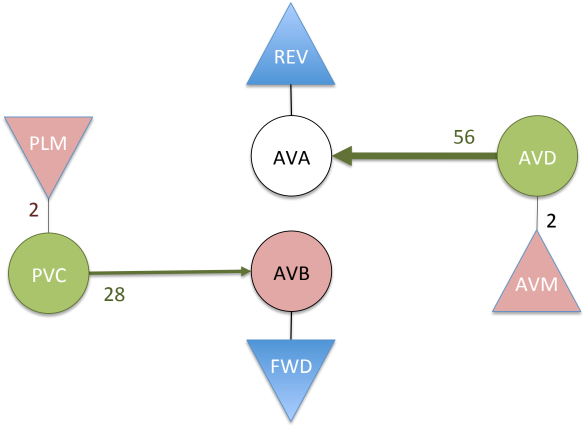
- chemical nature
- either active or not
- synaptic weight w_{syn}
- use `nif` to model each synaptic connection

Gap junction connection

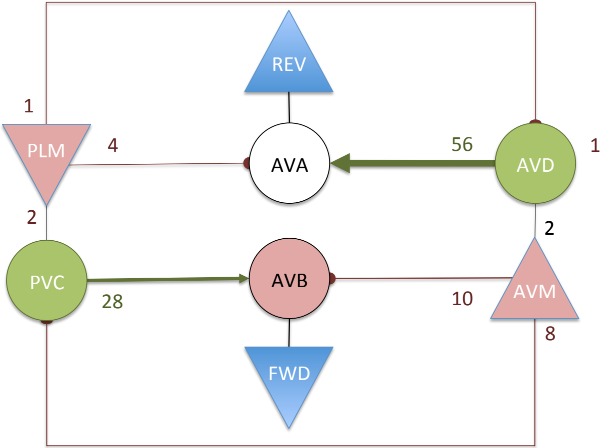
- instantaneous resistive connection
- linear combination of inputs
- gap junction weight w_{gap}

C.elegans Tap withdrawal circuit

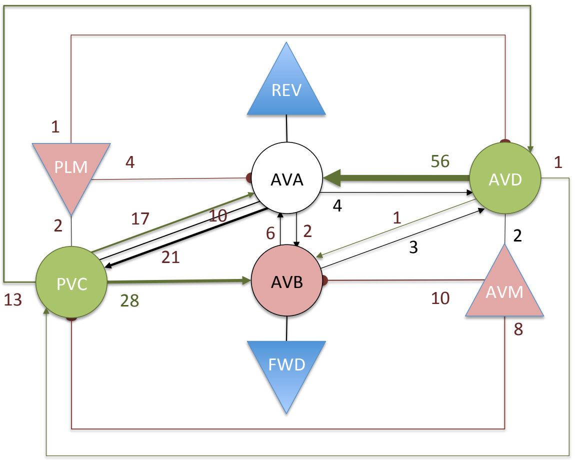
C.elegans Tap withdrawal circuit



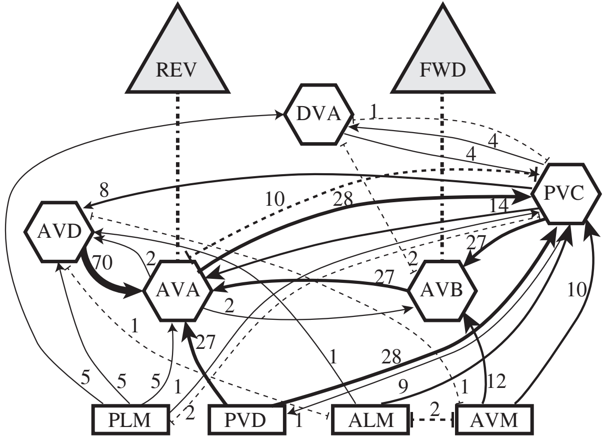
C.elegans Tap withdrawal circuit



C.elegans Tap withdrawal circuit



C.elegans Tap withdrawal circuit



Tap Withdrawal Simulations as Neural Program

Biological Model

$$\frac{dV^{(i)}}{dt} = \frac{V_{Leak} - V^{(i)}}{R_m^{(i)} C_m^{(i)}} + \frac{\sum_{j=1}^{N} (I_{syn}^{(ij)} + I_{gap}^{(ij)}) + I_{stim}^{(i)}}{C_m^{(i)}} \tag{1}$$

$$I_{gap}^{(ij)} = w_{gap}^{(ij)}\, g_{gap}^{(ij)}\, (V_j - V_i) \tag{2}$$

$$I_{syn}^{(ij)} = w_{syn}^{(ij)}\, g_{syn}^{(ij)}\, (E^{(ij)} - V^{(j)}) \tag{3}$$

$$g_{syn}^{(ij)}(V^{(j)}) = \frac{\bar{g}_{syn}}{1 + e^{K\left(\frac{V^{(j)} - V_{eq_j}}{V_{range}}\right)}} \tag{4}$$

Neural Program

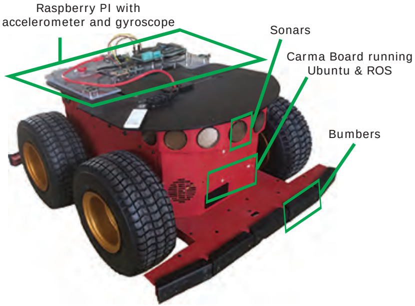
nwhile ($t \leq t_{dur}$, 0)
  compute $I_{gap}^{(ij)}$ using equation 2
  nwhile ($k \leq w_{syn}^{(ij)}$, 0)
    nif ($V^{(j)} \leq V_{eq}$, K/V_{range})
      $g_{syn}^{(ij)} \leftarrow g_{syn}^{(ij)} + g_{syn}$
  compute $I_{syn}^{(ij)}$ using equation 3
  compute $dV^{(i)}$ using equation 1
  $V^{(i)} \leftarrow V^{(i)} + dV^{(i)}$
  $t \leftarrow t + dt$

C.elegans Tap withdrawal simulations

Case study 2: Parallel parking

Given:

- P3AT-SH Pioneer rover
- Carma Devkit
- ROS on Ubuntu 12.04
- Pi with Gertboard



Goal:

Write a parallel parking controller as a neural program

Program skeleton

```
nwhile(currentDistance < targetLocation1, sigma1){
moving();
currentDistance = getPose();
}

updateTargetLocations();
nwhile(currentAngle < targetLocation2,    sigma2){
turning();
currentAngle = getAngle();
}

updateTargetLocations();
nwhile(currentDistance < targetLocation3, sigma3){
moving();
currentDistance = getPose();
}
...
```

Program skeleton

```
nwhile(currentDistance < targetLocation1, sigma1){
moving();
currentDistance = getPose();
}

updateTargetLocations();
nwhile(currentAngle < targetLocation2,    sigma2){
turning();
currentAngle = getAngle();
}

updateTargetLocations();
nwhile(currentDistance < targetLocation3, sigma3){
moving();
currentDistance = getPose();
}
...
```

Question: how to find the unknown parameters and how uncertain are we about each of them?

Learning

Parking example:

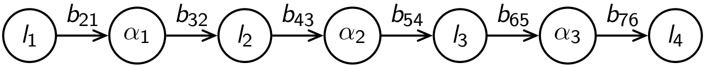
- Sequence of moves and turns
- Each action depends on the previous one
- The dependence is probabilistic
- RVs are normally distributed (assumption)

Learning

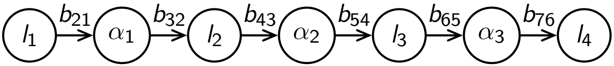
Parking example:

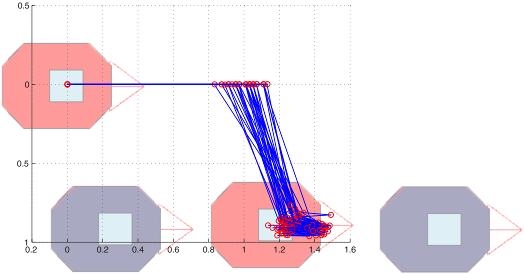
- Sequence of moves and turns
- Each action depends on the previous one
- The dependence is probabilistic
- RVs are normally distributed (assumption)

Gaussian Bayesian Network:



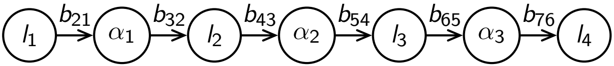
Learning: Good traces

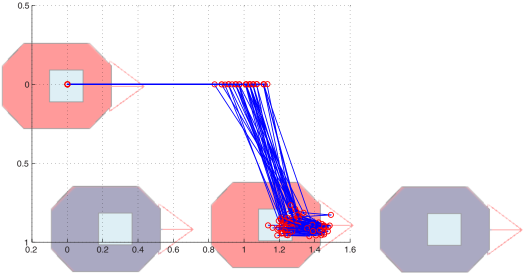




Task: learn the parameters of the GBN from the good traces

Learning: Good traces





Task: learn the parameters of the GBN from the good traces

1. Convert the GBN to the MGD[HG95]
2. Update the precision matrix **T** of the MGD[Nea03]
3. Extract σ^2s and b_{ij}s from **T**

Learning: Update step

- Iterative learning procedure
- Incrementally update mean μ and covariance matrix β of the prior
- Mean update:

$$\bar{\mathbf{x}} = \frac{\sum_{h=1}^{M} \mathbf{x}^{(h)}}{M}$$

$$\mu^* = \frac{v\mu + M\bar{\mathbf{x}}}{v + M}$$

Learning: Update step

- Iterative learning procedure
- Incrementally update mean μ and covariance matrix β of the prior
- Mean update:

$$\overline{\mathbf{x}} = \frac{\sum_{h=1}^{M} \mathbf{x}^{(h)}}{M}$$

$$\mu^* = \frac{v\mu + M\overline{\mathbf{x}}}{v + M}$$

- Covariance matrix update:

$$\mathbf{s} = \sum_{h=1}^{M} \left(\mathbf{x}^{(h)} - \overline{\mathbf{x}}\right)\left(\mathbf{x}^{(h)} - \overline{\mathbf{x}}\right)^T$$

$$\beta^* = \beta + s + \frac{vM}{v + M}\left(\mathbf{x}^{(h)} - \overline{\mathbf{x}}\right)\left(\mathbf{x}^{(h)} - \overline{\mathbf{x}}\right)^T$$

$$\left(\mathbf{T}^*\right)^{-1} \sim \beta^*$$

Learning: Data

"Good trajectories":

- `*.bag` files (collection of messages that are broadcasted in ROS)

  ↓
- extract coordinates in the 2-D space and angle

  ↓
- find important points

  ↓
- obtain samples in the form: $l_1, \alpha_1, l_2, \alpha_2, l_3, \alpha_3, l_4$

⋮	⋮	⋮
1.204	-0.911	-1.221
1.207	-0.920	-1.221
1.209	-0.927	-1.221
1.211	-0.930	-1.221
1.211	-0.931	-1.221
1.211	-0.931	-1.221
1.211	-0.931	-1.221
1.211	-0.931	-1.215
⋮	⋮	⋮

Learning: Data

"Good trajectories":

- *.bag files (collection of messages that are broadcasted in ROS)

↓

- extract coordinates in the 2-D space and angle

↓

- find important points

↓

- obtain samples in the form: $l_1, \alpha_1, l_2, \alpha_2, l_3, \alpha_3, l_4$

⋮	⋮	⋮
1.204	-0.911	-1.221
1.207	-0.920	-1.221
1.209	-0.927	-1.221
1.211	-0.930	-1.221
1.211	-0.931	-1.221
1.211	-0.931	-1.221
1.211	-0.931	-1.221
1.211	-0.931	-1.215
⋮	⋮	⋮

Learning: Data

"Good trajectories":

- $*$.bag files (collection of messages that are broadcasted in ROS)

  ↓
- extract coordinates in the 2-D space and angle

  ↓
- find important points

  ↓
- obtain samples in the form: $l_1, \alpha_1, l_2, \alpha_2, l_3, \alpha_3, l_4$

⋮	⋮	⋮
1.204	-0.911	-1.221
1.207	-0.920	-1.221
1.209	-0.927	-1.221
1.211	-0.930	-1.221
1.211	-0.931	-1.221
1.211	-0.931	-1.221
1.211	-0.931	-1.221
1.211	-0.931	-1.215
⋮	⋮	⋮

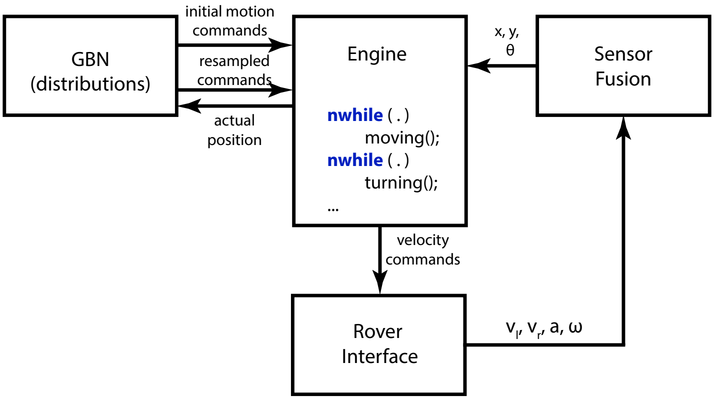
Learning: Data

"Good trajectories":

- *.bag files (collection of messages that are broadcasted in ROS)
  ↓
- extract coordinates in the 2-D space and angle
  ↓
- find important points
  ↓
- obtain samples in the form: $l_1, \alpha_1, l_2, \alpha_2, l_3, \alpha_3, l_4$

⋮	⋮	⋮
1.204	-0.911	-1.221
1.207	-0.920	-1.221
1.209	-0.927	-1.221
1.211	-0.930	-1.221
1.211	-0.931	-1.221
1.211	-0.931	-1.221
1.211	-0.931	-1.221
1.211	-0.931	-1.215
⋮	⋮	⋮

Parking system architecture



Conclusion and Future Work

Recap:

- use of smooth *Probit* distribution in conditional and loop statements
- use of Gaussian Bayesian Network to capture dependencies between *Probit* distributions
- Case studies: robust parking controller and tap withdrawal simulation

Future work:

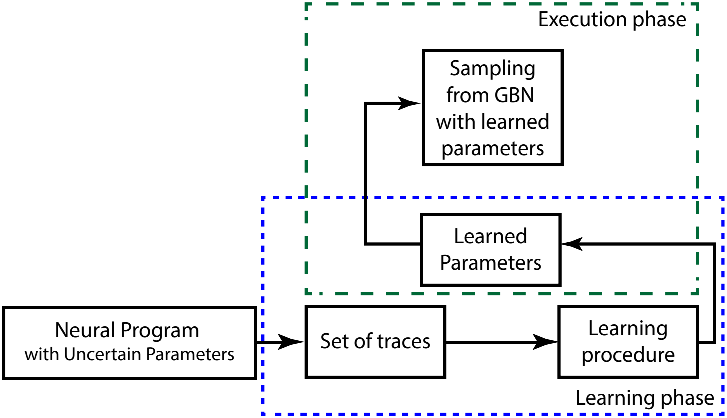
- Apply these techniques to monitoring

References I

David Heckerman and Dan Geiger.
Learning bayesian networks: A unification for discrete and gaussian domains.
In *UAI*, pages 274–284, 1995.

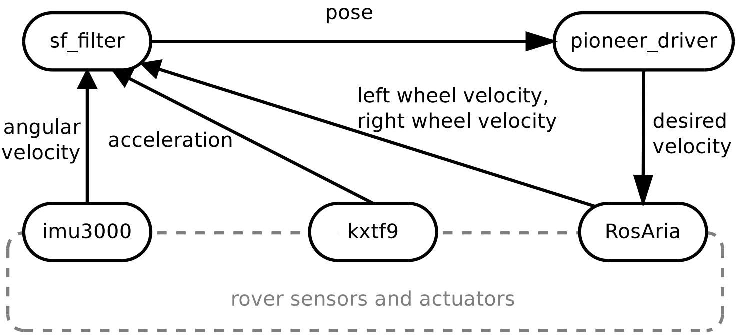
Richard E. Neapolitan.
Learning Bayesian Networks.
Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003.

Learning Parameters in a Neural Program



Integration into ROS

- Rover Interface
- Sensor Fusion
- GBN and Engine



Denotational Semantics

$$E ::= \quad \mathtt{x}_i \mid c \mid bop(E_1, E_2) \mid uop(E_1)$$

$$S ::= \quad \mathtt{skip} \mid \mathtt{x}_i := E \mid S_1; S_2 \mid \mathtt{nif}(\, \mathtt{x}_i \# c, \sigma^2 \,)\ S_1\ \mathtt{else}\ S_2 \mid \mathtt{nwhile}(\, \mathtt{x}_i \# c,\ \sigma^2)\{\ S_1\ \}$$

Denotational Semantics

$$[\![\texttt{skip}]\!](\mathrm{x}) = \mathrm{x}$$

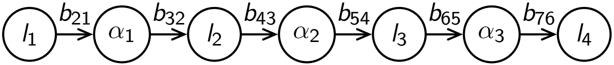
$$[\![\mathrm{x}_i := E\ ]\!](\mathrm{x}) = \mathrm{x}[[\![E]\!](\mathrm{x}) \mapsto \mathrm{x}_i]$$

$$[\![\ S_1; S_2\ ]\!](\mathrm{x}) = [\![S_2]\!]([\![S_1]\!](\mathrm{x}))$$

$$\begin{aligned}[\![\texttt{nif}(\,\mathrm{x}_i \# c, \sigma^2)\ S_1\ \texttt{else}\ S_2]\!](\mathrm{x}) = \\ [\![\texttt{check}(\mathrm{x}_i, a, \sigma^2, \#)]\!](\mathrm{x})[\![S_1]\!](\mathrm{x}) \quad + \\ [\![\neg\texttt{check}(\mathrm{x}_i, a, \sigma^2, \#)]\!](\mathrm{x})[\![S_2]\!](\mathrm{x})\end{aligned}$$

$$\begin{aligned}[\![\texttt{nwhile}(\,\mathrm{x}_i \# c,\ \sigma^2)\{\ S_1\ \}]\!](\mathrm{x}) = \\ \mathrm{x}[\![\neg\texttt{check}(\mathrm{x}_i, a, \sigma^2, \#\ )]\!](\mathrm{x}) \quad + \\ [\![\texttt{check}(\mathrm{x}_i, a, \sigma^2, \#\ )]\!](x)[\![\texttt{nwhile}(\,\mathrm{x}_i \# c,\ \sigma^2)\{\ S_1\ \}]\!]([\![S_1]\!]\mathrm{x})\end{aligned}$$

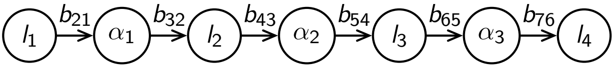
Learning: Conversion step



1. Define an ordering starting from initial node

$$t_1 = \frac{1}{\sigma_1^2} \qquad \mathbf{b}_i = \begin{pmatrix} b_{i1} \\ \vdots \\ b_{i,i-1} \end{pmatrix} \qquad \mu = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_n \end{pmatrix}$$

Learning: Conversion step



1. Define an ordering starting from initial node

$$t_1 = \frac{1}{\sigma_1^2} \qquad \mathbf{b}_i = \begin{pmatrix} b_{i1} \\ \vdots \\ b_{i,i-1} \end{pmatrix} \qquad \mu = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_n \end{pmatrix}$$

2. Use iterative algorithm from [Heckerman and Geiger, 1995]:

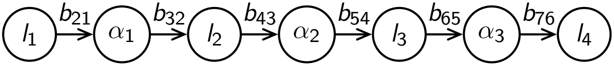
$$\mathbf{T}_1 = (t_1);$$

$$\textbf{for}(i = 2; i \leq n; i++)$$

$$\mathbf{T}_i = \begin{pmatrix} \mathbf{T}_{i-1} + t_i \mathbf{b}_i \mathbf{b}_i^T & -t_i \mathbf{b}_i \\ -t_i \mathbf{b}_i^T & t_i \end{pmatrix};$$

$$\mathbf{T} = \mathbf{T}_n;$$

Learning: Conversion step



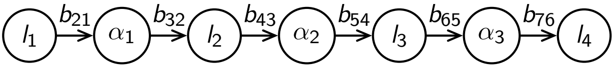
- l_1, α_1:

$$\mathbf{T}_1 = (t_1) = \frac{1}{\sigma_1^2};$$

$$\mathbf{b}_2 = \begin{pmatrix} b_{21} \end{pmatrix}$$

$$\mathbf{T}_2 = \begin{pmatrix} \frac{1}{\sigma_1^2} + \frac{b_{21}^2}{\sigma_2^2} & -\frac{b_{21}}{\sigma_2^2} \\ -\frac{b_{21}}{\sigma_2^2} & \frac{1}{\sigma_2^2} \end{pmatrix}$$

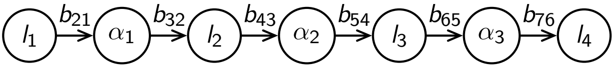
Learning: Conversion step



- l_2:

$$\mathbf{T}_3 = \begin{pmatrix} \frac{1}{\sigma_1^2} + \frac{b_{21}^2}{\sigma_2^2} & -\frac{b_{21}}{\sigma_2^2} & 0 \\ -\frac{b_{21}}{\sigma_2^2} & \frac{1}{\sigma_2^2} + \frac{b_{32}^2}{\sigma_3^2} & -\frac{b_{32}}{\sigma_3^2} \\ 0 & -\frac{b_{32}}{\sigma_3^2} & \frac{1}{\sigma_3^2} \end{pmatrix} \qquad \mathbf{b}_3 = \begin{pmatrix} 0 \\ b_{21} \end{pmatrix}$$

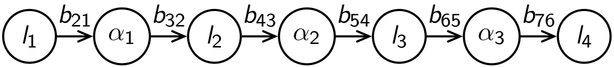
Learning: Conversion step



- α_2:

$$\mathbf{T}_4 = \begin{pmatrix} \frac{1}{\sigma_1^2} + \frac{b_{21}^2}{\sigma_2^2} & -\frac{b_{21}}{\sigma_2^2} & 0 & 0 \\ -\frac{b_{21}}{\sigma_2^2} & \frac{1}{\sigma_2^2} + \frac{b_{32}^2}{\sigma_3^2} & -\frac{b_{32}}{\sigma_3^2} & 0 \\ 0 & -\frac{b_{32}}{\sigma_3^2} & \frac{1}{\sigma_3^2} + \frac{b_{43}^2}{\sigma_4^2} & -\frac{b_{43}}{\sigma_4^2} \\ 0 & 0 & -\frac{b_{43}}{\sigma_4^2} & \frac{1}{\sigma_4^2} \end{pmatrix} \qquad \mathbf{b}_4 = \begin{pmatrix} 0 \\ 0 \\ b_{43} \end{pmatrix}$$

Learning: Conversion step



- l_3:

$$\mathbf{T}_5 = \begin{pmatrix} \frac{1}{\sigma_1^2} + \frac{b_{21}^2}{\sigma_2^2} & -\frac{b_{21}}{\sigma_2^2} & 0 & 0 & 0 \\ -\frac{b_{21}}{\sigma_2^2} & \frac{1}{\sigma_2^2} + \frac{b_{32}^2}{\sigma_3^2} & -\frac{b_{32}}{\sigma_3^2} & 0 & 0 \\ 0 & -\frac{b_{32}}{\sigma_3^2} & \frac{1}{\sigma_3^2} + \frac{b_{43}^2}{\sigma_4^2} & -\frac{b_{43}}{\sigma_4^2} & 0 \\ 0 & 0 & -\frac{b_{43}}{\sigma_4^2} & \frac{1}{\sigma_4^2} + \frac{b_{54}^2}{\sigma_5^2} & -\frac{b_{54}}{\sigma_5^2} \\ 0 & 0 & 0 & -\frac{b_{54}}{\sigma_5^2} & \frac{1}{\sigma_5^2} \end{pmatrix} \quad \mathbf{b}_5 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ b_{54} \end{pmatrix}$$

We can generalize T for arbitrary number of moves