Unconventional computing paradigms based on belief propagation

Alister Burr

Communications Research Group,
Dept. of Electronics
University of York

alister@ohm.york.ac.uk

THE UNIVERSITY *of York*

Motivation

- Follows on from Susan Stepney's talk two weeks ago on "Material Computation"
  - as I understood it, this was a plea to choose computation tasks appropriate to the properties of the material substrate used for unconventional computation
  - rather than to "force" the substrate to conform to the "classical" logic-based computation paradigm
- Andy Tyrrell also suggested that silicon might be such a material
- In any case, we still need a fairly general paradigm to describe the computation that a given material can perform
  - even if it is not the classical paradigm
  - otherwise our system may only be able to do one task
- Accordingly I'd like to describe a paradigm that I believe is sufficiently general
  - and which can be neatly implemented by exploiting the properties of the PN junction in silicon
- None of this is particularly new!

Outline

- Motivation
- **Introduction to “belief propagation”**
- Applications of belief propagation
- Implementation *in silico*
- Conclusions
- Bibliography

Belief Propagation

- a.k.a *message passing*, the *sum-product algorithm*
- It can be regarded as a general method for statistical inference
- We can use it to find the probabilities of a set of dependent (discrete-valued) variables,
  - given some probabilities
  - and functions describing their relationship
- Allows a potentially very complicated problem to be decomposed into a set of, very simple, interlinked problems which can be solved in a distributed fashion
- Will begin with a very simple, completely abstract example
  - then proceed to some more concrete application examples

Example

$$P(x_1,x_2,x_3)=f_1(x_1)f_2(x_2)f_3(x_3)f_4(x_1,x_2,x_3)$$

$$f_1(x_1)=\begin{cases}0.9 & x_1=0\\ 0.1 & x_1=1\end{cases}\quad f_2(x_2)=\begin{cases}0.1 & x_2=0\\ 0.9 & x_2=1\end{cases}\quad f_3(x_3)=\begin{cases}0.9 & x_2=0\\ 0.1 & x_2=1\end{cases}$$

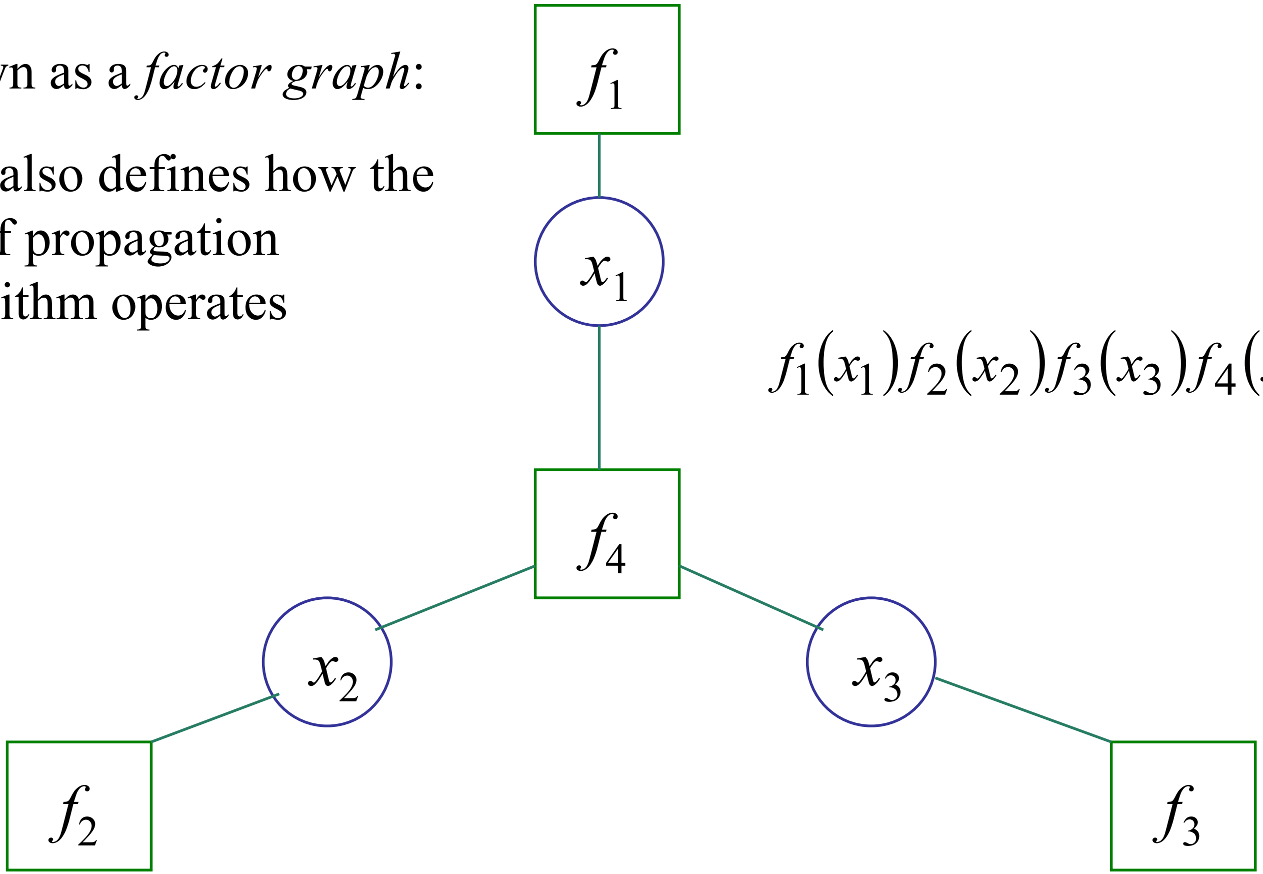
$$f_4(x_1,x_2,x_3)=\begin{cases}1 & x_1=x_2\ \&\ x_3=1\\ 1 & x_1\neq x_2\ \&\ x_3=0\\ 0 & \text{otherwise}\end{cases}$$

- Function values f_1, f_2, f_3, give probabilities of x_1, x_2, x_3
- Function f_4 gives a constraint on the values:
  - if x_1 and x_2 are the same, then x_3 must be zero
  - if different, x_3 must be one

THE UNIVERSITY *of York*

Factor graph

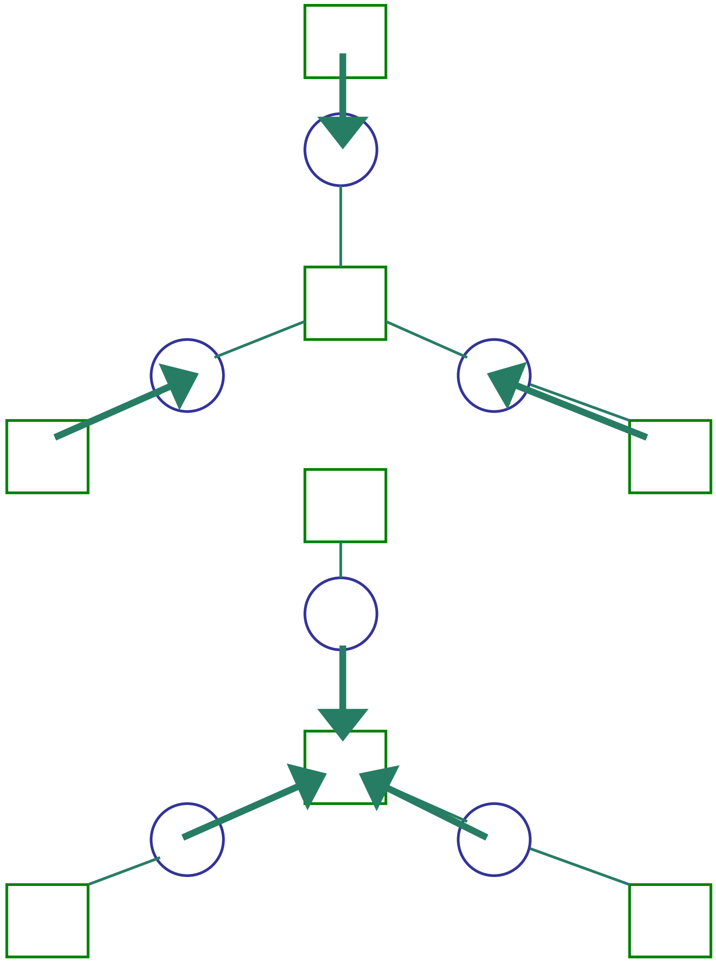
- We can conveniently describe the relationship between these variables by means of a simple graph,
- known as a *factor graph*:
- This also defines how the belief propagation algorithm operates



$$f_1(x_1)f_2(x_2)f_3(x_3)f_4(x_1,x_2,x_3)$$

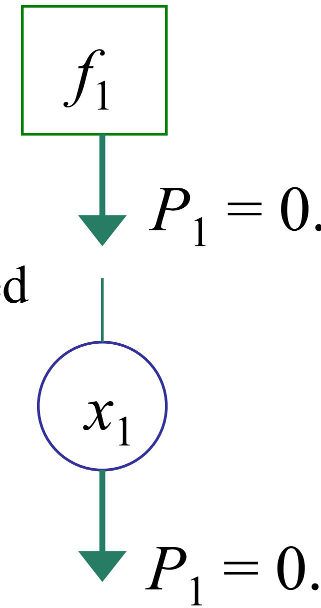
Belief propagation

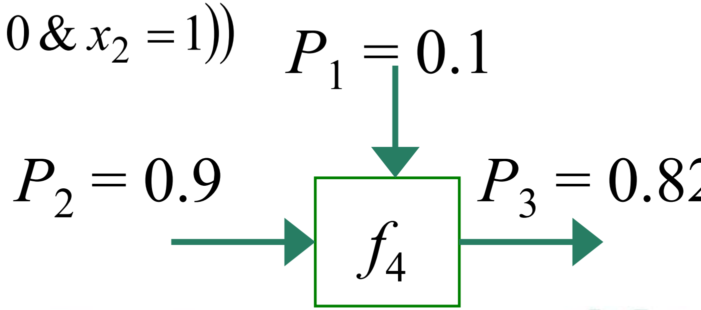
- The principle of belief propagation is that each node passes messages based on its knowledge of the variables along the edges of the factor graph
- Messages start at “leaf” nodes (nodes with only one branch connecting to them)
- Other nodes forward messages on the remaining connecting branch as soon as they have received messages on all the other branches
- Where a node has *n* branches, and it receives messages on all of them, it then sends a message on each, about the relevant variable, computed from the other *n*-1 messages



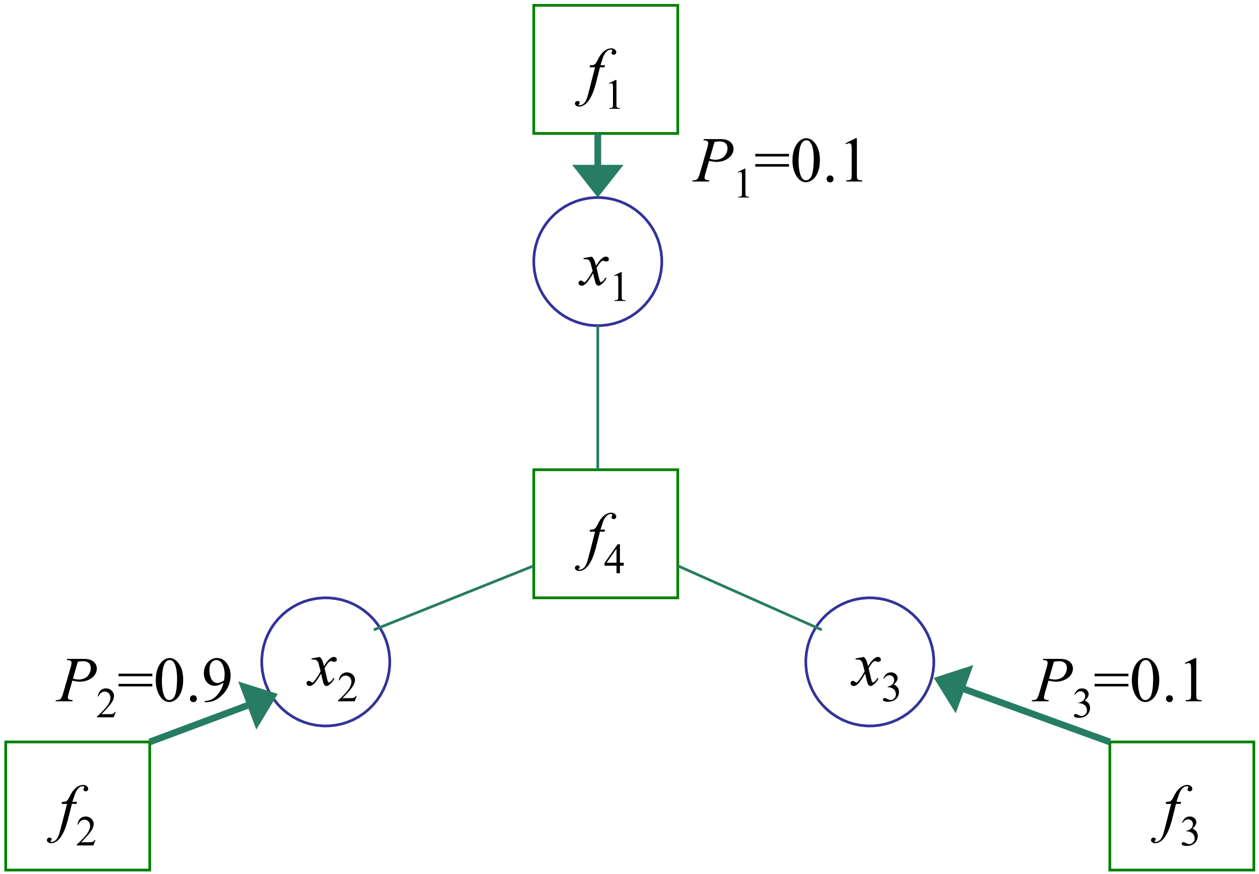
Calculating messages

- “Leaf” nodes: send the values of the corresponding function (i.e. the probabilities)
  - e.g. f_1 sends $P(x_1=\text{‘1’}) = P_1 = 0.1$
- Variable nodes: in this case just forward the message received on the incoming branch
  - if there is more than one incoming branch, forward the product of the messages on them
- Function nodes (other than leaves): on each branch send the probability of each variable calculated given the probabilities of the other variables incoming on the other branches
  - e.g. $$\begin{aligned} P(x_3=1) &= P\big((x_1=1\,\&\,x_2=0)\,\|\,(x_1=0\,\&\,x_2=1)\big) \\ &= P_1(1-P_2)+P_2(1-P_1) \\ &= 0.1\times0.1+0.9\times0.9=0.82 \\ &= \sum_{x_1',x_2'} f_4(x_1',x_2')\prod_{i=1,2} P(x_i=x_i') \end{aligned}$$

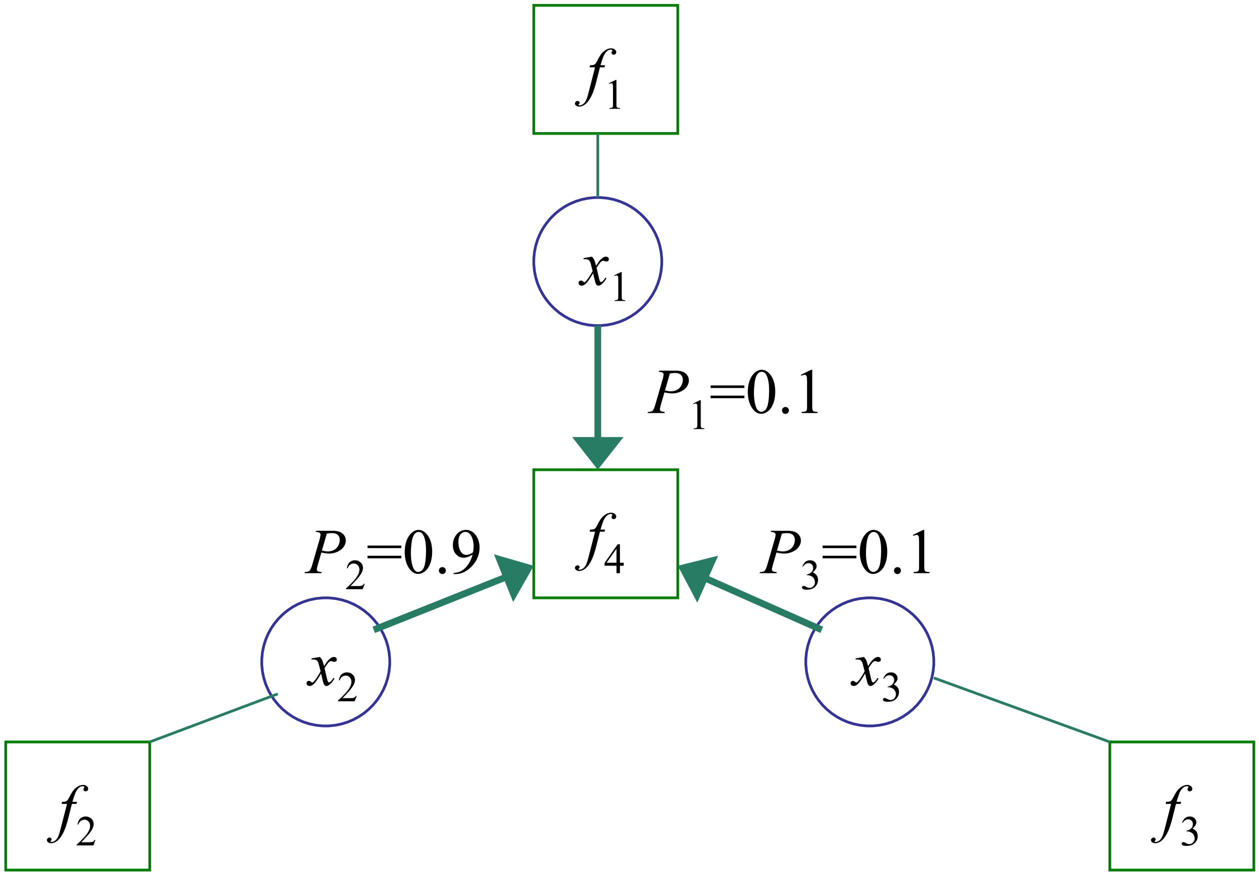




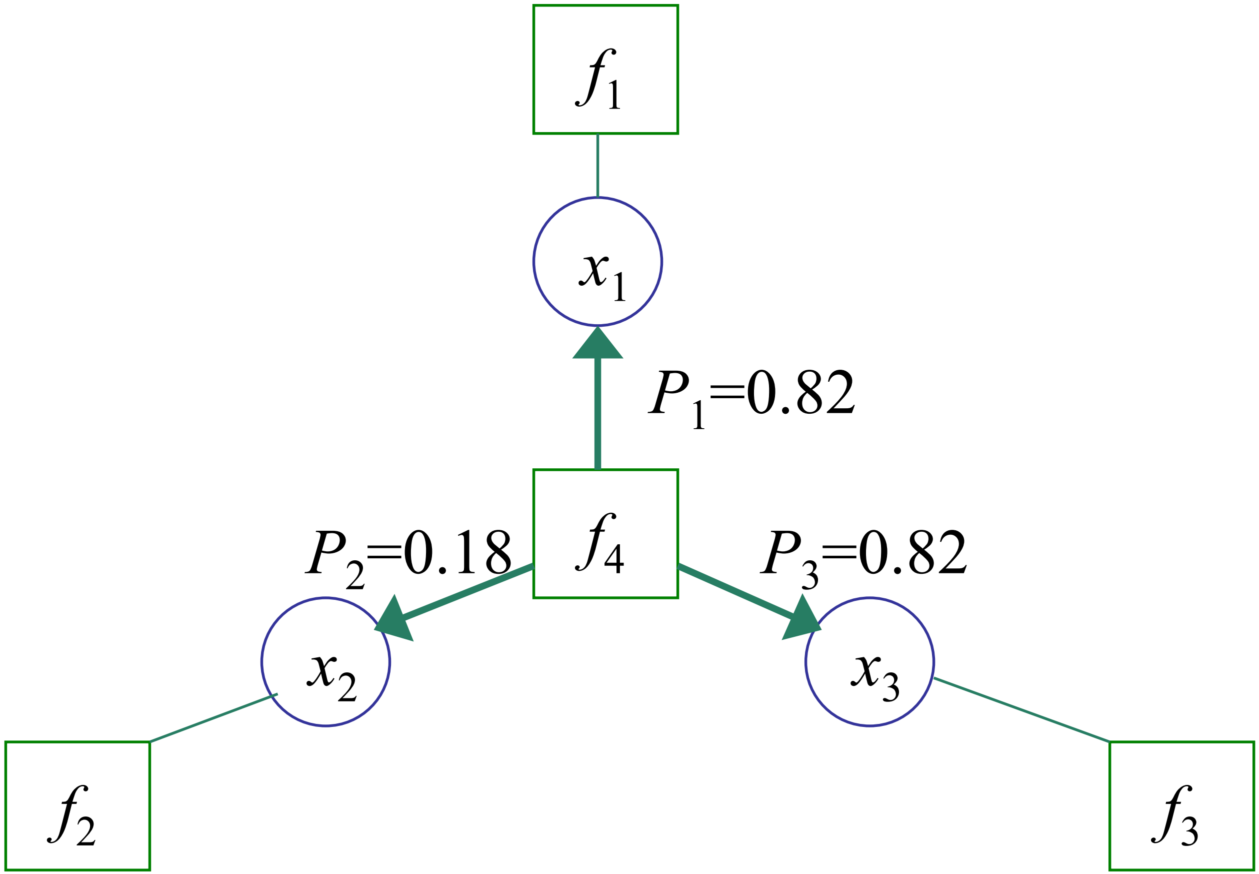
Example: stage 1



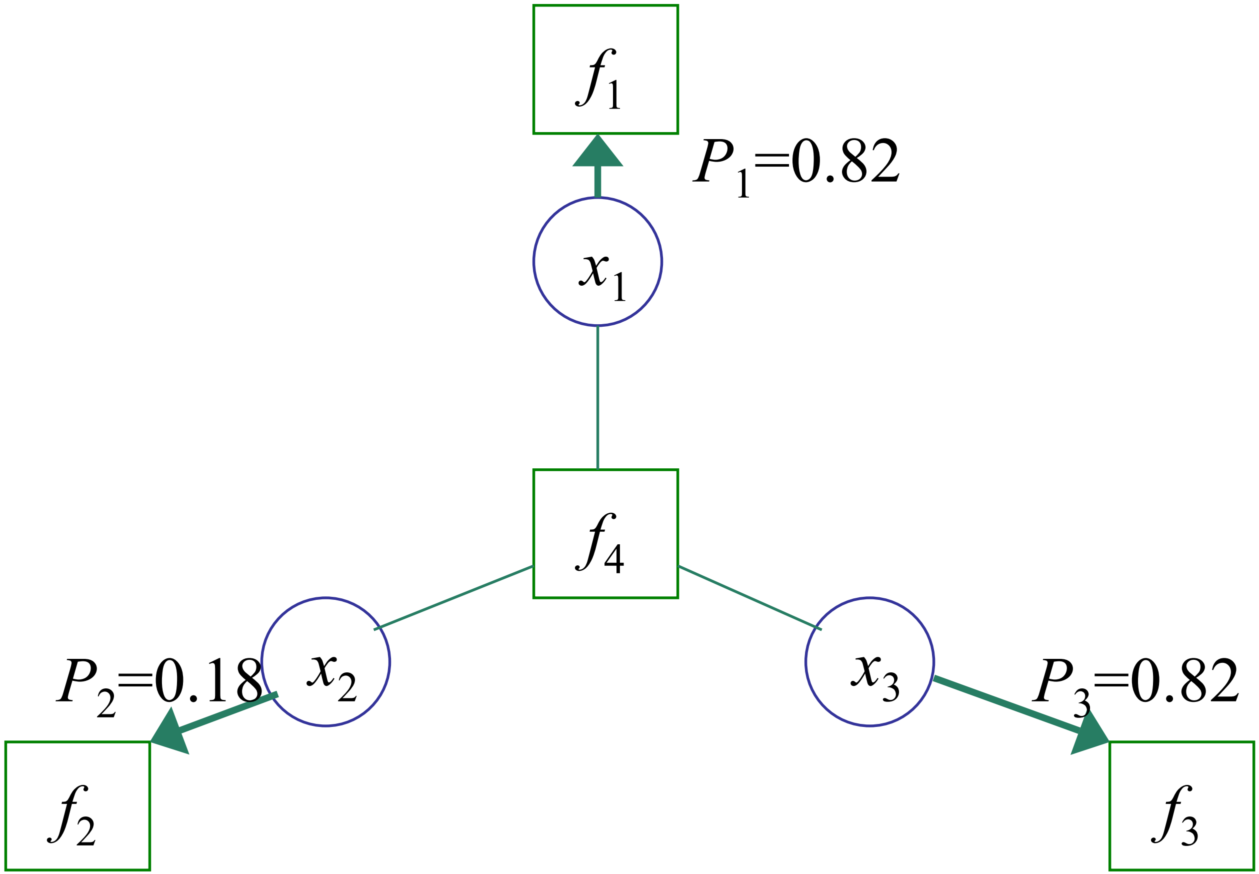
Example: stage 2



Example: stage 3



Termination



- We can then read off the marginal probability distributions of x_1, x_2 and x_3:
  - $P(x_1 = 1) = 0.82$; $P(x_2 = 1) = 0.18$; $P(x_3 = 1) = 0.82$;

Log probabilities

- In general the forwarding rule at factor nodes involves the sum and product of probabilities
  - hence the name *sum-product algorithm*
- However products are relatively complex to calculate
  - also probabilities typically require a large precision to represent small probabilities
  - hence it is convenient to use log probability in the messages:

$$L_i = \log(P_i)$$

- This means that products become sums:

$$\prod_i P_i \Rightarrow \sum_i L_i$$

- However sums become log of sum of exponentials:

$$\sum_i P_i \Rightarrow \log\left(\sum_i e^{L_i}\right)$$

Outline

- Motivation
- Introduction to “belief propagation”
- **Applications of belief propagation**
- Implementation *in silico*
- Conclusions
- Bibliography

Error correction decoding

- *This is the application I know about!*
- Here the variables are data and code bits
- The constraints are set by the code
  - in that only certain combinations of code bits (i.e. certain *codewords*) are permissible
- The task of the decoder is to find the most likely data, given a received codeword
  - which might contain errors

The example

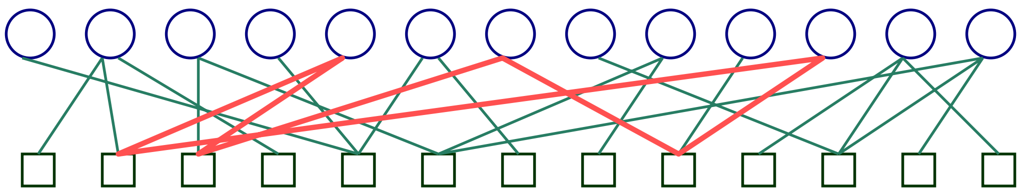
- The example is a particularly simple code, defined by f_4:
  - this constraint means that there is always an even number of '1's among the variables x_1, x_2 and x_3
  - i.e. this is an even parity code, in which x_3 is the parity check on the data bits x_1, x_2

$$f_4(x_1, x_2, x_3) = \begin{cases} 1 & x_1 = x_2 \ \& \ x_3 = 1 \\ 1 & x_1 \neq x_2 \ \& \ x_3 = 0 \\ 0 & \text{otherwise} \end{cases}$$

- Functions f_1, f_2 and f_3 assume that:
  - the received word is 010
  - the channel has an error probability of 0.1
- The result of the belief propagation algorithm ($P_1 = 0.18$, $P_2 = 0.82$) says that the most likely data bits are 0, 1 (not surprisingly)

More complicated codes

- Real error control codes used in modern communication systems (e.g. turbo-codes and LDPC codes used in 3G, WiFi, digital broadcast) are much larger
  - e.g. may have 10 000 code bits per codeword
  - 'brute force' decoding algorithms (e.g. exhaustive search) tend to have complexity exponential on the code length
- However they can still be described in terms of a (large) set of parity constraints (like f_4), which each involve only a few bits



- Belief propagation over the factor graph then provides a solution in feasible complexity
- Note however that there may be 'loops' in the graph
  - requires *iterative* version of belief propagation
  - not guaranteed to converge

Other applications

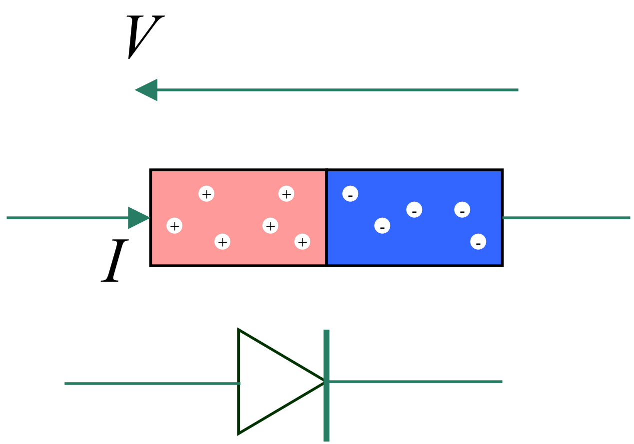
- Variables could have other significance:
  - Hypotheses (true/false)
  - Entities in a recognition problem (from a restricted set)
    - e.g. phonemes/phones
    - objects in a picture
    - concepts in natural language comprehension
  - States of a system
    - e.g. represented by a hidden Markov model
- Constraints can be generalised
  - e.g. to joint probability distributions
- Fundamentally a probabilistic technique
  - good at coping with uncertainty
  - also fundamentally about distributed processing
- May be a model for operation of some biological processes, or other complex systems
- Note that concept initially came from AI

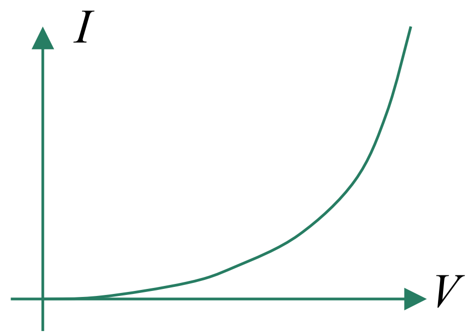
Outline

- Motivation
- Introduction to “belief propagation”
- Applications of belief propagation
- **Implementation *in silico***
- Conclusions
- Bibliography

Implementing log-sum-exp

- We have proposed replacing probabilities with logs of probabilities
  - converts products to sums
  - converts sums to "log-sum-exp": $\sum_i P_i \Rightarrow \log\left(\sum_i e^{L_i}\right)$
- This requires exponential conversion: expensive in conventional digital computation
- However in analogue, we can exploit the characteristics of a silicon PN junction:

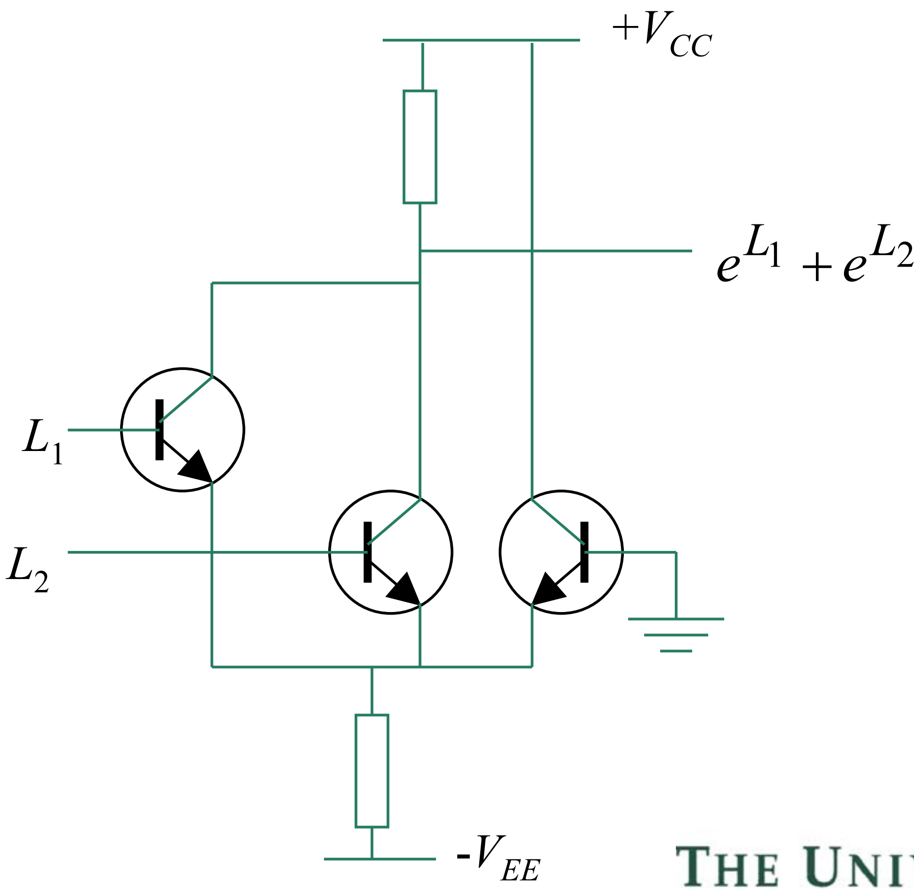




$$I = I_S\left(e^{V/V_T} - 1\right)$$

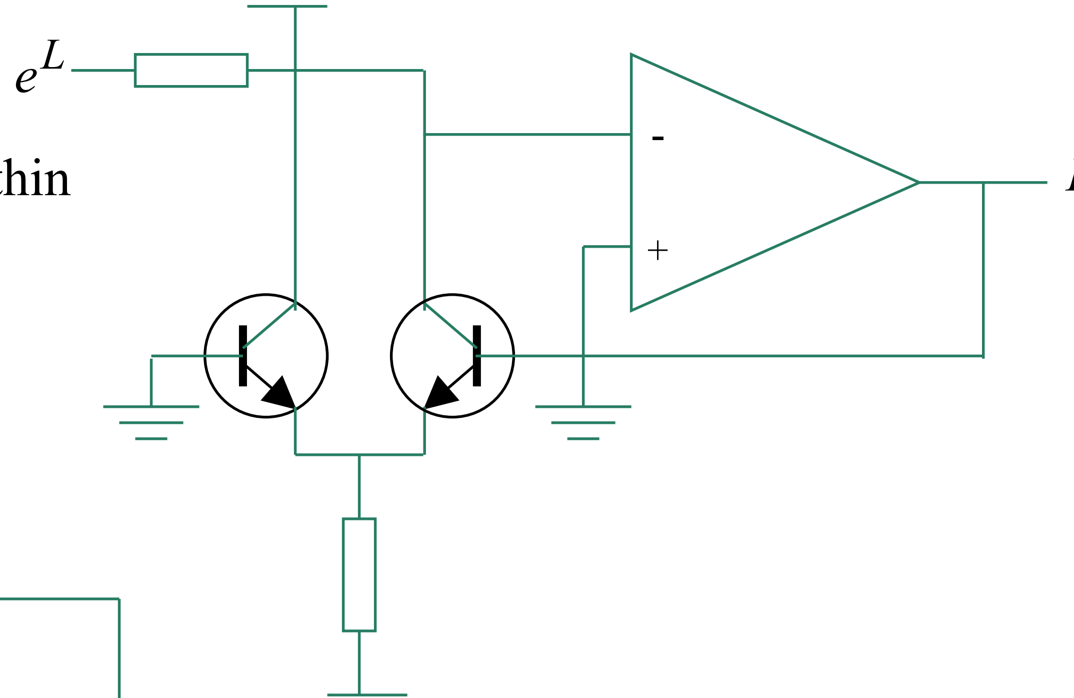
More practically

- In practice it is better to use a (bipolar) transistor
  - the same relationship applies to the collector current *versus* base-emitter voltage characteristic

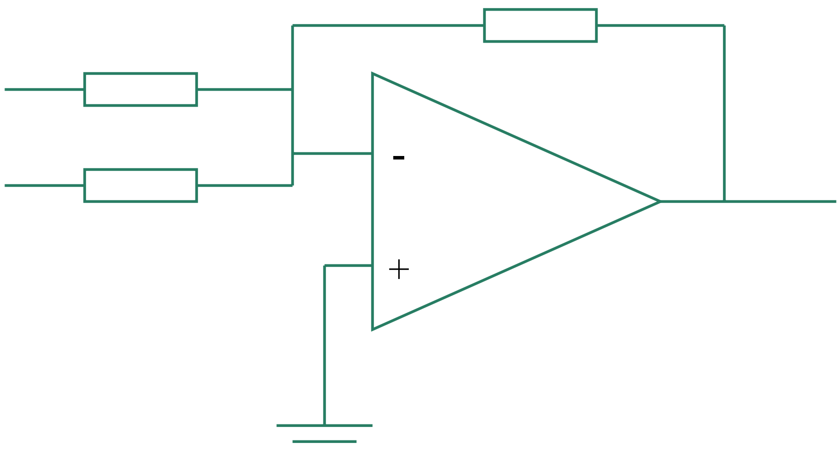


Log conversion and sum

- Log conversion can be implemented using the exponential converter within the feedback loop of an operational amplifier:



- and summation with a simple op. amp. summer:



Complexity

- Note that an operational amplifier contains in the same order of the number of transistors as a logic gate
- also that while a digital implementation of the sum-product operation might require at least 8 bit precision, this analogue implementation uses only one signal
- The number of sum terms in a sum-product operation in a factor node involving n binary variables is (in general) 2^{n-1}
  - and the number of operations is n
  - hence total number of terms is $n\, 2^{n-1}$
- For small n this is manageable
  - but it is clearly important to avoid functions of too many variables!

Conclusions

- If we are to perform computation tasks appropriate to the properties of the material substrate we are using, we need a fairly general-purpose paradigm to define that computation
- The purpose of this talk is to describe such a paradigm: **belief propagation**
  - which matches well to the properties of the PN junction in silicon
- Belief propagation (*a.k.a.* message passing, the sum-product algorithm) is a method for general statistical inference
  - can find the marginal probabilities of a set of linked variables given some probabilities and a set of simple functions describing their relationship
  - allows a distributed solution
- Have described a simple example
  - plus applications in error correction decoding

Conclusions (cont)

- Potential applications in various recognition problems, in deducing states of an unknown system, inference about hypotheses
- Fundamentally a distributed processing technique, and fundamentally about coping with uncertainty
  - may be a model for operation of some biological processes, or other complex systems
- If probabilities are represented logarithmically:
  - easier to handle the range of probabilities of interest
  - products become sums
  - however sums become ‘log-sum-exp’
- However the I-V characteristic of the PN junction in silicon provides a very simple way to implement the exponentiation
  - other analogue functions (sum, log. conversion) can also be implemented simply in a similar way

Bibliography

- Kschischang, F.R, Frey, B.J. and Loeliger, H.-A., "Factor graphs and the sum-product algorithm," *Information Theory, IEEE Transactions on* , vol.47, no.2, pp.498-519, Feb 2001
- McEliece, R.J.; MacKay, D.J.C.; Jung-Fu Cheng, "Turbo decoding as an instance of Pearl's "belief propagation" algorithm," *Selected Areas in Communications, IEEE Journal on* , vol.16, no.2, pp.140-152, Feb 1998
- J. Pearl *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann, 1988.
- David J. C. MacKay "Information theory, inference and learning algorithms" Cambridge University Press, 2003