Bayesuvius,
a small visual dictionary of Bayesian Networks

Robert R. Tucci
www.ar-tiste.xyz

September 1, 2020

Figure 1: View of Mount Vesuvius from Pompeii

Figure 2: Mount Vesuvius and Bay of Naples

Contents

0.1 Foreword

Welcome to Bayesuvius! a proto-book uploaded to github.

A different Bayesian network is discussed in each chapter. Each chapter title is the name of a B net. Chapter titles are in alphabetical order.

This is a volcano in its early stages. First version uploaded to a github repo called Bayesuvius on June 24, 2020. First version only covers 2 B nets (Linear Regression and GAN). I will add more chapters periodically. Remember, this is a moonlighting effort so I can't do it all at once.

For any questions about notation, please go to Notational Conventions section.

Requests and advice are welcomed.

Thanks for reading this.
Robert R. Tucci
www.ar-tiste.xyz

0.2 Notational Conventions and Preliminaries

Some abbreviations frequently used throughout this book.

- bnet= B net= Bayesian Network
- TPM= Transition Probability Matrix
- $ch(\underline{a})$ = children of node $\underline{a}$.
- $pa(\underline{a})$ = parents of node $\underline{a}$.
- $nb(\underline{a}) = pa(\underline{a}) \cup ch(\underline{a})$ = neighbors of node $\underline{a}$.
- i.i.d.= independent identically distributed.

Define $\mathbb{Z}, \mathbb{R}, \mathbb{C}$ to be the integers, real numbers and complex numbers, respectively.

For $a < b$, define $\mathbb{Z}_I$ to be the integers in the interval I, where $I = [a,b], [a,b), (a,b], (a,b)$ (i.e, I can be closed or open on either side).

$A_{>0} = \{k \in A : k > 0\}$ for $A = \mathbb{Z}, \mathbb{R}$.

Random Variables will be indicated by underlined letters and their values by non-underlined letters. Each node of a bnet will be labelled by a random variable. Thus, $\underline{x} = x$ means that node $\underline{x}$ is in state x.

$P_{\underline{x}}(x) = P(\underline{x} = x) = P(x)$ is the probability that random variable $\underline{x}$ equals $x \in S_{\underline{x}}$. $S_{\underline{x}}$ is the set of states (i.e., values) that $\underline{x}$ can assume and $n_{\underline{x}} = |S_{\underline{x}}|$ is the size (aka cardinality) of that set. Hence,

$$\sum_{x \in S_{\underline{x}}} P_{\underline{x}}(x) = 1 \tag{1}$$

$$P_{\underline{x},\underline{y}}(x,y) = P(\underline{x} = x, \underline{y} = y) = P(x,y) \tag{2}$$

$$P_{\underline{x}|\underline{y}}(x|y) = P(\underline{x} = x|\underline{y} = y) = P(x|y) = \frac{P(x,y)}{P(y)} \tag{3}$$

Kronecker delta function: For x, y in discrete set S,

$$\delta(x,y) = \begin{cases} 1 \text{ if } x = y \\ 0 \text{ if } x \neq y \end{cases} \tag{4}$$

Dirac delta function: For $x, y \in \mathbb{R}$,

$$\int_{-\infty}^{+\infty} dx\ \delta(x-y) f(x) = f(y) \tag{5}$$

Transition probability matrix of a node of a bnet can be either a discrete or a continuous probability distribution. To go from continuous to discrete, one replaces integrals over states of node by sums

over new states, and Dirac delta functions by Kronecker delta functions. More precisely, consider a function $f : S \to \mathbb{R}$. Let $S_{\underline{x}} \subset S$ and $S \to S_{\underline{x}}$ upon discretization (binning). Then

$$\int_S dx\ P_{\underline{x}}(x) f(x) \to \frac{1}{n_{\underline{x}}} \sum_{x \in S_{\underline{x}}} f(x)\ . \tag{6}$$

Both sides of last equation are 1 when $f(x) = 1$. Furthermore, if $y \in S_{\underline{x}}$, then

$$\int_S dx\ \delta(x-y) f(x) = f(y) \to \sum_{x \in S_{\underline{x}}} \delta(x,y) f(x) = f(y)\ . \tag{7}$$

Indicator function (aka Truth function):

$$\mathbb{1}(\mathcal{S}) = \begin{cases} 1 \text{ if } \mathcal{S} \text{ is true} \\ 0 \text{ if } \mathcal{S} \text{ is false} \end{cases} \tag{8}$$

For example, $\delta(x,y) = \mathbb{1}(x = y)$.

$$\vec{x} = (x[0], x[1], x[2] \ldots, x[nsam(\vec{x}) - 1]) = x[:] \tag{9}$$

$nsam(\vec{x})$ is the number of samples of $\vec{x}$. $\underline{x}[i] \in S_{\underline{x}}$ are i.i.d. (independent identically distributed) samples with

$$x[i] \sim P_{\underline{x}} \ \text{ (i.e. } P_{\underline{x[i]}} = P_{\underline{x}}) \tag{10}$$

$$P(\underline{x} = x) = \frac{1}{nsam(\vec{x})} \sum_i \mathbb{1}(x[i] = x) \tag{11}$$

If we use two sampled variables, say $\vec{x}$ and $\vec{y}$, in a given bnet, their number of samples $nsam(\vec{x})$ and $nsam(\vec{y})$ need not be equal.

$$P(\vec{x}) = \prod_i P(x[i]) \tag{12}$$

$$\sum_{\vec{x}} = \prod_i \sum_{x[i]} \tag{13}$$

$$\partial_{\vec{x}} = [\partial_{x[0]}, \partial_{x[1]}, \partial_{x[2]}, \ldots, \partial_{x[nsam(\vec{x})-1]}] \tag{14}$$

$$\begin{aligned} P(\vec{x}) &\approx [\prod_x P(x)^{P(x)}]^{nsam(\vec{x})} && (15) \\ &= e^{nsam(\vec{x}) \sum_x P(x) \ln P(x)} && (16) \\ &= e^{-nsam(\vec{x}) H(P_{\underline{x}})} && (17) \end{aligned}$$

$$f^{[1,\partial_x,\partial_y]}(x,y) = [f, \partial_x f, \partial_y f] \tag{18}$$

$$f^+ = f^{[1,\partial_x,\partial_y]} \tag{19}$$

For probabilty distributions $p(x), q(x)$ of $x \in S_{\underline{x}}$

- Entropy:

$$H(p) = -\sum_x p(x) \ln p(x) \geq 0 \tag{20}$$

- Kullback-Liebler divergence:

$$D_{KL}(p \parallel q) = \sum_x p(x) \ln \frac{p(x)}{q(x)} \geq 0 \tag{21}$$

- Cross entropy:

$$\begin{aligned} CE(p \to q) &= -\sum_x p(x) \ln q(x) & (22) \\ &= H(p) + D_{KL}(p \parallel q) & (23) \end{aligned}$$

Normal Distribution: $x, \mu, \sigma \in \mathbb{R}$, $\sigma > 0$

$$\mathcal{N}(\mu, \sigma^2)(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{24}$$

Uniform Distribution: $a < b$, $x \in [a, b]$

$$\mathcal{U}(a, b)(x) = \frac{1}{b-a} \tag{25}$$

Expected Value and Variance

Given a random variable $\underline{x}$ with states $S_{\underline{x}}$ and a function $f : S_{\underline{x}} \to \mathbb{R}$, define

$$E_{\underline{x}}[f(\underline{x})] = E_{x \sim P(x)}[f(x)] = \sum_x P(x) f(x) \tag{26}$$

$$\begin{aligned} Var_{\underline{x}}[f(\underline{x})] &= E_{\underline{x}}\left[(f(\underline{x}) - E_{\underline{x}}[f(\underline{x})])^2\right] & (27) \\ &= E_{\underline{x}}[f(\underline{x})^2] - (E_{\underline{x}}[f(\underline{x})])^2 & (28) \end{aligned}$$

Conditional Expected Value

Given a random variable $\underline{x}$ with states $S_{\underline{x}}$, a random variable $\underline{y}$ with states $S_{\underline{y}}$, and a function $f : S_{\underline{x}} \times S_{\underline{y}} \to \mathbb{R}$, define

$$E_{\underline{x}|\underline{y}}[f(\underline{x}, \underline{y})] = \sum_x P(x|\underline{y}) f(x, \underline{y}) \ , \tag{29}$$

$$E_{\underline{x}|\underline{y}=y}[f(\underline{x}, y)] = E_{\underline{x}|y}[f(\underline{x}, y)] = \sum_x P(x|y) f(x, y) \ . \tag{30}$$

Note that

$$E_{\underline{y}}[E_{\underline{x}|\underline{y}}[f(\underline{x},\underline{y})]] = \sum_{x,y} P(x|y)P(y)f(x,y) \tag{31}$$

$$= \sum_{x,y} P(x,y)f(x,y) \tag{32}$$

$$= E_{\underline{x},\underline{y}}[f(\underline{x},\underline{y})] \; . \tag{33}$$

Sigmoid function: For $x \in \mathbb{R}$,

$$\mathrm{sig}(x) = \frac{1}{1+e^{-x}} \tag{34}$$

$\mathcal{N}(!a)$ will denote a normalization constant that does not depend on a. For example, $P(x) = \mathcal{N}(!x)e^{-x}$ where $\int_0^\infty dx\; P(x) = 1$.

A **one hot** vector of zeros and ones is a vector with all entries zero with the exception of a single entry which is one. A **one cold** vector has all entries equal to one with the exception of a single entry which is zero. For example, if $x^n = (x_0, x_1, \ldots, x_{n-1})$ and $x_i = \delta(i,0)$ then x^n is one hot.

Short Summary of Boolean Algebra.

See Ref.[1] for more info about this topic.

Suppose $x, y, z \in \{0,1\}$. Define

$$x \text{ or } y = x \vee y = x + y - xy \; , \tag{35}$$

$$x \text{ and } y = x \wedge y = xy \; , \tag{36}$$

and

$$\text{not } x = \overline{x} = 1 - x \; , \tag{37}$$

where we are using normal addition and multiplication on the right hand sides.[1]

Actually, since $x \wedge y = xy$, we can omit writing the symbol $\wedge$. The symbol $\wedge$ is useful to exhibit the symmetry of the identities, and to remark about the analogous identities for sets, where $\wedge$ becomes intersection $\cap$ and $\vee$ becomes union $\cup$. However, for practical calculations, $\wedge$ is an unnecessary nuisance.

Since $x \in \{0,1\}$,

$$P(\overline{x}) = 1 - P(x) \; . \tag{38}$$

Clearly, from analyzing the simple event space $(x,y) \in \{0,1\}^2$,

$$P(x \vee y) = P(x) + P(y) - P(x \wedge y) \; . \tag{39}$$

[1]Note the difference between $\vee$ and modulus 2 addition $\oplus$. For $\oplus$ (aka XOR): $x \oplus y = x + y - 2xy$.

Associativity	$x \vee (y \vee z) = (x \vee y) \vee z$ $x \wedge (y \wedge z) = (x \wedge y) \wedge z$
Commutativity	$x \vee y = y \vee x$ $x \wedge y = y \wedge x$
Distributivity	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
Identity	$x \vee 0 = x$ $x \wedge 1 = x$
Annihilator	$x \wedge 0 = 0$ $x \vee 1 = 1$
Idempotence	$x \vee x = x$ $x \wedge x = x$
Absorption	$x \wedge (x \vee y) = x$ $x \vee (x \wedge y) = x$
Complementation	$x \wedge \overline{x} = 0$ $x \vee \overline{x} = 1$
Double negation	$\overline{(\overline{x})} = x$
De Morgan Laws	$\overline{x} \wedge \overline{y} = \overline{(x \vee y)}$ $\overline{x} \vee \overline{y} = \overline{(x \wedge y)}$

Table 1: Boolean Algebra Identities

Chapter 1

Back Propagation (Automatic Differentiation)

General Theory

Jacobians

Suppose $f : \mathbb{R}^{nx} \to \mathbb{R}^{nf}$ and

$$y = f(x) \ . \tag{1.1}$$

Then the Jacobian $\frac{\partial y}{\partial x}$ is defined as the matrix with entries[1]

$$\left[\frac{\partial y}{\partial x}\right]_{i,j} = \frac{\partial y_i}{\partial x_j} \ . \tag{1.2}$$

Jacobian of function composition. Supoose $f : \mathbb{R}^{nx} \to \mathbb{R}^{nf}$, $g : \mathbb{R}^{nf} \to \mathbb{R}^{ng}$. If

$$y = g \circ f(x) \ , \tag{1.3}$$

then

$$\frac{\partial y}{\partial x} = \frac{\partial g}{\partial f}\frac{\partial f}{\partial x} \ . \tag{1.4}$$

Right hand side of last equation is a product of two matrices so order of matrices is important.

Let

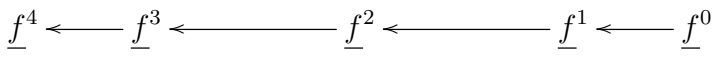
$$y = f^4 \circ f^3 \circ f^2 \circ f^1(x) \ . \tag{1.5}$$

This function composition chain can be represented by the bnet Fig.1.1(a) with TPMs

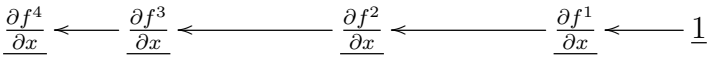
$$P(f^\mu | f^{\mu-1}) = \mathbb{1}(f^\mu = f^\mu(f^{\mu-1})) \tag{1.6}$$

for $\mu = 1, 2, 3, 4$.

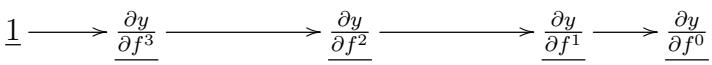
[1] Mnemonic for remembering order of indices: i in numerator/j in denominator becomes index i/j of Jacobian matrix.



(a) Composition



(b) Forward-p



(c) Back-p

Figure 1.1: bnets for function composition, forward propagation and back propagation for $nf = 5$ nodes.

Note that

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial f^3}\frac{\partial f^3}{\partial f^2}\left[\frac{\partial f^2}{\partial f^1}\frac{\partial f^1}{\partial x}\right] \tag{1.7}$$

$$= \frac{\partial y}{\partial f^3}\left[\frac{\partial f^3}{\partial f^2}\frac{\partial f^2}{\partial x}\right] \tag{1.8}$$

$$= \left[\frac{\partial y}{\partial f^3}\frac{\partial f^3}{\partial x}\right] \tag{1.9}$$

$$= \frac{\partial y}{\partial x}\,. \tag{1.10}$$

This forward propagation can be represented by the bnet Fig.1.1(b) with node TPMs

$$P(\frac{\partial f^{\mu+1}}{\partial x} \mid \frac{\partial f^{\mu}}{\partial x}) = \mathbb{1}(\frac{\partial f^{\mu+1}}{\partial x} = \frac{\partial f^{\mu+1}}{\partial f^{\mu}}\frac{\partial f^{\mu}}{\partial x}) \tag{1.11}$$

for $\mu = 1, 2, 3$.

Note that

$$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial f^3}\frac{\partial f^3}{\partial f^2}\right]\frac{\partial f^2}{\partial f^1}\frac{\partial f^1}{\partial x} \tag{1.12}$$

$$= \left[\frac{\partial y}{\partial f^2}\frac{\partial f^2}{\partial f^1}\right]\frac{\partial f^1}{\partial x} \tag{1.13}$$

$$= \left[\frac{\partial y}{\partial f^1}\frac{\partial f^1}{\partial x}\right] \tag{1.14}$$

$$= \frac{\partial y}{\partial x}\,. \tag{1.15}$$

This back propagation can be represented by the bnet Fig.1.1(c) with node TPMs

$$P(\frac{\partial y}{\partial f^\mu} \mid \frac{\partial y}{\partial f^{\mu+1}}) = \mathbb{1}(\frac{\partial y}{\partial f^\mu} = \frac{\partial y}{\partial f^{\mu+1}}\frac{\partial f^{\mu+1}}{\partial f^\mu}) \tag{1.16}$$

for $\mu = 2, 1, 0$.

$\frac{\partial f^{\mu+1}}{\partial f^\mu}$ is a Jacobian matrix so the order of multiplication matters. In forward prop, it pre-multiplies, and in back prop it post-multiplies.

Application to Neural Networks

Absorbing b_i^λ into $w_{i|j}$.

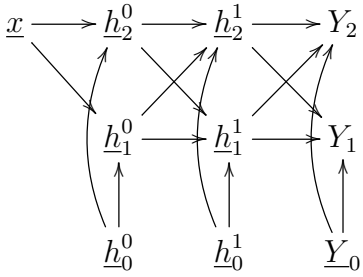


Figure 1.2: Nodes $\underline{h}_0^0, \underline{h}_0^1, \underline{Y}_0$ are all set to 1. They allow us to absorb b_i^λ into the first column of $w_{i|j}^\lambda$.

Below are, printed in blue, the TPMs for the nodes of a NN bnet, as given in Chapter 24. For all hidden layers $\lambda = 0, 1, \ldots, \Lambda - 2$,

$$P(h_i^\lambda \mid h_\cdot^{\lambda-1}) = \delta\left(h_i^\lambda, \mathcal{A}_i^\lambda(\sum_j w_{i|j}^\lambda h_j^{\lambda-1} + b_i^\lambda)\right) \tag{1.17}$$

for $i = 0, 1, \ldots, nh(\lambda) - 1$. For the output visible layer $\lambda = \Lambda - 1$:

$$P(Y_i \mid h_\cdot^{\Lambda-2}) = \delta\left(Y_i, \mathcal{A}_i^{\Lambda-1}(\sum_j w_{i|j}^{\Lambda-1} h_j^{\Lambda-2} + b_i^{\Lambda-1})\right) \tag{1.18}$$

for $i = 0, 1, \ldots, ny-1$.

For each λ, replace the matrix $w^{\lambda}_{\cdot|\cdot}$ by the augmented matrix $[b^{\lambda}_{\cdot}, w^{\lambda}_{\cdot|\cdot}]$ so that the new $w^{\lambda}_{\cdot|\cdot}$ satisfies

$$w^{\lambda}_{i|0} = b^{\lambda}_{i} \tag{1.19}$$

Let the nodes $\underline{h}^{\lambda}_0$ for all λ and $\underline{Y}_0$ be root nodes (so no arrows pointing into them). For each λ, draw arrows from $\underline{h}^{\lambda}_0$ to all other nodes in that same layer. Draw arrows from $\underline{Y}_0$ to all other nodes in that same layer.

After performing the above steps, the TPMs, printed in blue, for the nodes of the NN bnet are as follows:

For all hidden layers $\lambda = 0, 1, \ldots, \Lambda - 2$,

$$P(h^{\lambda}_0) = \delta(h^{\lambda}_0, 1) \ , \tag{1.20}$$

and

$$P(h^{\lambda}_i \mid h^{\lambda-1}_{\cdot}, h^{\lambda}_0 = 1) = \delta\left(h^{\lambda}_i, \mathcal{A}^{\lambda}_i(\sum_j w^{\lambda}_{i|j} h^{\lambda-1}_j)\right) \tag{1.21}$$

for $i = 1, \ldots, nh(\lambda) - 1$. For the output visible layer $\lambda = \Lambda - 1$:

$$P(Y_0) = \delta(Y_0, 1) \ , \tag{1.22}$$

and

$$P(Y_i \mid h^{\Lambda-2}_{\cdot}, Y_0 = 1) = \delta\left(Y_i, \mathcal{A}^{\Lambda-1}_i(\sum_j w^{\Lambda-1}_{i|j} h^{\Lambda-2}_j)\right) \tag{1.23}$$

for $i = 1, 2, \ldots, ny - 1$.

From here on, we will rename y above by $Y = \hat{y}$ and consider samples $y[i]$ for $i = 0, 1, \ldots, nsam-1$. The Error (aka loss or cost function) is

$$\mathcal{E} = \frac{1}{nsam} \sum_{s=0}^{nsam-1} \sum_{i=0}^{ny-1} |Y_i - y_i[s]|^2 \tag{1.24}$$

To perform simple gradient descent, one uses:

$$(w^{\lambda}_{i|j})' = w^{\lambda}_{i|j} - \eta \frac{\partial \mathcal{E}}{\partial w^{\lambda}_{i|j}} \ . \tag{1.25}$$

One has

$$\frac{\partial \mathcal{E}}{\partial w^{\lambda}_{i|j}} = \frac{1}{nsam} \sum_{s=0}^{nsam-1} \sum_{i=0}^{ny-1} 2(Y_i - y_i[s]) \frac{\partial Y}{\partial w^{\lambda}_{i|j}} \ . \tag{1.26}$$

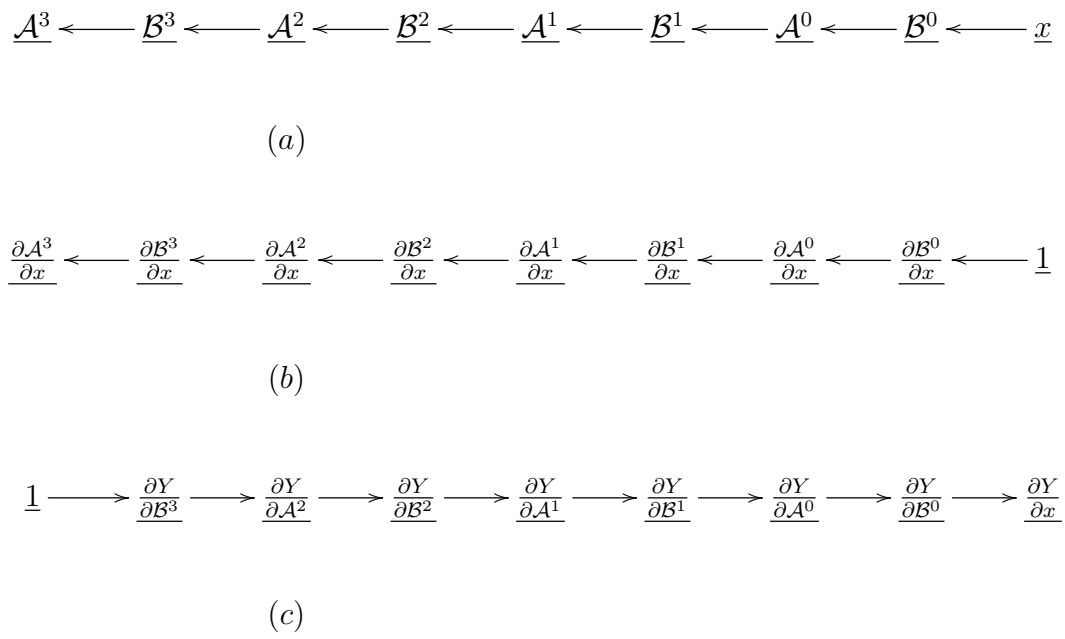


Figure 1.3: bnets for (a) function composition, (b) forward propagation and (c) back propagation for a neural net with 4 layers (3 hidden and output visible).

Define $\mathcal{B}_i^\lambda$ thus

$$\mathcal{B}_i^\lambda(h^{\lambda-1}) = \sum_j w_{i|j}^\lambda h_j^{\lambda-1} \ . \tag{1.27}$$

Then

$$\frac{\partial Y}{\partial w_{i|j}^\lambda} = \frac{\partial Y}{\partial \mathcal{B}_i^\lambda}\frac{\partial \mathcal{B}_i^\lambda}{\partial w_{i|j}^\lambda} \tag{1.28}$$

$$= \frac{\partial Y}{\partial \mathcal{B}_i^\lambda} h_j^{\lambda-1} \tag{1.29}$$

$$\frac{\partial \mathcal{E}}{\partial w_{i|j}^\lambda} = \frac{\partial \mathcal{E}}{\partial \mathcal{B}_j^\lambda}\frac{\partial \mathcal{B}_j^\lambda}{\partial w_{i|j}^\lambda} \tag{1.30}$$

$$= \frac{\partial \mathcal{E}}{\partial \mathcal{B}_j^\lambda} h_j^{\lambda-1} \ . \tag{1.31}$$

This suggest that we can calculate the derivatives of the error $\mathcal{E}$ with respect to the weights $w_{i|j}^\lambda$ in

two stages, using an intermediate quantity δ_j^λ:

$$\begin{cases} \delta_j^\lambda = \frac{\partial \mathcal{E}}{\partial \mathcal{B}_j^\lambda} \\ \frac{\partial \mathcal{E}}{\partial w_{i|j}^\lambda} = \delta_j^\lambda h_j^{\lambda-1} \end{cases} \tag{1.32}$$

To apply what we learned in the earlier General Theory section of this chapter, consider a NN with 4 layers (3 hidden, and the output visible one). Define the functions f_i as follows:

$$f_i^0 = x_i \tag{1.33}$$

$$\text{Layer 0:} \quad f_i^1 = \mathcal{B}_i^0(x_i), \quad f_i^2 = \mathcal{A}_i^0(\mathcal{B}_i^0) \tag{1.34}$$
$$\text{Layer 1:} \quad f_i^3 = \mathcal{B}_i^1(\mathcal{A}_i^0), \quad f_i^4 = \mathcal{A}_i^1(\mathcal{B}_i^1) \tag{1.35}$$
$$\text{Layer 2:} \quad f_i^5 = \mathcal{B}_i^2(\mathcal{A}_i^1), \quad f_i^6 = \mathcal{A}_i^2(\mathcal{B}_i^2) \tag{1.36}$$
$$\text{Layer 3:} \quad f_i^7 = \mathcal{B}_i^3(\mathcal{A}_i^2), \quad f_i^8 = \mathcal{A}_i^3(\mathcal{B}_i^3) \tag{1.37}$$

See Fig.1.3. The TPMs, printed in blue, for the nodes of the bnet (c) for back propagation, are:

$$P(\frac{\partial Y}{\partial \mathcal{B}^\lambda} \mid \frac{\partial Y}{\partial \mathcal{B}^{\lambda+1}}) = \mathbb{1}(\frac{\partial Y}{\partial \mathcal{B}^\lambda} = \frac{\partial Y}{\partial \mathcal{B}^{\lambda+1}} \frac{\partial \mathcal{B}^{\lambda+1}}{\partial \mathcal{A}^\lambda} \frac{\partial \mathcal{A}^\lambda}{\partial \mathcal{B}^\lambda}) \ . \tag{1.38}$$

One has

$$\frac{\partial \mathcal{A}_i^\lambda}{\partial \mathcal{B}_j^\lambda} = D\mathcal{A}_i{}^\lambda(\mathcal{B}_i^\lambda)\delta(i,j) \tag{1.39}$$

where $D\mathcal{A}_i^\lambda(z)$ is the derivative of $\mathcal{A}_i^\lambda(z)$.

From Eq.(1.27)

$$\mathcal{B}_i^{\lambda+1}(\mathcal{A}^\lambda) = \sum_j w_{i|j}^{\lambda+1} \mathcal{A}_j^\lambda \tag{1.40}$$

so

$$\frac{\partial \mathcal{B}_i^{\lambda+1}}{\partial \mathcal{A}_j^\lambda} = w_{i|j}^{\lambda+1} \ . \tag{1.41}$$

Therefore, Eq.(1.38) implies

$$P(\frac{\partial Y}{\partial \mathcal{B}_j^\lambda} \mid \frac{\partial Y}{\partial \mathcal{B}_j^{\lambda+1}}) = \mathbb{1}(\frac{\partial Y}{\partial \mathcal{B}_j^\lambda} = \sum_i \frac{\partial Y}{\partial \mathcal{B}_i^{\lambda+1}} D\mathcal{A}_j^\lambda(\mathcal{B}_j^\lambda) w_{i|j}^{\lambda+1}) \ , \tag{1.42}$$

$$P(\frac{\partial \mathcal{E}}{\partial \mathcal{B}_j^\lambda} \mid \frac{\partial \mathcal{E}}{\partial \mathcal{B}_j^{\lambda+1}}) = \mathbb{1}(\frac{\partial \mathcal{E}}{\partial \mathcal{B}_j^\lambda} = \sum_i \frac{\partial \mathcal{E}}{\partial \mathcal{B}_i^{\lambda+1}} D\mathcal{A}_j^\lambda(\mathcal{B}_j^\lambda) w_{i|j}^{\lambda+1}) \ , \tag{1.43}$$

$$\boxed{P(\delta_j^\lambda \mid \delta_j^{\lambda+1}) = \mathbb{1}(\delta_j^\lambda = \sum_i \delta_i^{\lambda+1} D\mathcal{A}_j^\lambda(\mathcal{B}_j^\lambda)) w_{i|j}^{\lambda+1})} \,. \tag{1.44}$$

First delta of iteration, belonging to output layer $\lambda = \Lambda - 1$:

$$\begin{aligned} \delta_j^{\Lambda-1} &= \frac{\partial \mathcal{E}}{\partial \mathcal{B}_j^{\Lambda-1}} && (1.45)\\ &= \frac{1}{nsam} \sum_{s=0}^{nsam-1} \sum_{i=0}^{ny-1} 2(Y_i - y_i[s]) D\mathcal{A}_i^{\Lambda-1}(\mathcal{B}_i^{\Lambda-1}) \delta(i,j) && (1.46)\\ &= \frac{1}{nsam} \sum_{s=0}^{nsam-1} 2(Y_j - y_j[s]) D\mathcal{A}_j^{\Lambda-1}(\mathcal{B}_j^{\Lambda-1}) && (1.47) \end{aligned}$$

Cute expression for derivative of sigmoid function:

$$D\mathrm{sig}(x) = \mathrm{sig}(x)(1 - \mathrm{sig}(x)) \tag{1.48}$$

Generalization to bnets (instead of Markov chains induced by layered structure of NNs):

$$P(\delta_{\underline{x}} \mid (\delta_{\underline{a}})_{\underline{a} \in ch(\underline{x})}) = \mathbb{1}(\delta_{\underline{x}} = \sum_{\underline{a} \in ch(\underline{x})} \delta_{\underline{a}} D\mathcal{A}_{\underline{x}}(\mathcal{B}_{\underline{x}})) w_{\underline{a}|\underline{x}}) \tag{1.49}$$

Reverse arrows of original bnet and define the TPM of nodes of "time reversed" bnet by

$$\boxed{P(\delta_{\underline{x}} \mid (\delta_{\underline{a}})_{\underline{a} \in pa(\underline{x})}) = \mathbb{1}(\delta_{\underline{x}} = \sum_{\underline{a} \in pa(\underline{x})} \delta_{\underline{a}} D\mathcal{A}_{\underline{x}}(\mathcal{B}_{\underline{x}})) w_{\underline{x}|\underline{a}}^T)} \tag{1.50}$$

Chapter 2

Basic Curve Fitting Using Gradient Descent

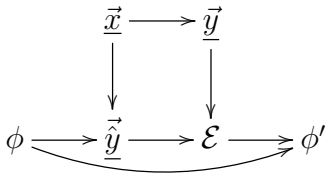


Figure 2.1: Basic curve fitting bnet.

Samples $(x[i], y[i]) \in S_{\underline{x}} \times S_{\underline{y}}$ are given. $nsam(\vec{x}) = nsam(\vec{y})$.
Estimator function $\hat{y}(x;\phi)$ for $x \in S_{\underline{x}}$ and $\phi \in \mathbb{R}$ is given.
Let

$$P_{\underline{x},\underline{y}}(x,y) = \frac{1}{nsam(\vec{x})} \sum_i \mathbb{1}(x = x[i], y = y[i]) \ . \tag{2.1}$$

Let

$$\mathcal{E}(\vec{x},\vec{y},\phi) = \frac{1}{nsam(\vec{y})} \sum_i |y[i] - \hat{y}(x[i];\phi)|^2 \tag{2.2}$$

$\mathcal{E}$ is called the mean square error.

Best fit is parameters ϕ^* such that

$$\phi^* = \underset{\phi}{\operatorname{argmin}}\, \mathcal{E}(\vec{x},\vec{y},\phi) \ . \tag{2.3}$$

The node TPMs for the basic curve fitting bnet Fig.2.1 are printed below in blue.

$$P(\phi) = \text{given} \ . \tag{2.4}$$

The first time it is used, ϕ is arbitrary. After the first time, it is determined by previous stage.

$$P(\vec{x}) = \prod_i P_{\underline{x}}(x[i]) \tag{2.5}$$

$$P(\vec{y}|\vec{x}) = \prod_i P_{\underline{y}|\underline{x}}(y[i] \mid x[i]) \tag{2.6}$$

$$P(\hat{y}[i]|\phi, \vec{x}) = \delta(\hat{y}[i], \hat{y}(x[i]; \phi)) \tag{2.7}$$

$$P(\mathcal{E}|\vec{\hat{y}}, \vec{y}) = \delta(\mathcal{E}, \frac{1}{nsam(\vec{x})} \sum_i |y[i] - \hat{y}[i]|^2) \ . \tag{2.8}$$

$$P(\phi'|\phi, \mathcal{E}) = \delta(\phi', \phi - \eta \partial_\phi \mathcal{E}) \tag{2.9}$$

$\eta > 0$ is the descent rate. If $\Delta\phi = \phi' - \phi = -\eta \frac{\partial \mathcal{E}}{\partial \phi}$, then $\Delta\mathcal{E} = \frac{-1}{\eta}(\Delta\phi)^2 < 0$ so this will minimize the error $\mathcal{E}$. This is called "gradient descent".

Chapter 3

Bell and Clauser-Horne Inequalities in Quantum Mechanics

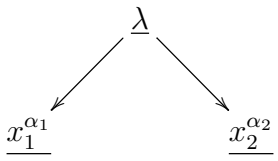
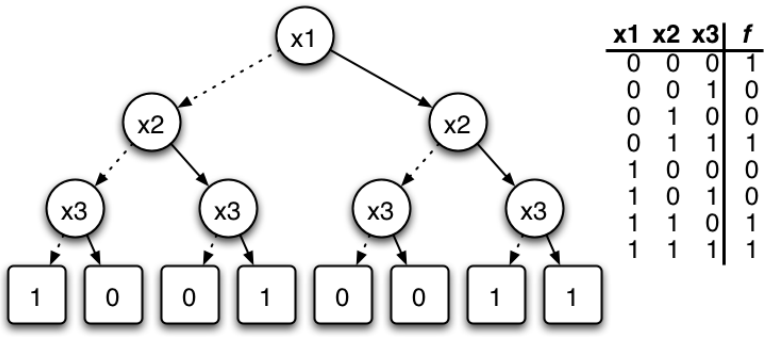


Figure 3.1: bnet used to discuss Bell and Clauser-Horne inequalities in Quantum Mechanics.

I wrote an article about this in 2008 for my blog "Quantum Bayesian Networks". See Ref.[2].

Chapter 4

Binary Decision Diagrams



x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Figure 4.1: Binary decision tree and truth table for the function $f(x_1, x_2, x_3) = \bar{x}_1(x_2 + \bar{x}_3) + x_1x_2$

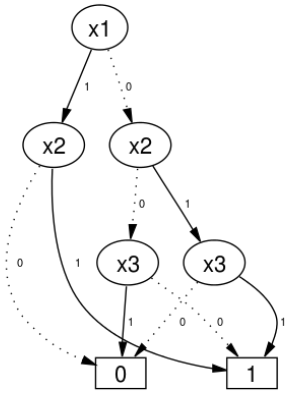


Figure 4.2: BDD for the function f of Fig.4.1.

This chapter is based on Wikipedia article Ref.[3].

Binary Decision Diagrams (BDDs) can be understood as a special case of Decision Trees (dtrees). We will assume that the reader has read Chapter 5 on dtrees before reading this chapter.

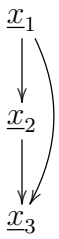
Both Figs.4.1 and 4.2 were taken from the aforementioned Wikipedia article. They give a simple example of a function $f : \{0,1\}^3 \to \{0,1\}$ represented in Fig.4.1 as a **binary decision tree** and in Fig.4.2 as a **binary decision diagram (BDD)**. The goal of this chapter is to find for each of those figures a bnet with the same graph structure.

We begin by noting that the function $f : \{0,1\}^3 \to \{0,1\}$ is a special case of a probability distribution $P : \{0,1\}^3 \to [0,1]$. In fact, if we restrict P to be deterministic, then $P_{det} : \{0,1\}^3 \to \{0,1\}$ has the same domain and range as f. Henceforth, we will refer to $f(x_1, x_2, x_3)$ as $P(x_1, x_2, x_3)$, keeping in mind that we are restricting our attention to deterministic probability distributions.

If we apply the chain rule for conditional probabilities to $P(x_1, x_2, x_3)$, we get

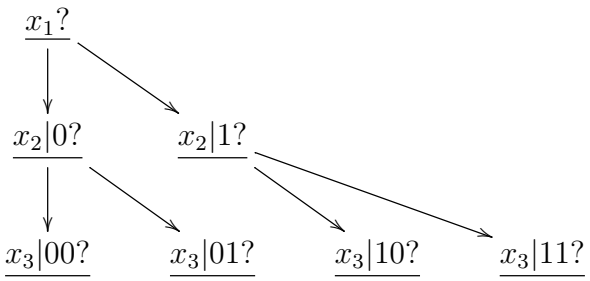
$$P(x_1, x_2, x_3) = P(x_3|x_1, x_2)P(x_2|x_1)P(x_1) \ , \tag{4.1}$$

which can be represented by the bnet:



(4.2)

But in Chapter 5, we learned how to represent the bnet of Eq.(4.2) as the bnet tree Eq.(4.3). In that tree, the nodes pose questions with 3 possible answers $0, 1, null$. In Eq.(4.3), $x_2|a?$ stands for "what is x_2 if $x_1 = a$?" and $x_3|a, b?$ stands for "what is x_3 if $x_1 = a, x_2 = b$?".



(4.3)

The node TPMs, printed in blue, for the bnet of Eq.(4.3) are as follows. If $x_1, x_2, x_3 \in \{0, 1, null\}$ and $a, b \in \{0, 1\}$, then

$$P(\underline{x_1?} = x_1) = \begin{cases} P_{\underline{x_1}}(x_1) & \text{if } x_1 \in \{0,1\} \\ 0 & \text{if } x_1 = null \end{cases} \tag{4.4}$$

$$P(\underline{x_2|a?} = x_2 \mid \underline{x_1?} = x_1) = \begin{cases} P_{\underline{x_2|x_1}}(x_2|a) & \text{if } x_1 = a \\ \mathbb{1}(x_2 = null) & \text{otherwise} \end{cases} \tag{4.5}$$

$$P(\underline{x_3|a,b?} = x_3 \mid \underline{x_2|b?} = x_2) = \begin{cases} P_{\underline{x_3}|\underline{x_1},\underline{x_2}}(x_3|a,b) & \text{if } (x_1,x_2) = (a,b) \\ \mathbb{1}(x_3 = null) & \text{otherwise} \end{cases} \tag{4.6}$$

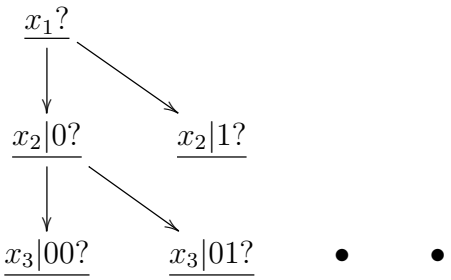
The bnet shown in Eq.(4.3) contains the same info and has the same graph structure as the binary decision tree Fig.4.1. As when we were converting dtrees to their image bnets, the info in the endpoint nodes of Fig.4.1 is implicit in the node TPMs of the image bnet Eq.(4.3). If one wants to make the endpoint node info more explicit in the image bnet, one can add it to the descriptors of the state names of the leaf nodes of the image bnet. For example, one can add descriptors "gives $f = 0$" or "gives $f = 1$" to the "0" or "1" states of those leaf nodes.

The BDD shown in Fig.4.2 emphasizes the fact that

$$P(x_1, x_2, x_3|x_1 = 1) = P(x_2|x_1 = 1) = x_2 \ . \tag{4.7}$$

The BDD of Fig.4.2 corresponds to the bnet of Eq.(4.8).

(4.8)

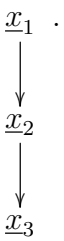


What happens if we consider an f for which $P(x_3|x_1, x_2) = P(x_3|x_2)$ so that one of the arcs of the fully connected bnet Eq.(4.2) is unnecessary? In that case,

$$P(x_1, x_2, x_3) = P(x_3|x_2)P(x_2|x_1)P(x_1) \ , \tag{4.9}$$

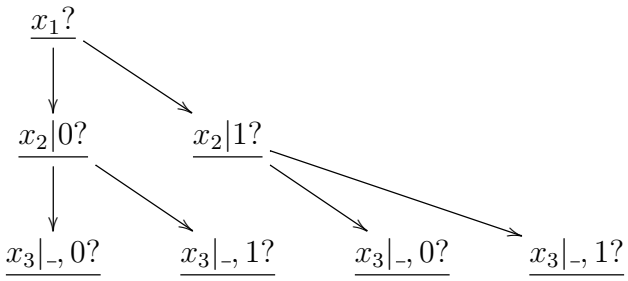
which can be represented by the Markov chain bnet:

(4.10)



Following the prescriptions of Chapter 5, we can represent the bnet of Eq.(4.10) as the bnet

tree Eq.(4.11). In that tree, the nodes pose questions with 3 possible answers $0, 1, null$.



(4.11)

Chapter 5

Decision Trees

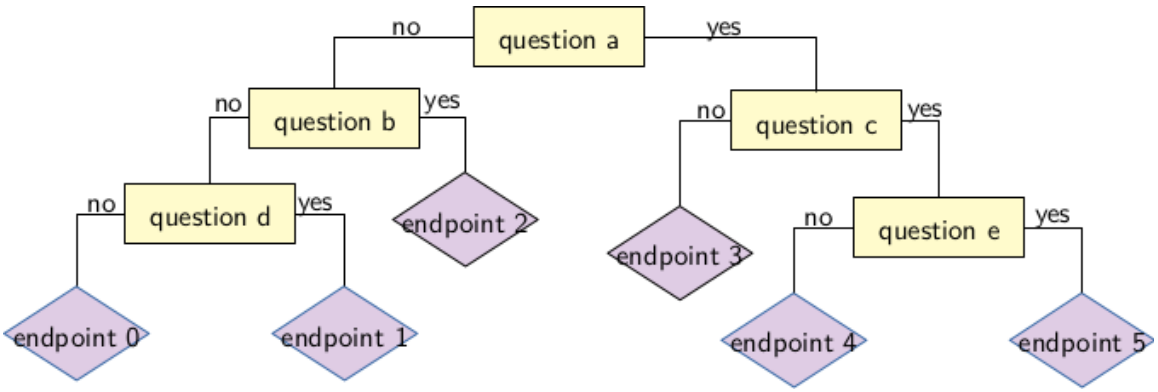


Figure 5.1: Typical decision tree.

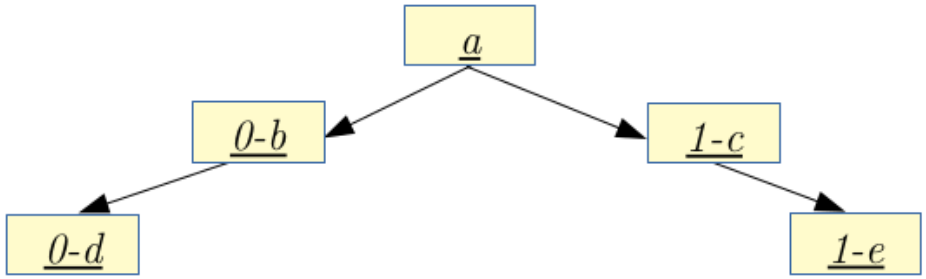


Figure 5.2: Bnet corresponding to decision tree Fig.5.1

Fig.5.1 shows a typical decision tree (dtree). The yellow rectangles pose questions. In general, the answers to those questions can be multiple choices with more than two choices, but in Fig.5.1 we have chosen the simplest case of only two choices, true or false. The purple diamonds represent endpoints, goals, final conclusions, single states of reality, etc.

A trivial observation that is often not made in dtree educational literature is that every dtree maps into a special bnet, let's call it its "image" bnet, in a very natural way. To get the image bnet, just follow the following simple steps:

1. **Keep the yellow question nodes but reinterpret them as bnet nodes. Reinterpret the connections among the dtree question nodes as arrows pointing down from the root node.**

   The image bnet nodes have 3 states, $0 = no$ and $1 = yes$ and $null$. Table 5.1 gives the node TPM $[P(x|a)]_{x\in\{0,1,null\},a\in\{0,1,null\}}$ where $p_1 \in [0,1]$ can be different for each node and is given in the info that specifies the dtree. In Table 5.1, $a_0 = 0$ if the dtree node being replaced has input "no" and $a_0 = 1$ if its input is "yes". $!a_0$ means not a_0 (i.e., $!a_0 = 1 - a_0$).

2. **This method of naming the image bnet nodes is not necessary but a good practice.** Give as name to each image bnet node an abridged version of the question that labels the dtree node it is replacing. Use as a suffix to the name of a bnet node either a 0 or a 1 depending whether the dtree node it is replacing has a 0 or a 1 as input. This suffix is not necessary because its info is already encoded into which column of the node TPM has zero probability for the $null$ state, but it's a redundancy which makes the bnet easier to read and understand.

3. **Erase the purple endpoint nodes and connectors to them.** The info in each endpoint node can be preserved by storing it as a descriptor (e.g., tool tip) for the output states of the leaf node that is the parent to the endpoint in the image bnet. The endpoint info can be added to the descriptor of the $no = 0$ state if the endpoint has 0 as input or to the descriptor of the $yes = 1$ state if the endpoint has 1 as input.

$P(x\|a)$	$a = a_0$	$a = !a_0$	$a = null$
$x = 0$	$1 - p_1$	0	0
$x = 1$	p_1	0	0
$x = null$	0	1	1

Table 5.1: Transition probability matrix of a node of a dtree image bnet.

Table 5.2 describes the node types commonly used in dtrees.

When drawing dtrees, some people put info like explanations and probabilities on the connectors between the nodes of the dtree. That info can all be preserved in the TPM and the descriptors of the node names and node state names of the image bnet nodes. Often, the educational literature states that dtrees are more explicit and carry more info than their image bnets, but if one follows the above prescriptions, both can carry the same info.

A deterministic node commonly used in dtrees is one that asks the question $x < \alpha$?. for some real number $\alpha \in (L, U)$ and some variable x (for example, $x =$ height of a person). For such an interval splitting node, the TPM would be as given in Table 5.3. If the interval $[L, U]$ is binned into a number $nbins$ of bins, then this TPM will have dimensions $(nbins + 1)\times($ the number of states of the parent node).

dtree node types (usual shape in parenthesis)	**their node TPM $P(x\|a)$ in image bnet**
chance node (oval)	$P(x\|a)$ arbitrary. random
decision node (square)	$P(x\|a) = \delta(x, f(a))$ where $f(\cdot)$ is a function of a. deterministic
endpoint node (diamond)	no $P(x\|a)$
fixed node	$P(x\|a) = \delta(x, x_0)$. x_0 does not depend on a whereas for decision node it does. deterministic.

Table 5.2: dtree node types.

$P(x\|a)$	states of parent node with $a = a_0$	states of parent node with $a =!a_0$	$a = null$
$[x \in bin]_{\forall\, bin \subset [L,\alpha)}$	1	0	0
$[x \in bin]_{\forall\, bin \subset [\alpha,U]}$	0	0	0
$x = null$	0	1	1

Table 5.3: Transition probability matrix for interval splitting node.

A naive Bayes bnet (see Chapter 23) consists of a single "class" node that fans out with arrows pointing to other "feature" nodes. If each leaf node of a naive Bayes bnet fans out into a set of new leaf nodes, and those new leaf nodes also fan out and so on recursively, we get a tree bnet. The bnet that arises from this recursive application of naive Bayes has the same graph structure as the image bnet of a dtree. However, it is more general because its node TPMs are more general; it has more weights (weight= parameters of node TPMs) than a dtree with the same graph.

Chapter 6

Digital Circuits

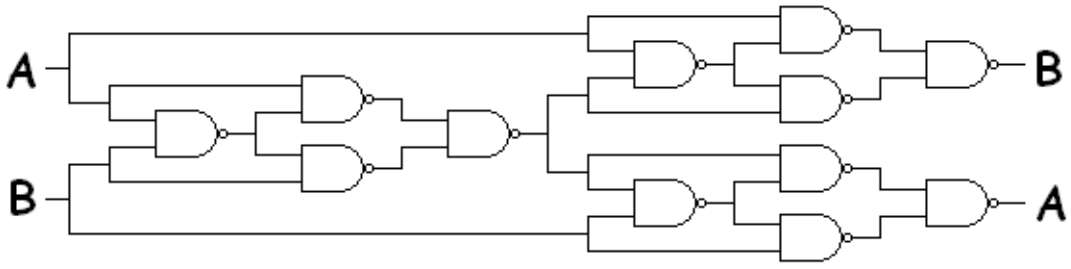


Figure 6.1: Typical digital circuit of NAND gates.

Digital (logic) gate: node with na input ports and nx output ports which represents a function

$$f : \{0,1\}^{na} \to \{0,1\}^{nx} \ . \tag{6.1}$$

Suppose
$a^{na} = (a_i)_{i=0,1,\ldots,na-1}$ where $a_i \in \{0,1\}$,
$x^{nx} = (x_i)_{i=0,1,\ldots,nx-1}$ where $x_i \in \{0,1\}$.
f maps a^{na} into x^{nx}.
Digital circuit (dcircuit) = circuit of digital gates.

Mapping any dcircuit to a bnet (Option A- See Fig.6.2)):

1. Replace every dcircuit gate described by Eq.(6.1) by nx bnet nodes $\underline{x}_i$ for $i = 0, 1, \ldots, nx-1$ such that

$$P(x_i|a^{na}) = \delta(x_i, f_i(a^{na})) \tag{6.2}$$

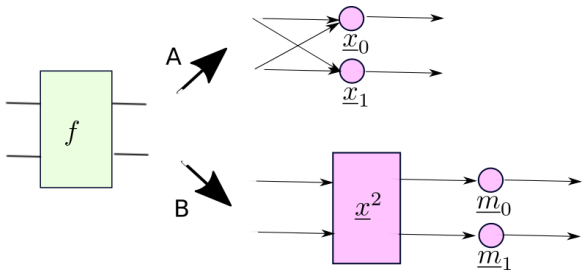


Figure 6.2: 2 options for mapping dcircuit node with multiple output ports into bnet.

2. Replace all connectors of the dcircuit by arrows pointing in the direction of the bit flow.

Mapping any dcircuit to a bnet (Option B- See Fig.6.2)):

1. Replace every dcircuit gate described by Eq.(6.1) with one bnet node called $\underline{x}^{nx}$ and, if $nx > 0$, nx "marginalizer nodes" $\underline{m}_i$ for $i = 0, 1, \ldots, nx - 1$, such that

$$P(x^{nx}|a^{na}) = \delta(x^{nx}, f(a^{na})) \ , \tag{6.3}$$

   and

$$P(m_i|x^{nx}) = \delta(m_i, x_i)) \ . \tag{6.4}$$

2. Replace all connectors of the dcircuit by arrows pointing in the direction of the bit flow.

Options A and B don't work for digital circuits with feedback loops such as flip-flops. Those could probably be modeled with dynamical bnets.

Chapter 7

Do-Calculus: COMING SOON

Chapter 8

D-Separation: COMING SOON

Chapter 9

Dynamical Bayesian Networks: COMING SOON

Chapter 10

Expectation Maximization

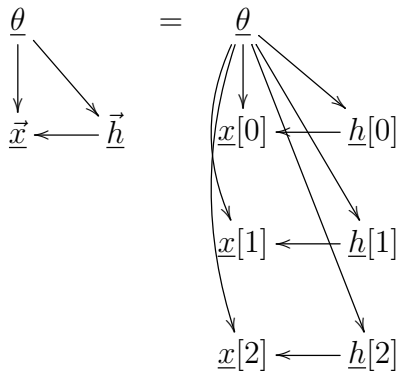


Figure 10.1: bnet for EM with $nsam = 3$.

This chapter is based on Wikipedia Ref.[4].

The bnet for Expectation Maximization (EM) is given by Fig.10.1 for $nsam = 3$. Later on in this chapter, we will give the node TPMs for this bnet for the special case in which $P(x[i] \mid \theta)$ is a mixture (i.e., weighted sum) of Gaussians.

Note that if we erase the $\underline{h}[i]$ nodes from Fig.10.1, we get the bnet for naive Bayes, which is used for classification into the states of $\underline{\theta}$. However, there is one big difference. With naive Bayes, the leaf nodes have different TPMs. Here, we will assume they are i.i.d. Naive Bayes is used for classification: i.e., given the states of the leaf nodes, we infer the state of the root node. EM is used for clustering; i.e., given many i.i.d. samples, we fit their distribution by a weighted sum of prob distributions, usually Gaussians.

Let

$\mathcal{L}$ =likelihood function.

$nsam$ = number of samples.

$\vec{x} = (x[0], x[1], \ldots, x[nsam-1])$ = **observed data**. $x[i] \in S_{\underline{x}}$ for all i.

$\vec{h} = (h[0], h[1], \ldots, h[nsam-1])$ = **hidden or missing data**. $h[i] \in S_{\underline{h}}$ for all i.

We assume that the samples $(x[i], h[i])$ are i.i.d. for different i at fixed θ. What this means is that there are probability distributions $P_{\underline{x}|\underline{h},\underline{\theta}}$ and $P_{\underline{h}|\underline{\theta}}$ such that

$$P(\vec{x}, \vec{h}|\theta) = \prod_i \left[P_{\underline{x}|\underline{h},\underline{\theta}}(x[i] \mid h[i], \theta) P_{\underline{h}|\underline{\theta}}(h[i] \mid \theta) \right] \ . \tag{10.1}$$

Definition of likelihood functions:

$$\underbrace{P(\vec{x}|\theta)}_{\mathcal{L}(\theta;\vec{x})} \ = \ \sum_{\vec{h}} \underbrace{P(\vec{x}, \vec{h}|\theta)}_{\mathcal{L}(\theta;\vec{x},\vec{h})} \tag{10.2}$$

θ^* = maximum likelihood estimate of θ (no prior $P(\theta)$ assumed):

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, \mathcal{L}(\theta; \vec{x}) \tag{10.3}$$

The EM algorithm:

1. **Expectation step:**
$$Q(\theta|\theta^{(t)}) = E_{\vec{h}|\vec{x},\theta^{(t)}} \ln P(\vec{x}, \vec{h}|\theta) \tag{10.4}$$

2. **Maximization step:**
$$\theta^{(t+1)} = \underset{\theta}{\operatorname{argmax}}\, Q(\theta|\theta^{(t)}) \tag{10.5}$$

Claim: $\lim_{t\to\infty} \theta^{(t)} = \theta^*$.

Motivation

$$\begin{aligned} Q(\theta|\theta) &= E_{\vec{h}|\vec{x},\theta} \ln P(\vec{x}, \vec{h}|\theta) && (10.6)\\ &= E_{\vec{h}|\vec{x},\theta}[\ln P(\vec{h}|\vec{x}, \theta) + \ln P(\vec{x}|\theta)] && (10.7)\\ &= -H[P(\underline{\vec{h}}|\vec{x}, \theta)] + \ln P(\vec{x}|\theta) && (10.8) \end{aligned}$$

$$\begin{aligned} \partial_\theta Q(\theta|\theta) &= -\sum_{\vec{h}} \partial_\theta P(\underline{\vec{h}}|\vec{x}, \theta) + \partial_\theta \ln P(\vec{x}|\theta) && (10.9)\\ &= \partial_\theta \ln P(\vec{x}|\theta) && (10.10) \end{aligned}$$

So if $\theta^{(t)} \to \theta$ and $Q(\theta|\theta)$ is max at $\theta = \theta^*$, then $\ln P(\vec{x}|\theta)$ is max at $\theta = \theta^*$ too.

For a more rigorous proof that $\lim_{t\to\infty} \theta^{(t)} = \theta^*$, see Wikipedia article Ref.[4] and references therein.

EM for Gaussian mixture

$x[i] \in \mathbb{R}^d = S_{\underline{x}}$. $S_{\underline{h}}$ discrete and not too large. $n_{\underline{h}} = |S_{\underline{h}}|$ is number of Gaussians that we are going to fit the samples with.

Let

$$\theta = [w_h, \mu_h, \Sigma_h]_{h \in S_{\underline{h}}} \ , \tag{10.11}$$

where $[w_h]_{h \in S_{\underline{h}}}$ is a probability distribution of weights, and where $\mu_h \in \mathbb{R}^d$ and $\Sigma_h \in \mathbb{R}^{d \times d}$ are the mean value vector and covariance matrix of a d-dimensional Gaussian distribution.

The TPMs, printed in blue, for the nodes of Fig.10.1, for the special case of a mixture of Gaussians, are as follows:

$$P(x[i] \mid h[i] \mid \theta) = \mathcal{N}_d(x[i]; \mu_{h[i]}, \Sigma_{h[i]}) \tag{10.12}$$

$$P(h[i] \mid \theta) = w_{h[i]} \tag{10.13}$$

Note that

$$\begin{aligned} P(x[i] \mid \theta) &= \sum_h P(x[i] \mid h[i] = h, \theta) P(h[i] = h \mid \theta) && (10.14) \\ &= \sum_h w_h \mathcal{N}_d(x[i]; \mu_h, \Sigma_h) && (10.15) \end{aligned}$$

$$\begin{aligned} P(\vec{x}, \vec{h} | \theta) &= \prod_i \left[w_{h[i]} \mathcal{N}_d(x[i]; \mu_{h[i]}, \Sigma_{h[i]}) \right] && (10.16) \\ &= \prod_i \prod_h \left[w_h \mathcal{N}_d(x[i]; \mu_h, \Sigma_h) \right]^{\mathbb{1}(h = h[i])} && (10.17) \end{aligned}$$

Old Faithful: See Wikipedia Ref.[4] for an animated gif of a classic example of using EM to fit samples with a Gaussian mixture. Unfortunately, could not include it here because pdflatex does not support animated gifs. The gif shows samples in a 2 dimensional space (eruption time, delay time) from the Old Faithful geyser. In that example, $d = 2$ and $n_{\underline{h}} = 2$. Two clusters of points in a plane are fitted by a mixture of 2 Gaussians.

K-means clustering is often presented as the main competitor to EM for doing **clustering (non-supervised learning)**. In K-means clustering, the sample points are split into K mutually disjoint sets $S_0, S_1, \ldots, S_{K-1}$. The algorithm is easy to describe:

1. Initialize by choosing at random K data points $(\mu_k)_{k=0}^{K-1}$ called means or centroids and placing μ_k in S_k for all k.

2. **STEP 1:** For each data point, add it to the S_k whose centroid μ_k is closest to it.

3. **STEP 2:** Recalculate the centroids. Set μ_k equal to the mean value of set S_k.

4. Repeat steps 1 and 2 until the centroids stop changing by much.

Step 1 is analogous to the expectation step in EM, and Step 2 to the maximization step in EM (θ estimation versus μ_k estimation). We won't say anything further about K-means clustering because it isn't related to bnets in any way, and this is a book about bnets. For more info about K-means clustering, see Ref.[5].

Chapter 11

Generative Adversarial Networks (GANs)

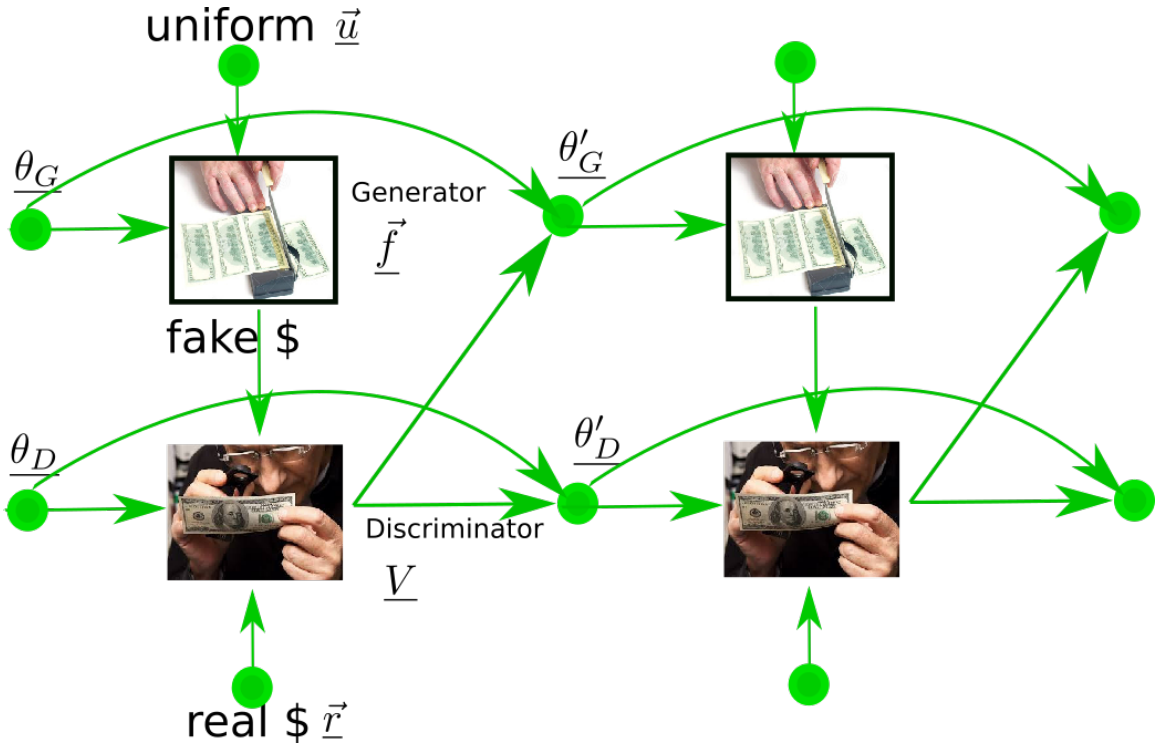


Figure 11.1: Generative Adversarial Network (GAN)

Original GAN, Ref.[6](2014).

Generator G (counterfeiter) generates samples $\vec{f}$ of fake money and submits them to Discriminator D (Treasury agent). D also gets samples $\vec{r}$ of real money. D submits veredict $V \in [0, 1]$. G depends on parameter θ_G and D on parameter θ_D. Veredict V and initial θ_G, θ_D are used to get new parameters θ'_G, θ'_D.Process is repeated (Dynamical Bayesian Network) until saddle point in

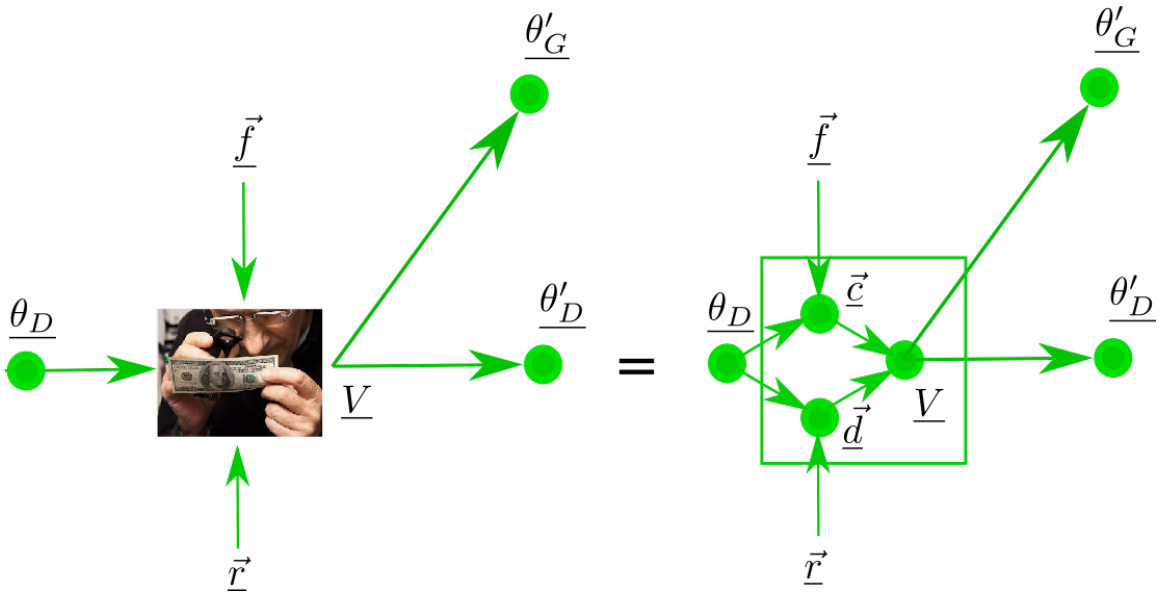


Figure 11.2: Discriminator node $\underline{V}$ in Fig.11.1 can be split into 3 nodes $\vec{\underline{c}}$, $\vec{\underline{d}}$ and $\underline{V}$.

$V(\theta_G, \theta_D)$ is reached. D makes G better and vice versa. Zero-sum game between D and G.
Let $\mathcal{D}$ be the domain of $D(\cdot, \theta_D)$. Assume that for any $x \in \mathcal{D}$,

$$0 \leq D(x, \theta_D) \leq 1 \; . \tag{11.1}$$

For any $S \subset \mathcal{D}$, define

$$\sum_{x \in S} D(x, \theta_D) = \lambda(S, \theta_D) \; . \tag{11.2}$$

In general, $G(\cdot, \theta_G)$ need not be real valued.
Assume that for every $u \in S_{\underline{u}}$, $G(u, \theta_G) = f \in S_{\underline{f}} \subset \mathcal{D}$. Define

$$\overline{D}(f, \theta_D) = 1 - D(f, \theta_D) \; . \tag{11.3}$$

Note that

$$0 \leq \overline{D}(f, \theta_D) \leq 1 \; . \tag{11.4}$$

Define:

$$V(\theta_G, \theta_D) = \sum_r P(r) \ln D(r, \theta_D) + \sum_u P(u) \ln \overline{D}(G(u, \theta_G), \theta_D) \; . \tag{11.5}$$

We want the first variation of $V(\theta_G, \theta_D)$ to vanish.

$$\delta V(\theta_G, \theta_D) = 0 \; . \tag{11.6}$$

This implies

$$\partial_{\theta_G} V(\theta_G, \theta_D) = \partial_{\theta_D} V(\theta_G, \theta_D) = 0 \tag{11.7}$$

and

$$V_{opt} = \min_{\theta_G} \max_{\theta_D} V(\theta_G, \theta_D) \ . \tag{11.8}$$

Node TPMs for Figs.11.1 and 11.2 are given next in blue:

$$P(\theta_G) = \text{ given} \tag{11.9}$$

$$P(\theta_D) = \text{ given} \tag{11.10}$$

$$P(\vec{u}) = \prod_i P(u[i]) \ \text{ (usually uniform distribution)} \tag{11.11}$$

$$P(\vec{r}) = \prod_i P(r[i]) \tag{11.12}$$

$$P(f[i] \mid \vec{u}, \theta_G) = \delta[f[i], G(u[i], \theta_G)] \tag{11.13}$$

$$P(c[i] \mid \vec{f}, \theta_D) = \delta(c[i], \overline{D}(f[i], \theta_D)) \tag{11.14}$$

$$P(d[j] \mid \vec{r}, \theta_D) = \delta(d[j], D(r[j], \theta_D)) \tag{11.15}$$

$$P(V|\vec{d}, \vec{c}) = \delta(V, \frac{1}{N} \ln \prod_{i,j} (c[i]d[j])) \tag{11.16}$$

where $N = nsam(\vec{r})nsam(\vec{u})$.

Let $\eta_G, \eta_D > 0$. Maximize V wrt θ_D, and minimize it wrt θ_G.

$$P(\theta'_G|V, \theta_G) = \delta(\theta'_G, \theta_G - \eta_G \partial_{\theta_G} V) \tag{11.17}$$

$$P(\theta'_D|V, \theta_D) = \delta(\theta'_D, \theta_D + \eta_D \partial_{\theta_D} V) \tag{11.18}$$

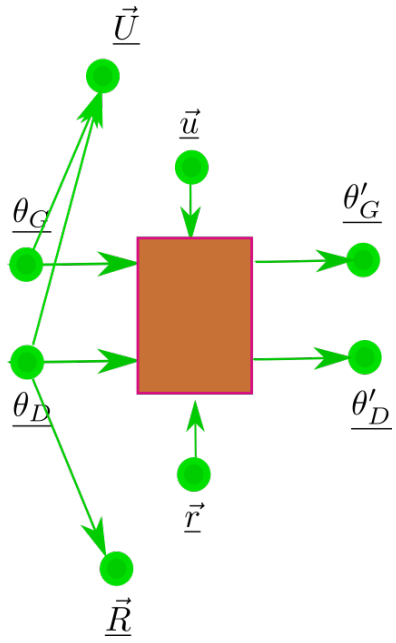


Figure 11.3: GAN, Constraining Bayesian Network

Constraining B net given in Fig.11.3. It adds 2 new nodes, namely $\underline{\vec{U}}$ and $\underline{\vec{R}}$, to the bnet of Fig.11.1. The purpose of these 2 barren (childrenless) nodes is to constrain certain functions to be probability distributions.

Node TPMs for the 2 new nodes given next in blue.

$$P(U[i] \mid \theta_G) = \frac{\overline{D}(G(U[i], \theta_G), \theta_D))}{\overline{\lambda}(\theta_G, \theta_D)} \tag{11.19}$$

where $S_{\underline{U[i]}} = S_{\underline{u}}$ and $\overline{\lambda}(\theta_G, \theta_D) = \sum_u \overline{D}(G(u, \theta_G), \theta_D))$.

$$P(R[i] \mid \theta_G, \theta_D) = \frac{D(R[i], \theta_D)}{\lambda(\theta_D)} \tag{11.20}$$

where $S_{\underline{R[i]}} = S_{\underline{r}}$ and $\lambda(\theta_D) = \sum_r D(r, \theta_D)$.

$$P(V|\vec{u}, \vec{r}) = \delta(V, \frac{1}{N} \ln \prod_{i,j} (P(\underline{R[i]} = r[i] \mid \theta_G, \theta_D) P(\underline{U[i]} = u[j] \mid \theta_G))) \tag{11.21}$$

where $N = nsam(\vec{r}) nsam(\vec{u})$.

$\mathcal{L}$ = likelihood

$$\mathcal{L} = P(\vec{r}, \vec{u}|\theta_G, \theta_D) \tag{11.22}$$

$$= \prod_{i,j} \left[\frac{D(r[i], \theta_D)}{\lambda(\theta_D)} \frac{\overline{D}(G(u[j], \theta_G), \theta_D))}{\overline{\lambda}(\theta_G, \theta_D)} \right] \tag{11.23}$$

$$\ln \mathcal{L} = N[V(\theta_G, \theta_D) - \ln \lambda(\theta_D) - \ln \overline{\lambda}(\theta_G, \theta_D)] \tag{11.24}$$

Chapter 12

Graph Structure Learning for bnets: COMING SOON

Chapter 13

Hidden Markov Model

A Hidden Markov Model (HMM) is a generalization of a Kalman Filter (KF). KFs are discussed in Chapter 16. The bnets of HHMs and KFs bnets are the same. The only difference is that a KF assumes special node TPMs.

See Wikipedia article Ref.[7] to learn about the history and many uses of HMMs. This chapter is based on Ref.[8].

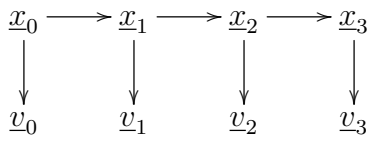


Figure 13.1: HMM bnet with $n = 4$.

Suppose

$\underline{v}^n = (\underline{v}_0, \underline{v}_1, \ldots, \underline{v}_{n-1})$ are n visible nodes that are measured, and

$\underline{x}^n = (\underline{x}_0, \underline{x}_1, \ldots, \underline{x}_{n-1})$ are the n hidden, unmeasurable state nodes of a system that is being monitored.

For the bnet of Fig.13.1, one has

$$P(x^n, v^n) = \prod_{i=0}^{n-1} P(x_i|x_{i-1})P(v_i|x_i) \ , \tag{13.1}$$

where $x_{-1} = 0$.

Let $x_{<i} = (x_0, x_1, \ldots, x_{i-1})$.

For $i = 0, 1, \ldots, n-1$, define

$\mathcal{F}_i$=future measurements probability

$$\mathcal{F}_i(x_i) = P(v_{>i}|x_i) \tag{13.2}$$

$\overline{\mathcal{F}}_i$= past and present measurements probability

$$\overline{\mathcal{F}}_i(x_i) = P(v_{<i}, v_i, x_i) \tag{13.3}$$

λ_i= present measurement probability

$$\lambda_i(x_i) = P(v_i|x_i) \tag{13.4}$$

$\mathcal{F}_i$, $\overline{\mathcal{F}}_i$ and λ_i can be represented graphically as follows:

$$\mathcal{F}_i(x_i) = \tfrac{1}{P(x_i)} \sum_{x_{>i}} \quad \begin{array}{ccc} x_i & \longrightarrow & x_{>i} \\ & & \downarrow \\ & & v_{>i} \end{array} \tag{13.5}$$

$$\overline{\mathcal{F}}_i(x_i) = \sum_{x_{<i}} \quad \begin{array}{ccc} x_{<i} & \longrightarrow & x_i \\ \downarrow & & \downarrow \\ v_{<i} & & v_i \end{array} \tag{13.6}$$

$$\lambda_i(x_i) = \tfrac{1}{P(x_i)} \quad \begin{array}{c} x_i \\ \downarrow \\ v_i \end{array} \tag{13.7}$$

Claim 1 *For $i \geq 0$,*

$$P(x_i, v^n) = \overline{\mathcal{F}}_i(x_i)\mathcal{F}_i(x_i) \ . \tag{13.8}$$

For $i > 0$,

$$P(x_{i-1}, x_i, v^n) = \overline{\mathcal{F}}_{i-1}(x_{i-1})\lambda_i(x_i)P(x_i|x_{i-1})\mathcal{F}_i(x_i) \ . \tag{13.9}$$

proof:

$$\begin{aligned} P(x_i, v^n) &= \sum_{x_{<i}} \sum_{x_{>i}} P(x^n, v^n) && (13.10)\\ &= \sum_{x_{<i}} \sum_{x_{>i}} P(x^n, v^n|x_i)P(x_i) && (13.11)\\ &= \sum_{x_{<i}} \sum_{x_{>i}} P(x_{<i}, v_{<i}, v_i|x_i)P(x_{>i}, v_{>i}|x_i)P(x_i) && (13.12)\\ &= P(v_{<i}, v_i|x_i)P(v_{>i}|x_i)P(x_i) && (13.13)\\ &= \overline{\mathcal{F}}_i(x_i)\mathcal{F}_i(x_i) && (13.14) \end{aligned}$$

$$
\begin{aligned}
P(x_{i-1}, x_i, v^n) &= \sum_{x_{<i-1}} \sum_{x_{>i}} P(x^n, v^n) && (13.15)\\
&= \sum_{x_{<i-1}} \sum_{x_{>i}} P(x^n, v^n | x_{i-1}, x_i) P(x_{i-1}, x_i) && (13.16)\\
&= \sum_{x_{<i-1}} \sum_{x_{>i}} P(x_{<i-1}, v_{<i-1}, v_{i-1} | x_{i-1}) P(v_i | x_i) P(x_{i-1}, x_i) P(x_{>i}, v_{>i} | x_i) && (13.17)\\
&= P(v_{<i-1}, v_{i-1} | x_{i-1}) P(v_i | x_i) P(x_{i-1}, x_i) P(v_{>i} | x_i) && (13.18)\\
&= \overline{\mathcal{F}}_{i-1}(x_{i-1}) \lambda_i(x_i) P(x_i | x_{i-1}) \mathcal{F}_i(x_i) && (13.19)
\end{aligned}
$$

QED

Claim 2 *For $i > 0$, $\mathcal{F}_i$ and $\overline{\mathcal{F}}_i$ can be calculated recursively as follows:*

$$\overline{\mathcal{F}}_i(x_i) = \sum_{x_{i-1}} \overline{\mathcal{F}}_{i-1}(x_{i-1}) \lambda_i(x_i) P(x_i | x_{i-1}) \tag{13.20}$$

$$\mathcal{F}_{i-1}(x_{i-1}) = \sum_{x_i} \lambda_i(x_i) P(x_i | x_{i-1}) \mathcal{F}_i(x_i) \tag{13.21}$$

proof:

$$
\begin{aligned}
\overline{\mathcal{F}}_i(x_i) \mathcal{F}_i(x_i) &= P(x_i, v^n) && (13.22)\\
&= \sum_{x_{i-1}} P(x_{i-1}, x_i, v^n) && (13.23)\\
&= \sum_{x_{i-1}} \overline{\mathcal{F}}_{i-1}(x_{i-1}) \lambda_i(x_i) P(x_i | x_{i-1}) \mathcal{F}_i(x_i) && (13.24)
\end{aligned}
$$

$$
\begin{aligned}
\overline{\mathcal{F}}_{i-1}(x_{i-1}) \mathcal{F}_{i-1}(x_{i-1}) &= P(x_{i-1}, v^n) && (13.25)\\
&= \sum_{x_i} P(x_{i-1}, x_i, v^n) && (13.26)\\
&= \sum_{x_i} \overline{\mathcal{F}}_{i-1}(x_{i-1}) \lambda_i(x_i) P(x_i | x_{i-1}) \mathcal{F}_i(x_i) && (13.27)
\end{aligned}
$$

QED

Claim 3

$$P(x_i | x_{i-1}, v^n) = \frac{\lambda_i(x_i) \mathcal{F}_i(x_i)}{\mathcal{F}_{i-1}(x_{i-1})} P(x_i | x_{i-1}) \tag{13.28}$$

$$P(x_{i-1} | x_i, v^n) = \frac{\lambda_i(x_i) \overline{\mathcal{F}}_{i-1}(x_{i-1})}{\overline{\mathcal{F}}_i(x_i)} P(x_i | x_{i-1}) \tag{13.29}$$

proof:

$$P(x_i|x_{i-1},v^n) = \frac{P(x_{i-1},x_i,v^n)}{P(x_{i-1},v^n)} \tag{13.30}$$

$$= \frac{\overline{\mathcal{F}}_{i-1}(x_{i-1})\lambda_i(x_i)P(x_i|x_{i-1})\mathcal{F}_i(x_i)}{\overline{\mathcal{F}}_{i-1}(x_{i-1})\mathcal{F}_{i-1}(x_{i-1})} \tag{13.31}$$

Analogous proof for Eq.(13.29).
QED

Chapter 14

Influence Diagrams & Utility Nodes

Influence diagrams are just arbitrary bnets enhanced with a new kind of node called an utility node. The rest of this brief chapter will be devoted to discussing utility nodes.

An utility node can be understood as a node composed of 3 simpler bnet nodes. This is illustrated in Fig.14.1.

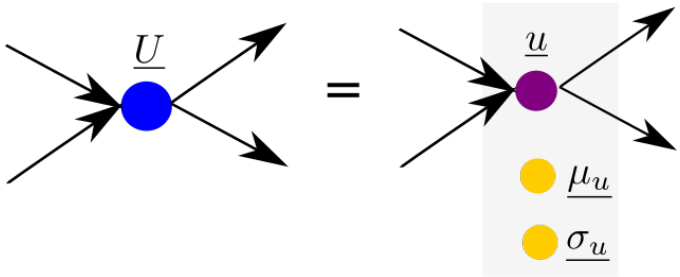


Figure 14.1: An utility node can be understood as a node composed of 3 simpler bnet nodes.

The TPMs, printed in blue, for the nodes of Fig.14.1, are as follows:

$$P(U|pa(U)) = \text{given} \tag{14.1}$$

$$P(u|pa(U)) = P(U = u|pa(U)) \tag{14.2}$$

Node $\underline{\mu}_u$ calculates the expected value (mean value) of $\underline{u}$:

$$P(\mu_u) = \delta(\mu_u, E_{\underline{u}}[\underline{u}]) \tag{14.3}$$

Node $\underline{\sigma}_u$ calculates the standard deviation of $\underline{u}$:

$$P(\sigma_u) = \delta(\sigma_U, \sqrt{E_{\underline{u}}[(\underline{u} - E_{\underline{u}}[\underline{u}])^2]}) \tag{14.4}$$

Note that in order to calculate expected values, it is necessary that $\underline{U}, \underline{u} \in \mathbb{R}$. Note that nodes $\underline{u}$, $\underline{\mu}_u$, $\underline{\sigma}_u$ must all 3 have access to the TPM $P(U|pa(U))$ of node $\underline{U}$. In fact, in order to calculate $E_{\underline{u}}[\cdot]$, it is necessary for nodes $\underline{\mu}_u$ and $\underline{\sigma}_u$ to have access not just to $P(U|pa(U))$ but also to $P(pa(U))$.

See Fig.14.2. An influence diagram may have multiple utility nodes ($\underline{U}_1$ and $\underline{U}_2$ in Fig.14.2). Then one can define a merging utility node $\underline{U}$ that sums the values of all the other utility nodes.

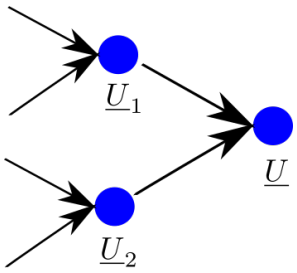


Figure 14.2: An influence diagram may have multiple utility nodes, say $\underline{U}_1$ and $\underline{U}_2$. Then one can define an utility node $\underline{U} = \underline{U}_1 + \underline{U}_2$.

For the node $\underline{U}$ of Fig.14.2,

$$P(U|U_1, U_2) = \delta(U, U_1 + U_2) \tag{14.5}$$

Chapter 15

Junction Tree Algorithm

The Junction Tree (JT) algorithm is an algo for evaluating exact marginals of a bnet, including cases in which some nodes are fixed to a single state. (fixed nodes are called the a priori evidence.)

The JT algo starts by clustering the loops of a bnet into bigger nodes so as to transform the bnet into a tree bnet. Then it applies Pearl Belief Propagation (see Chapter 21) to the ensuing tree. The first breakthrough paper to achieve this agenda in full was Ref.[9] by Lauritzen, and Spiegelhalter in 1988. See the Wikipedia article Ref.[10] for more info and references on the JT algorithm.

I won't describe the JT algo any further here, because it would take too long for this brief book to give a complete treatment of it, including the mathematical proofs. If all you want to do is to code the JT algo, without delving into the mathematical theorems and proofs behind it, I strongly recommend Ref.[11]. Ref.[11] is an excellent cookbook for programmers of the JT algo. My open source program QuantumFog (see Ref.[12]) implements the JT algo in Python, following the recipe of Ref.[11].

Chapter 16

Kalman Filter

A Kalman Filter is a special case of a Hidden Markov Model. HMMs are discussed in Chapter 13.

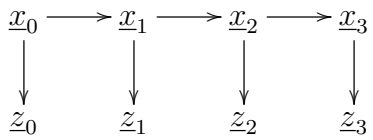


Figure 16.1: Kalman Filter bnet with $T = 4$.

Let $t = 0, 1, 2, \ldots, T-1$.
$\underline{x}_t \in S_{\underline{x}}$ are random variables that represent the hidden (unobserved) true state of the system.
$\underline{z}_t \in S_{\underline{z}}$ are random variables that represent the measured (observed) state of the system.
The Kalman Filter bnet Fig.16.1 has the following node TPMs, printed in blue:

$$P(x_t|x_{t-1}) = \mathcal{N}(F_t x_{t-1} + B_t u_t, Q_t) \ , \tag{16.1}$$

where F_t, Q_t, B_t, u_t are given. $P(x_t|x_{t-1})$ becomes $P(x_t)$ for $t = 0$.

$$P(z_t|x_t) = \mathcal{N}(H_t x_t, R_t) \ , \tag{16.2}$$

where H_t, R_t are given.
Define

$$\underline{Z}_t = (\underline{z}_{t'})_{t' \le t} \ . \tag{16.3}$$

Define $\hat{x}_t$ and P_t by

$$P(x_t|Z_t) = \mathcal{N}(\hat{x}_t, P_t) \ . \tag{16.4}$$

Problem: Find $\hat{x}_t$ and P_t in terms of

1. current (at time t) given values of F, Q, H, R, B, u

2. current (at time t) observed value of z

3. prior (previous) value (at time $t-1$) of $\hat{x}$ and P.

See Fig.16.2. For that figure,

$$P(\hat{x}_t, P_t | z_t, \hat{x}_{t-1}, P_{t-1}) = \delta(\hat{x}_t, ?)\delta(P_t, ?) \; . \tag{16.5}$$

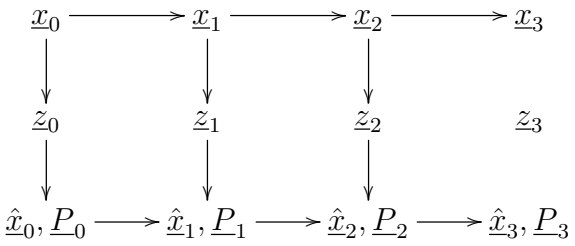


Figure 16.2: Kalman Filter bnet with deterministic nodes for $\hat{x}_t, P_t$.

Solution copied from Wikipedia Ref.[13]:

Define $\eta_{t|t} = \eta_t$ for $\eta = \hat{x}, P$.

- **Predict**

  Predicted (a priori) state estimate

$$\hat{x}_{t|t-1} = F_t \hat{x}_{t-1|t-1} + B_t u_t \tag{16.6}$$

  Predicted (a priori) estimate covariance

$$P_{t|t-1} = F_t P_{t-1|t-1} F_t^{\mathsf{T}} + Q_t \tag{16.7}$$

- **Update**

  Innovation (or measurement pre-fit residual)

$$\tilde{y}_{t|t-1} = z_t - H_t \hat{x}_{t|t-1} \tag{16.8}$$

  Innovation (or pre-fit residual) covariance

$$S_t = H_t P_{t|t-1} H_t^{\mathsf{T}} + R_t \tag{16.9}$$

Optimal Kalman gain

$$K_t = P_{t|t-1} H_t^{\mathsf{T}} S_t^{-1} \tag{16.10}$$

Updated (a posteriori) state estimate

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t \tilde{y}_t \tag{16.11}$$

Updated (a posteriori) estimate covariance

$$P_{t|t} = (I - K_t H_t) P_{t|t-1} \tag{16.12}$$

Measurement post-fit residual

$$\tilde{y}_{t|t} = z_t - H_t \hat{x}_{t|t} \tag{16.13}$$

Chapter 17

Linear and Logistic Regression

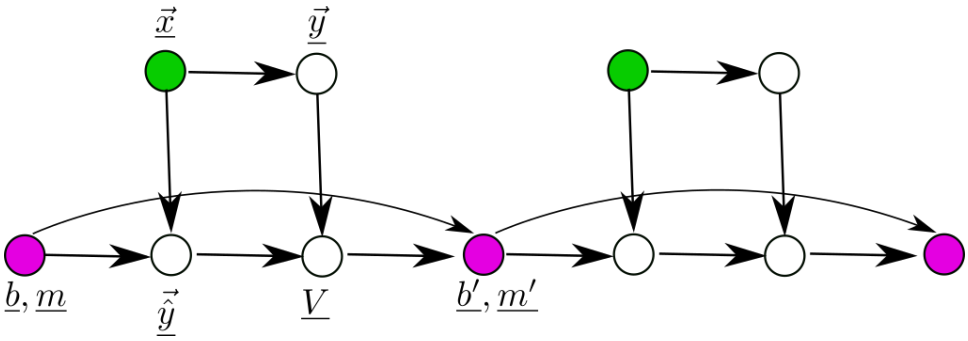


Figure 17.1: Linear Regression

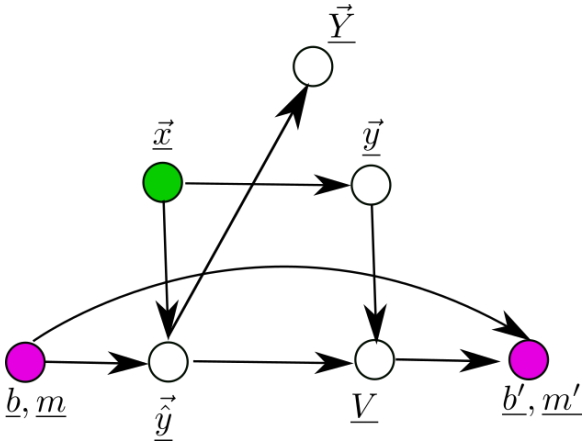


Figure 17.2: B net of Fig.17.1 with new $\vec{Y}$ node.

Estimators $\hat{y}$ for linear and logistic regression.

- **Linear Regression:** $y \in \mathbb{R}$. Note $\hat{y} \in \mathbb{R}$. $(x, \hat{y}(x))$ is the graph of a straight line with y-intercept b and slope m.

$$\hat{y}(x; b, m) = b + mx \tag{17.1}$$

- **Logistic Regression:** $y \in \{0,1\}$. Note $\hat{y} \in [0,1]$. $(x, \hat{y}(x))$ is the graph of a sigmoid. Often in literature, b, m are replaced by β_0, β_1.

$$\hat{y}(x; b, m) = \text{sig}(b + mx) \tag{17.2}$$

Define

$$V(b,m) = \sum_{x,y} P(x,y)|y - \hat{y}(x;b,m)|^2 \ . \tag{17.3}$$

We want to minimize $V(b,m)$ (called a cost or loss function) wrt b and m.

Node TPMs of B net of Fig.17.1 given next in blue.

$$P(b,m) = \text{given} \tag{17.4}$$

The first time it is used, (b,m) is arbitrary. After the first time, it is determined by previous stage.

Let

$$P_{\underline{x},\underline{y}}(x,y) = \frac{1}{nsam(\vec{x})} \sum_i \mathbb{1}(x = x[i], y = y[i]) \ . \tag{17.5}$$

$$P(\vec{x}) = \prod_i P(x[i]) \tag{17.6}$$

$$P(\vec{y}|\vec{x}) = \prod_i P(y[i] \mid x[i]) \tag{17.7}$$

$$P(\vec{\hat{y}}|\vec{x}, b, m) = \prod_i \delta(\hat{y}[i], \hat{y}(x[i], b, m)) \tag{17.8}$$

$$P(V|\vec{\hat{y}}, \vec{y}) = \delta(V, \frac{1}{nsam(\vec{x})} \sum_i |y[i] - \hat{y}[i]|^2) \tag{17.9}$$

Let $\eta_b, \eta_m > 0$. For $x = b, m$, if $x' - x = \Delta x = -\eta \frac{\partial V}{\partial x}$, then $\Delta V \approx \frac{-1}{\eta}(\Delta x)^2 \leq 0$ for $\eta > 0$. This is called "gradient descent".

$$P(b'|V, b) = \delta(b', b - \eta_b \partial_b V) \tag{17.10}$$

$$P(m'|V, m) = \delta(m', m - \eta_m \partial_m V) \tag{17.11}$$

Generalization to x with multiple components(features)

Suppose that for each sample i, instead of $x[i]$ being a scalar, it has n components called features:

$$x[i] = (x_0[i], x_1[i], x_2[i], \ldots x_{n-1}[i]) \ . \tag{17.12}$$

Slope m is replaced by weights

$$w = (w_0, w_1, w_3, , \ldots, w_{n-1}) \ , \tag{17.13}$$

and the product of 2 scalars $mx[i]$ is replaced by the inner vector product $w^T x[i]$.

Alternative $V(b, m)$ for logistic regression

For logistic regression, since $y[i] \in \{0, 1\}$ and $\hat{y}[i] \in [0, 1]$ are both in the interval $[0, 1]$, they can be interpreted as probabilities. Define probability distributions $p[i](x)$ and $\hat{p}[i](x)$ for $x \in \{0, 1\}$ by

$$p[i](1) = y[i], \quad p[i](0) = 1 - y[i] \tag{17.14}$$

$$\hat{p}[i](1) = \hat{y}[i], \quad \hat{p}[i](0) = 1 - \hat{y}[i] \tag{17.15}$$

Then for logistic regression, the following 2 cost functions $V(b, m)$ can be used as alternatives to the cost function Eq.(17.3) previously given.

$$V(b, m) = \frac{1}{nsam(\vec{x})} \sum_i D_{KL}(p[i] \parallel \hat{p}[i]) \tag{17.16}$$

and

$$\begin{aligned} V(b, m) &= \frac{1}{nsam(\vec{x})} \sum_i CE(p[i] \to \hat{p}[i]) && (17.17) \\ &= \frac{-1}{nsam(\vec{x})} \sum_i \{y[i] \ln \hat{y}[i] + (1 - y[i]) \ln(1 - \hat{y}[i])\} && (17.18) \\ &= \frac{-1}{nsam(\vec{x})} \sum_i \ln \left\{\hat{y}[i]^{y[i]} (1 - \hat{y}[i])^{(1-y[i])}\right\} && (17.19) \\ &= \frac{-1}{nsam(\vec{x})} \sum_i \ln P(\underline{Y} = y[i] \mid \hat{y} = \hat{y}[i]) && (17.20) \\ &= -\sum_{x,y} P(x, y) \ln P(\underline{Y} = y | \hat{y} = \hat{y}(x, b, m)) && (17.21) \end{aligned}$$

Above, we used

$$P(\underline{Y} = Y | \hat{y}) = \hat{y}^Y [1 - \hat{y}]^{1-Y} \tag{17.22}$$

for $Y \in S_{\underline{Y}} = \{0, 1\}$. (Bernoulli distribution).

There is no node corresponding to $\underline{Y}$ in the B net of Fig.17.1. Fig.17.2 shows a new B net that has a new node called $\underline{\vec{Y}}$ compared to the B net of Fig.17.1. One defines the TPMs for all nodes of Fig.17.2 except $\underline{\vec{Y}}$ and $\underline{V}$ the same as for Fig.17.1. For $\underline{\vec{Y}}$ and $\underline{V}$, one defines

$$P(Y[i] \mid \vec{\hat{y}}) = P(\underline{Y} = Y[i] \mid \hat{y}[i]) \tag{17.23}$$

$$P(V|\vec{Y}, \vec{y}) = \delta(V, \frac{-1}{nsam(\vec{x})} \ln \mathcal{L}) \ , \tag{17.24}$$

where $\mathcal{L} = \prod_i P(\underline{Y} = y[i] \mid \hat{y}[i])$=likelihood.

Chapter 18

Markov Blankets

This chapter is based on the Wikipedia article, Ref.[14]. Markov blankets and Markov boundaries of bnets were apparently invented by Judea Pearl. His 1988 book Ref.[15], instead of a research paper, is usually given as the original reference.

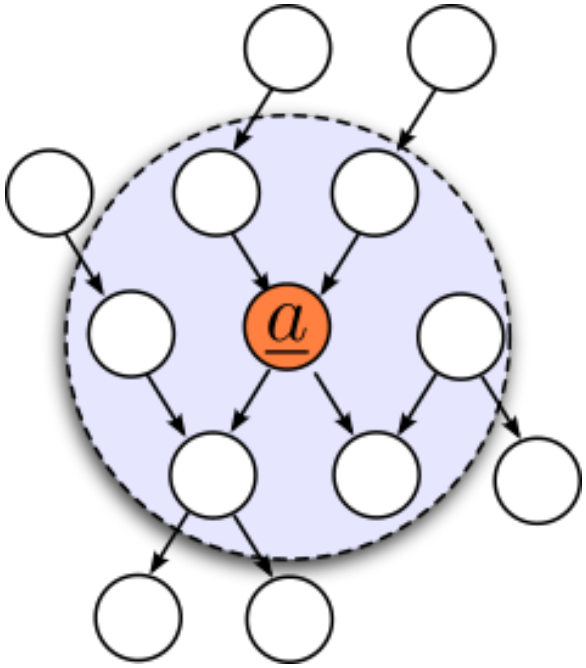


Figure 18.1: In a bnet, the minimal Markov blanket, aka Markov boundary, of node $\underline{a}$.

We will treat vectors of random variables as if they were sets when using the $\in$, $\subset$ and $-$ operations. For example, if $\underline{x} = (\underline{x}_0, \underline{x}_1, \underline{x}_2, \underline{x}_3)$ and $\underline{b} = (\underline{x}_1, \underline{x}_2)$, then $\underline{x}_1 \in \underline{b} \subset \underline{x}$ and $\underline{x} - \underline{b} = (\underline{x}_0, \underline{x}_3)$.

Below, $H(\underline{a} : \underline{b}|\underline{c})$ denotes the conditional mutual information of random variables $\underline{a}$ and $\underline{b}$ conditioned on random variable $\underline{c}$. $H(\underline{a} : \underline{b}|\underline{c})$ is used in Shannon Information Theory, where it is

defined by

$$H(\underline{a} : \underline{b}|\underline{c}) = \sum_{a,b,c} P(a,b,c) \ln \frac{P(a,b|c)}{P(a|c)P(b|c)} \ . \tag{18.1}$$

$H(\underline{a} : \underline{b}|\underline{c}) = 0$ iff $\underline{a}$ and $\underline{b}$ are independent (uncorrelated) when $\underline{c}$ is held fixed.

Suppose $\underline{a} \in \underline{X}$, $\underline{B} \subset \underline{X}$, but $\underline{a} \notin \underline{B}$. Then $\underline{B}$ is a Markov blanket of $\underline{a}$ if

$$H(\underline{a} : \underline{X} - \underline{a}|\underline{B}) = 0 \ . \tag{18.2}$$

In other words, one may assume that $\underline{a}$ depends on $\underline{B}$ only, and is independent of all random variables in $\underline{X} - (\underline{a} \cup \underline{B})$.

The minimal Markov blanket is called the Markov boundary.

In a bnet, the Markov boundary of a node $\underline{a}$, contains:

1. the parents of $\underline{a}$,
2. the children of $\underline{a}$,
3. the parents, other than $\underline{a}$, of the children of $\underline{a}$.

This is illustrated in Fig.18.1.

Chapter 19

Markov Chains

A Markov Chain is simply a bnet with the graph structure of a chain. For example, Fig.19.1 shows a chain with $n = 4$ nodes.

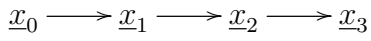


Figure 19.1: Markov chain with $n = 4$ nodes.

Because of its graph structure, the TPM of each node only depends on the state of the previous node:

$$P(x_t|(x_a)_{a\neq t}) = P(x_t|x_{t-1}) \ , \tag{19.1}$$

where $(x_a)_{a\neq t}$ are all the nodes except x_t itself and $t = 1, 2, \ldots, n-1$.

If there exists a single TPM $P_{\underline{x}_1|\underline{x}_0}$ such that

$$P(x_t|x_{t-1}) = P_{\underline{x}_1|\underline{x}_0}(x_t|x_{t-1}) \tag{19.2}$$

for $t = 1, 2, \ldots, n-1$, then we say that the Markov chain is **time homogeneous**.

Chapter 20

Markov Chain Monte Carlo (MCMC)

Monte Carlo methods are methods for using random number generation to sample probability distributions. The subject of Monte Carlo methods has many branches, as you can see from its Wikipedia category list, Ref.[16]. MCMC (Markov Chain Monte Carlo) is just one of those branches, albeit a major one. Metropolis-Hastings (MH) sampling is a very important MCMC method. Gibbs sampling is a special case of MH sampling. This chapter covers both, MH and Gibbs sampling. It also covers a few other types of sampling.

Throughout this chapter, we use $P_{\underline{x}} : S_{\underline{x}} \to [0, 1]$ to denote the target probability distribution that we wish to obtain samples from.

Inverse Cumulative Sampling

For more info about this topic and some original references, see Ref.[17].

This is one of the simplest methods for obtaining samples from a probability distribution $P_{\underline{x}}$, but it requires knowledge of the inverse cumulative distribution of $P_{\underline{x}}$, which is often not available.

The **cumulative distribution** function is defined by:

$$CUM_{\underline{x}}(x) = P(\underline{x} < x) = \int_{x' < x} dx' \; P_{\underline{x}}(x') \; . \tag{20.1}$$

Note that

$$P_{\underline{x}}(x) = \frac{d}{dx} CUM_{\underline{x}}(x) \; . \tag{20.2}$$

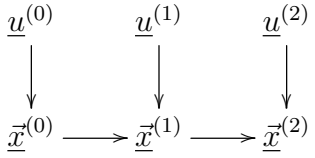


Figure 20.1: bnet for Inverse Cumulative Sampling

For $t = 0, 1, \ldots, T-1$, let

$\underline{u}^{(t)} \in [0,1]$= random variable, uniformly distributed over $[0,1]$.

$\vec{\underline{x}}^{(t)} = (\underline{x}^{(t)}[s])_{s=0,1,\ldots,nsam(t)-1}$ where $\underline{x}^{(t)}[s] \in S_{\underline{x}}$ for all s. Vector of samples collected up to time t.

The TPMs, printed in blue, for the nodes of bnet Fig.20.1, are:

$$P(u^{(t)}) = 1 \tag{20.3}$$

$$P(\vec{x}^{(t)} | \vec{x}^{(t-1)}, u^{(t)}) = \delta(\quad \vec{x}^{(t)}, [\vec{x}^{(t-1)}, CUM_{\underline{x}}^{-1}(u^{(t)})] \quad) \tag{20.4}$$

Motivation

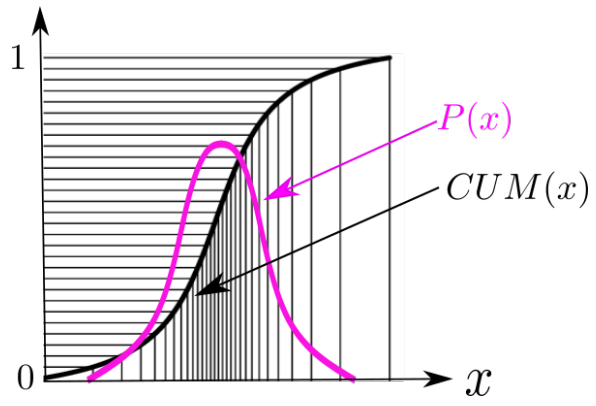


Figure 20.2: Motivation for Inverse Cumulative Sampling.

See Fig.20.2.

Note that if $\underline{u}$ is uniformly distributed over the interval $[0,1]$ and $a \in [0,1]$, then

$$P(\underline{u} < a) = a \ . \tag{20.5}$$

Thus

$$\begin{aligned} P(CUM_{\underline{x}}^{-1}(\underline{u}) < x) &= P(\underline{u} < CUM_{\underline{x}}(x)) & (20.6) \\ &= CUM_{\underline{x}}(x) \ . & (20.7) \end{aligned}$$

Therefore,

$$dP(CUM_{\underline{x}}^{-1}(\underline{u}) < x) = P_{\underline{x}}(x)dx \ . \tag{20.8}$$

Rejection Sampling

For more info about this topic and some original references, see Ref.[18].

This method samples from a "candidates" probability distribution $P_{\underline{c}} : S_{\underline{x}} \to [0,1]$, in cases where sampling directly from the target probability distribution $P_{\underline{x}} : S_{\underline{x}} \to [0,1]$ is not possible.

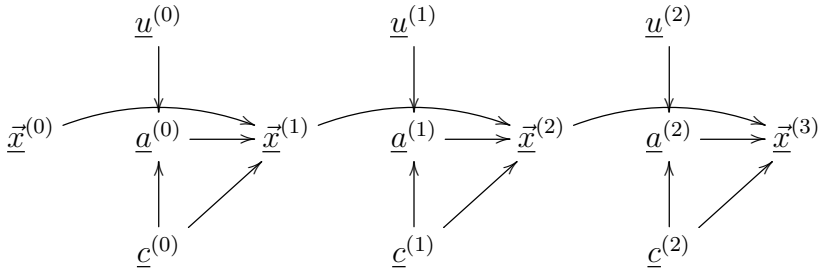


Figure 20.3: bnet for Rejection Sampling

For $t = 0, 1, \ldots, T-1$, let

$\underline{u}^{(t)} \in [0,1]$= random variable, uniformly distributed over $[0,1]$.

$\underline{a}^{(t)} \in \{0,1\}$= accept candidate? (no=0, yes=1)

$\underline{c}^{(t)} \in S_{\underline{x}}$= sample that is a candidate for being accepted

$\vec{\underline{x}}^{(t)} = (\underline{x}^{(t)}[s])_{s=0,1,\ldots,nsam(t)-1}$ where $\underline{x}^{(t)}[s] \in S_{\underline{x}}$ for all s. Vector of samples collected up to time t.

This algorithm requires a priori definition of a candidate probability distribution $P_{\underline{c}} : S_{\underline{x}} \to \mathbb{R}$ such that

$$P_{\underline{x}}(x) < \beta P_{\underline{c}}(x) \tag{20.9}$$

for all $x \in S_{\underline{x}}$, for some $\beta \in \mathbb{R}$.

The TPMs, printed in blue, for the nodes of bnet Fig.20.3, are:

$$P(u^{(t)} = u) = 1 \tag{20.10}$$

$$P(\underline{c}^{(t)} = c) = P_{\underline{c}}(c) \tag{20.11}$$

$$P(\underline{a}^{(t)} = a | \underline{c}^{(t)} = c, \underline{u}^{(t)} = u) = \begin{cases} \delta(a,0) & \text{if } u\beta P_{\underline{c}}(c) \geq P_{\underline{x}}(c) \\ \delta(a,1) & \text{if } u\beta P_{\underline{c}}(c) < P_{\underline{x}}(c) \end{cases} \tag{20.12}$$

$$P(\vec{x}^{(t)} | \vec{x}^{(t-1)}, \underline{a}^{(t)} = a, \underline{c}^{(t)} = c) = \begin{cases} \delta(\vec{x}^{(t)}, \vec{x}^{(t-1)}) & \text{if } a = 0 \\ \delta(\vec{x}^{(t)}, [\vec{x}^{(t-1)}, c]) & \text{if } a = 1 \end{cases} \tag{20.13}$$

This last equation is only defined for $t > 0$. For $t = 0$, the left hand side reduces to $P(\vec{x}^{(0)})$ which must be specified a priori.

Motivation

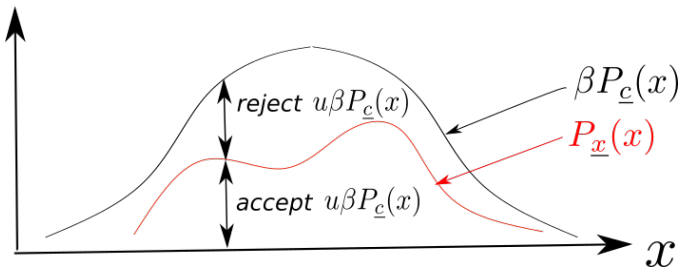


Figure 20.4: Motivation for Rejection Sampling.

See Fig.20.4.

Metropolis-Hastings Sampling

For more info about this topic and some original references, see Refs.[19] and [20].

An advantage of this method is that it can sample unnormalized probability distributions $(constant)P_{\underline{x}}$ because it only uses ratios of $P_{\underline{x}}$ at two different points. Another advantage of this method is that it scales much better than other sampling methods as the number of dimensions of the sampled variable $\underline{x}$ increases.

This method produces samples that take a finite amount of time to reach steady state. The samples are also theoretically correlated instead of being i.i.d. as one desires. To mitigate for the steady state problem, one discards an initial set of samples (the "burn-in" period). To mitigate for the correlation problem, one calculates the autocorrelation between the samples and keeps only samples separated by a time interval after which the samples cease to be autocorrelated to a good approximation.

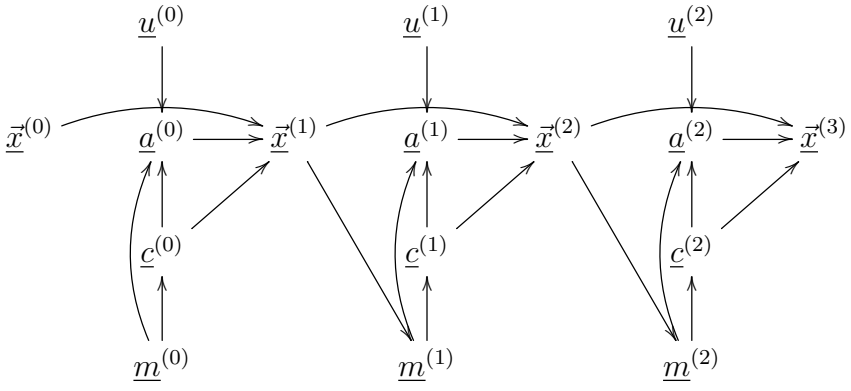


Figure 20.5: bnet for Metropolis-Hastings Sampling

For $t = 0, 1, \ldots, T-1$, let
$\underline{u}^{(t)} \in [0,1]$= random variable, uniformly distributed over $[0,1]$.
$\underline{a}^{(t)} \in \{0,1\}$= accept candidate? (no=0, yes=1)
$\underline{c}^{(t)} \in S_{\underline{x}}$= sample that is a candidate for being accepted
$\underline{m}^{(t)} \in S_{\underline{x}}$= memory of last accepted sample
$\underline{\vec{x}}^{(t)} = (\underline{x}^{(t)}[s])_{s=0,1,\ldots,nsam(t)-1}$ where $\underline{x}^{(t)}[s] \in S_{\underline{x}}$ for all s. Vector of samples collected up to time t.

A **proposal TPM** $P_{\underline{c}|\underline{x}} : S_{\underline{x}}^2 \to [0,1]$ must be specified a priori for this algorithm.
The TPMs, printed in blue, for the nodes of bnet Fig.20.5, are:

$$P(u^{(t)} = u) = 1 \tag{20.14}$$

$$P(\underline{c}^{(t)} = c | \underline{m}^{(t)} = m) = P_{\underline{c}|\underline{x}}(c|m) \tag{20.15}$$

$$P(\underline{a}^{(t)} = a | \underline{c}^{(t)} = c, \underline{u}^{(t)} = u, \underline{m}^{(t)} = m) = \begin{cases} \delta(a,0) & \text{if } u \geq \alpha(c|m) \\ \delta(a,1) & \text{if } u < \alpha(c|m) \end{cases} \tag{20.16}$$

where the **acceptance probability** α is defined as

$$\alpha(c|m) = \min\left(1, \frac{P_{\underline{c}|\underline{x}}(m|c) P_{\underline{x}}(c)}{P_{\underline{c}|\underline{x}}(c|m) P_{\underline{x}}(m)}\right) . \tag{20.17}$$

Note that if the proposal distribution is symmetric, then

$$\alpha(c|m) = \min\left(1, \frac{P_{\underline{x}}(c)}{P_{\underline{x}}(m)}\right) . \tag{20.18}$$

$$P(\underline{\vec{x}}^{(t)} | \underline{\vec{x}}^{(t-1)}, \underline{a}^{(t)} = a, \underline{c}^{(t)} = c) = \begin{cases} \delta(\vec{x}^{(t)}, \vec{x}^{(t-1)}) & \text{if } a = 0 \\ \delta(\vec{x}^{(t)}, [\vec{x}^{(t-1)}, c]) & \text{if } a = 1 \end{cases} \tag{20.19}$$

This last equation is only defined for $t > 0$. For $t = 0$, the left hand side reduces to $P(\vec{x}^{(0)})$ which must be specified a priori.

$$P(\underline{m}^{(t)} = m | \underline{\vec{x}}^{(t)}) = \delta(m, \text{last component of } \vec{x}^{(t)}) . \tag{20.20}$$

This last equation is only defined for $t > 0$. For $t = 0$, the left hand side reduces to $P(\underline{m}^{(0)} = m)$ which must be specified a priori.

Motivation

See Fig.20.6.

Consider a time homogeneous (its TPM is the same for all times) Markov chain with TPM $P(x'|x) = [T]_{x',x}$. Its **stationary distribution**, if it exists, is defined as

$$\pi = \lim_{n \to \infty} T^n \pi_0 . \tag{20.21}$$

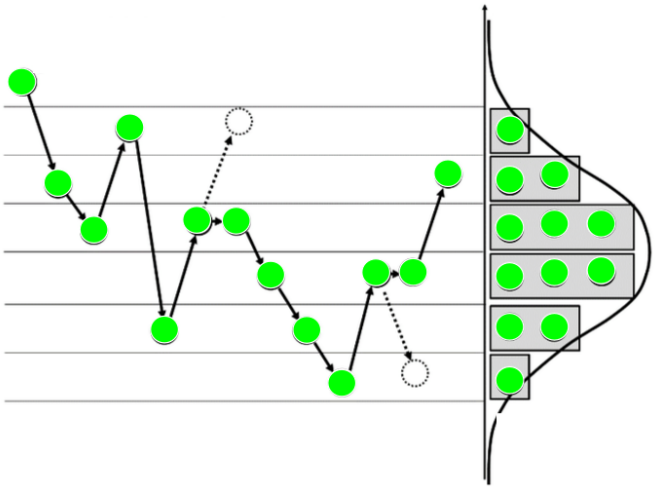


Figure 20.6: Motivation for Metropolis-Hastings Sampling.

Suppose the prob distribution $P_{\underline{x}}(x)$ that we wish to sample from satisfies

$$P_{\underline{x}}(x) = \pi(x) \ . \tag{20.22}$$

Reversibility (detailed balance): For all $x, x' \in S_{\underline{x}}$,

$$P(x'|x)\pi(x) = P(x|x')\pi(x') \ . \tag{20.23}$$

Detailed balance is a sufficient (although not necessary) condition for a unique stationary prob distribution π to exist.[1]

Let

$$P(x'|x) = P(\underline{a} = 1|x', x)P_{\underline{c}|\underline{x}}(x'|x) + \delta(x, x')P(\underline{a} = 0|x) \ , \tag{20.24}$$

where

$$P(\underline{a} = 0|x) = \sum_{x'} P(\underline{a} = 0|x', x)P_{\underline{c}|\underline{x}}(x'|x) \ . \tag{20.25}$$

Claim 4 *If*

$$P(\underline{a} = 1|x', x) = \alpha(x'|x) \ , \tag{20.26}$$

then detailed balance is satisfied.

proof: Assume $x \neq x'$.

[1] As explained lucidly in Ref.[19], besides detailed balance, 2 other properties must also be satisfied by the Markov chain, irreducibiity and aperiodicity. However, because of how it is constructed, the Metropolis-Hastings algorithm automatically produces a Markov chain that has those 2 properties.

$$
\begin{aligned}
P(x'|x)P(x) &= P(\underline{a}=1|x',x)P_{\underline{c}|\underline{x}}(x'|x)P_{\underline{x}}(x) && (20.27)\\
&= \min\left(1, \frac{P_{\underline{c}|\underline{x}}(x|x')P_{\underline{x}}(x')}{P_{\underline{c}|\underline{x}}(x'|x)P_{\underline{x}}(x)}\right) P_{\underline{c}|\underline{x}}(x'|x)P_{\underline{x}}(x) && (20.28)\\
&= \min\left(P_{\underline{c}|\underline{x}}(x'|x)P_{\underline{x}}(x), P_{\underline{c}|\underline{x}}(x|x')P_{\underline{x}}(x')\right) && (20.29)\\
&= P(x|x')P(x') && (20.30)
\end{aligned}
$$

QED

Gibbs Sampling

For more info about this topic and some original references, see Ref.[21].

Gibbs sampling is a special case of Metropolis-Hastings sampling. Gibbs sampling is ideally suited for application to a bnet, because it is stated in terms of the conditional prob distributions of N random variables, and conditional prob distributions are part of the definition of a bnet.

Consider a bnet with nodes $\underline{x}_0, \underline{x}_1, \ldots, \underline{x}_{N-1}$

Identify the random variable $\underline{x} = (\underline{x}_0, \underline{x}_1, \ldots, \underline{x}_{N-1})$ with the random variable $\underline{x}$ used in Metropolis-Hastings sampling. For Gibbs sampling, we use the following proposal distribution:

$$
P_{\underline{c}|\underline{x}}(c|m) = \prod_{j=0}^{N-1} P(c_j \mid [m_i]_{i\neq j}) \; . \tag{20.31}
$$

Eq.(20.31) can be simplified using Markov Blankets (see Chapter 18) to the following:

$$
P_{\underline{c}|\underline{x}}(c|m) = \prod_{j=0}^{N-1} P(c_j \mid [m_i : \forall i \ni \underline{x}_i \in MB(\underline{x}_j)]) \; , \tag{20.32}
$$

where, for any node $\underline{a}$, we denote its Markov blanket by $MB(\underline{a})$.

An alternative proposal distribution that leads to much faster convergence is as follows. The idea is to make the components $c_j^{(t)}$ of candidate sample $c^{(t)}$ depend on the previous components $(c_i^{(t)})_{i<j}$. See the bnet Fig.20.7. The TPM for the nodes of that bnet are

$$
P(\underline{c}_j^{(t)} = c_j \mid (\underline{c}_i^{(t)})_{i<j} = (c_i)_{i<j}, \underline{m}^{(t-1)} = m) = P(c_j|(c_i)_{i<j}, (m_i)_{i>j}) \tag{20.33}
$$

for $j = 0, 1, \ldots, N-1$. This implies

$$
P_{\underline{c}|\underline{x}}(\underline{c}^{(t)} = c|\underline{m}^{(t-1)} = m) = \prod_{j=0}^{N-1} P(c_j|(c_i)_{i<j}, (m_i)_{i>j}) \; . \tag{20.34}
$$

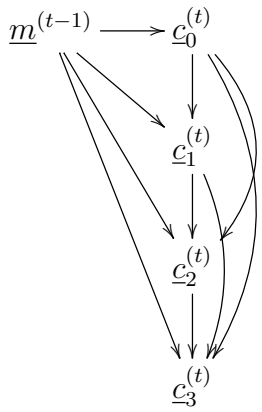


Figure 20.7: In Gibbs sampling, the proposal distribution $P_{\underline{c}|\underline{x}}$ can be defined by making the components $c_j^{(t)}$ of candidate sample $c^{(t)}$ depend on the previous components $(c_i^{(t)})_{i<j}$.

As before, we can condition only on the Markov blanket of each node $\underline{x}_j$.

$$P_{\underline{c}|\underline{x}}(\underline{c}^{(t)} = c|\underline{m}^{(t-1)} = m) = \prod_{j=0}^{N-1} P(c_j|(c_i)_{i<j}, (m_i)_{i>j}, \text{ use only } c_i \text{ and } m_i \ni \underline{x}_i \in MB(\underline{x}_j)) \ . \tag{20.35}$$

Importance Sampling

For more info about this topic and some original references, see Ref.[22].

Suppose random variables $\underline{x}[s] \in S_{\underline{x}}$ for $s = 0, 1, \ldots, nsam - 1$ are i.i.d. with probability distribution $P_{\underline{x}}$. Then

$$E_{\underline{x}}[f(x)] \approx \frac{1}{nsam} \sum_{s=0}^{nsam-1} f(x[s]) \tag{20.36}$$

for any $f : S_{\underline{x}} \to \mathbb{R}$. Sometimes, instead of using i.i.d. samples $\underline{x}[s] \in S_{\underline{x}}$ where $\underline{x}[s] \sim P_{\underline{x}}$, we wish to use i.i.d. samples $\underline{y}[s] \in S_{\underline{x}}$ where $\underline{y}[s] \sim P_{\underline{y}}$.

$$\begin{aligned} E_{\underline{x}}[f(\underline{x})] &= \sum_x P_{\underline{x}}(x) f(x) && (20.37) \\ &= \sum_x P_{\underline{y}}(x)) \frac{P_{\underline{x}}(x)}{P_{\underline{y}}(x)} f(x) && (20.38) \\ &= E_{\underline{y}}[\frac{P_{\underline{x}}(y)}{P_{\underline{y}}(y)} f(y)] && (20.39) \end{aligned}$$

Sampling from $P_{\underline{y}}(y)$ instead of $P_{\underline{x}}(x)$ might reduce (or increase) variance for a particular $f : S_{\underline{x}} \to \mathbb{R}$.

$$Var_{\underline{x}}[f(x)] = E_{\underline{x}}[(f(x))^2] - (E_{\underline{x}}[f(x)])^2 \tag{20.40}$$

$$\begin{aligned} Var_{\underline{y}}[\frac{P_{\underline{x}}(y)}{P_{\underline{y}}(y)} f(y)] &= E_{\underline{y}}[(\frac{P_{\underline{x}}(y)}{P_{\underline{y}}(y)} f(y))^2] - (E_{\underline{y}}[\frac{P_{\underline{x}}(y)}{P_{\underline{y}}(y)} f(y)])^2 & (20.41) \\ &= E_{\underline{x}}[\frac{P_{\underline{x}}(x)}{P_{\underline{y}}(x)} (f(x))^2] - (E_{\underline{x}}[f(x)])^2 & (20.42) \end{aligned}$$

Chapter 21

Message Passing (Belief Propagation)

Message Passing (aka Belief Propagation) was first proposed in 1982 Ref.[23] by Judea Pearl to simplify the exact evaluation of probability marginals of bnets (exact inference). It gives exact results for trees and polytrees (i.e. bnets with a single connected component and no acyclic loops). For bnets with loops, it gives approximate results (loopy belief propagation), and it has been generalized to the junction tree algorithm (see Chapter 15) which can do exact inference for general bnets with loops. The basic idea behind the junction tree algorithm is to eliminate loops by clustering them into single nodes.

For more information about Belief Propagation, see Ref.[24].

Pearl MP for arbitrary bnets

Consider Fig.21.1, which illustrates a bnet node $\underline{x}$ receiving and sending messages to its neighbors.

Note that in Fig.21.1, a function is labeled λ (a likelihood function) if the message direction, and the graph arrow, point in opposite directions (i.e, message "swimming upstream"), whereas a function is labeled π (a probability function) if they point in the same direction (i.e, message "swimming downstream"). In the symbols $\overrightarrow{|}$ and $\overleftarrow{|}$, the arrow indicates the direction of the message, whereas the vertical line indicates the direction of the graph arrow. For example, in $\lambda_{\underline{b}\,\overrightarrow{|}\,\underline{x}}(x)$ the message goes from $\underline{b}$ to $\underline{x}$ and the graph arrow points in the opposite direction. In $\pi_{\underline{b}\,\overleftarrow{|}\,\underline{x}}(x)$ the message goes from $\underline{x}$ to $\underline{b}$ and the graph arrow points in the same direction. The argument of the π and λ functions is always the same as the letter in the subscript that is closest to argument; i.e., the letter in the subscript that is to the right of the vertical bar.

Note that in Fig.21.1, we indicate messages that travel "downstream" (resp., "upstream"), by arrows with dashed (resp., dotted) lines as shafts. Mnemonic: think of the shaft as a velocity vector field for the message. You travel faster when you swim downstream as opposed to upstream.

$pa(\underline{x})$ = parents of node $\underline{x}$

$ch(\underline{x})$ = children of node $\underline{x}$

$nb(\underline{x}) = pa(\underline{x}) \cup ch(\underline{x})$ = neighbors of node $\underline{x}$

$\underline{x}$'s incoming messages (one comes from each parent and child of $\underline{x}$)

- $[\lambda_{\underline{b}\,\overrightarrow{|}\,\underline{x}}(\cdot)]_{\underline{b}\in ch(\underline{x})}$

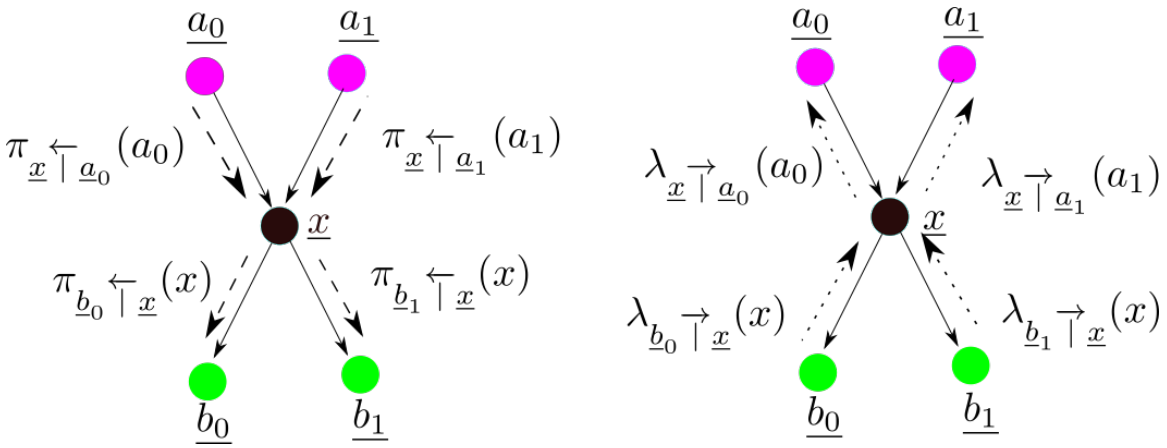


Figure 21.1: Node $\underline{x}$ receiving and sending messages to its neighbors. (neighbors= parents and children).

- $[\pi_{\underline{x}\,|\,\underline{a}}^{\leftarrow}(\cdot)]_{\underline{a}\in pa(\underline{x})}$

$$IN_{\underline{x}} = ([\lambda_{\underline{b}\,|\,\underline{x}}^{\rightarrow}(\cdot)]_{\underline{b}\in ch(\underline{x})}, [\pi_{\underline{x}\,|\,\underline{a}}^{\leftarrow}(\cdot)]_{\underline{a}\in pa(\underline{x})}) \tag{21.1}$$

$$= [IN_{\underline{x},\underline{n}}]_{\underline{n}\in nb(\underline{x})} \tag{21.2}$$

$\underline{x}$'s outgoing messages (one goes to each parent and child of $\underline{x}$)

- $[\pi_{\underline{b}\,|\,\underline{x}}^{\leftarrow}(\cdot)]_{\underline{b}\in ch(\underline{x})}$
- $[\lambda_{\underline{x}\,|\,\underline{a}}^{\rightarrow}(\cdot)]_{\underline{a}\in pa(\underline{x})}$

$$OUT_{\underline{x}} = ([\pi_{\underline{b}\,|\,\underline{x}}^{\leftarrow}(\cdot)]_{\underline{b}\in ch(\underline{x})}, [\lambda_{\underline{x}\,|\,\underline{a}}^{\rightarrow}(\cdot)]_{\underline{a}\in pa(\underline{x})}) \tag{21.3}$$

$$= [OUT_{\underline{x},\underline{n}}]_{\underline{n}\in nb(\underline{x})} \tag{21.4}$$

$\underline{x}$'s memory

$$\mathcal{M}_{\underline{x}} = (IN_{\underline{x}}, OUT_{\underline{x}}) \tag{21.5}$$

For times $t = 0, 1, \ldots, T-1$, we calculate $\mathcal{M}_{\underline{x}}^{(t)}$ in two steps: first we calculate $IN_{\underline{x}}^{(t)}$, then we calculate $OUT_{\underline{x}}^{(t)}$:

1. **Calculating $IN_{\underline{x}}^{(t)}$ from signals received from $\underline{n} \in R \subset nb(\underline{x})$:**

   For all $\underline{n} \in nb(\underline{x})$,

$$IN_{\underline{x},\underline{n}}^{(t)} = \begin{cases} IN_{\underline{x},\underline{n}}^{(t-1)} \text{ if } \underline{n} \notin R \\ OUT_{\underline{n},\underline{x}}^{(t-1)} \text{ if } \underline{n} \in R \end{cases} \tag{21.6}$$

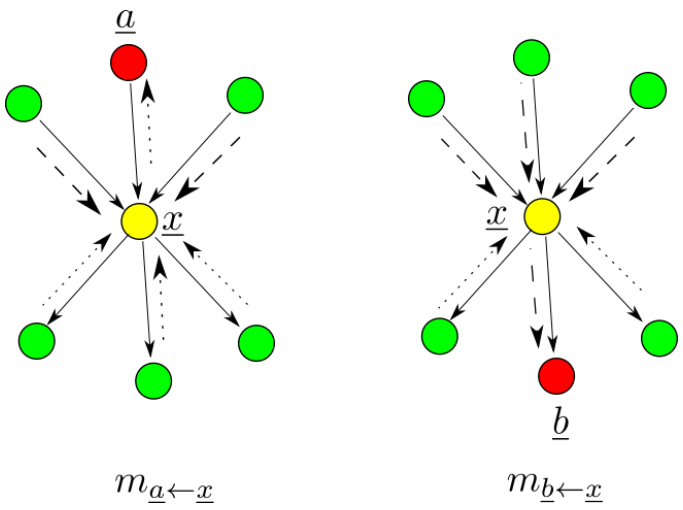


Figure 21.2: The yellow node is a gossip monger. It receives messages from all the green nodes, and then it relays a joint message to the red node. Union of green nodes and the red node = full neighborhood of yellow node. There are two possible cases: the red node is either a parent or a child of the yellow one. As usual, we use arrows with dashed (resp., dotted) shafts for downstream (resp., upstream) messages.

No instantaneous replies (message backflow): If $\underline{x}$ receives signals from $R \subset nb(\underline{x})$ and sends them to $S \subset nb(\underline{x})$, then S and R are disjoint and $R \cup S = nb(\underline{x})$.

2. **Calculating $OUT_{\underline{x}}^{(t)}$ from already calculated $IN_{\underline{x}}^{(t)}$:**

   Let $\underline{a}^{na} = (\underline{a}_i)_{i=0,1,\ldots,na-1}$ denote the parents of $\underline{x}$ and $\underline{b}^{nb} = (\underline{b}_i)_{i=0,1,\ldots,nb-1}$ its children.

   Define

$$\pi_{\underline{x}\,\uparrow}(x) = \sum_{a^{na}} P(x|a^{na}) \prod_i \pi_{\underline{x}\,\uparrow\,\underline{a}_i}(a_i) \tag{21.7}$$

$$= E_{\underline{a}^{na}}[P(x|a^{na})] \tag{21.8}$$

   and

$$\lambda_{\downarrow\,\underline{x}}(x) = \prod_i \lambda_{\underline{b}_i\,\downarrow\,\underline{x}}(x)\ . \tag{21.9}$$

   From the $m_{\underline{a}\leftarrow\underline{x}}$ panel of Fig.21.2, we get[1]

[1] As usual in this book, $\mathcal{N}(!x)$ means a constant that is independent of x.

$$\underbrace{\lambda_{\underline{x}\,|\,\underline{a}_i}^{\rightarrow}(a_i)}_{OUT} = \mathcal{N}(!a_i)\sum_x \left[\underbrace{\lambda_{|\,\underline{x}}^{\rightarrow}(x)}_{IN}\sum_{(a_k)_{k\neq i}}\left(P(x|a^{na})\prod_{k\neq i}\underbrace{\pi_{\underline{x}\,|\,\underline{a}_k}^{\leftarrow}(a_k)}_{IN}\right)\right] \tag{21.10}$$

$$= \mathcal{N}(!a_i)\sum_x \left[\lambda_{|\,\underline{x}}^{\rightarrow}(x)E_{(\underline{a}_k)_{k\neq i}}[P(x|a^{na})]\right] \tag{21.11}$$

$$= \mathcal{N}(!a_i)E_{(\underline{a}_k)_{k\neq i}}E_{\underline{x}|a^{na}}\lambda_{|\,\underline{x}}^{\rightarrow}(x) \tag{21.12}$$

From the $m_{\underline{b}\leftarrow\underline{x}}$ panel of Fig.21.2, we get

$$\underbrace{\pi_{\underline{b}_j\,|\,\underline{x}}^{\leftarrow}(x)}_{OUT} = \mathcal{N}(!x)\,\underbrace{\pi_{\underline{x}\,|}^{\leftarrow}(x)}_{IN}\prod_{i\neq j}\underbrace{\lambda_{\underline{b}_i\,|\,\underline{x}}^{\rightarrow}(x)}_{IN} \tag{21.13}$$

Claim: Define

$$BEL^{(t)}(x) = \mathcal{N}(!x)\lambda_{|\,\underline{x}}^{(t)\,\rightarrow}(x)\pi_{\underline{x}\,|}^{(t)\,\leftarrow}(x)\ . \tag{21.14}$$

Then

$$\lim_{t\to\infty} BEL^{(t)}(x) = P(x)\ . \tag{21.15}$$

This says that the belief in x equals the product of messages received from all parents and children.

Example of Pearl MP

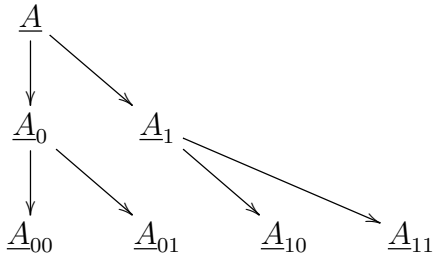


Figure 21.3: bnet for a full binary tree with 3 levels. $\underline{A}$ is the apex root node.

Fig.21.4 is the **messages-bnet** for the **source bnet** Fig.21.3. The node TPMs, printed in blue, for the bnet Fig.21.4 are as follows.

The following equation applies with $t = 1$, and $\underline{x} \in \{\underline{A}_0, \underline{A}_1\}$. It also applies with $t = 2$ and $\underline{x} \in \{\underline{A}_{00}, \underline{A}_{01}, \underline{A}_{10}, \underline{A}_{11}\}$.

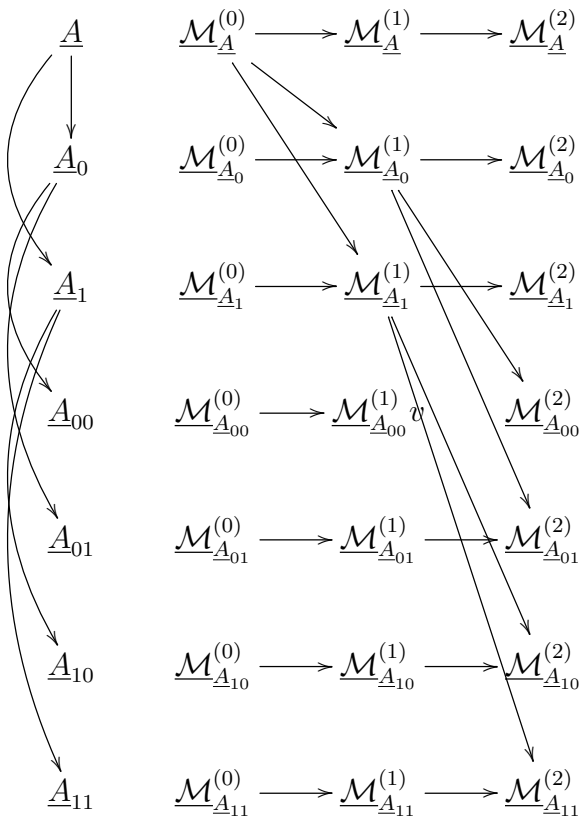


Figure 21.4: messages-bnet for the source-bnet Fig.21.3.

$$P(\mathcal{M}^{(t)}_{\underline{x}} \mid [\mathcal{M}^{(t-1)}_{\underline{a}}]_{\underline{a}\in pa(\underline{x})\cup\underline{x}}) \quad = \quad \delta(\mathcal{M}^{(t)}_{\underline{x}}, \mathcal{M}^{(t)}_{\underline{x}}([\mathcal{M}^{(t-1)}_{\underline{a}}]_{\underline{a}\in pa(\underline{x})\cup\underline{x}})) \tag{21.16}$$

Whenever the only arrow entering $\mathcal{M}^{(t)}_{\underline{x}}$ is from $\mathcal{M}^{(t-1)}_{\underline{x}}$, there is no change in $\mathcal{M}^{(t)}_{\underline{x}}$:

$$P(\mathcal{M}^{(t)}_{\underline{x}} \mid \mathcal{M}^{(t-1)}_{\underline{x}}) = \delta(\mathcal{M}^{(t)}_{\underline{x}}, \mathcal{M}^{(t-1)}_{\underline{x}}) \; . \tag{21.17}$$

As for the root nodes of the messages-bnet at time $t = 0$, they should encode the prior prob distributions of the source bnet Fig.21.3. In this example, the only root node of the source bnet is $\underline{A}$. Thus, let

$$IN^{(0)}_{\underline{A}} = 0, \;\; OUT^{(0)}_{\underline{A}} = P_{\underline{A}}(\cdot), \;\; \mathcal{M}^{(0)}_{\underline{A}} = (IN^{(0)}_{\underline{A}}, OUT^{(0)}_{\underline{A}}) \tag{21.18}$$

$$P(\mathcal{M}^{(0)}_{\underline{A}}) = \delta(\mathcal{M}^{(0)}_{\underline{A}}, \mathcal{M}^{(0)}_{\underline{A}}(\ldots)) \; . \tag{21.19}$$

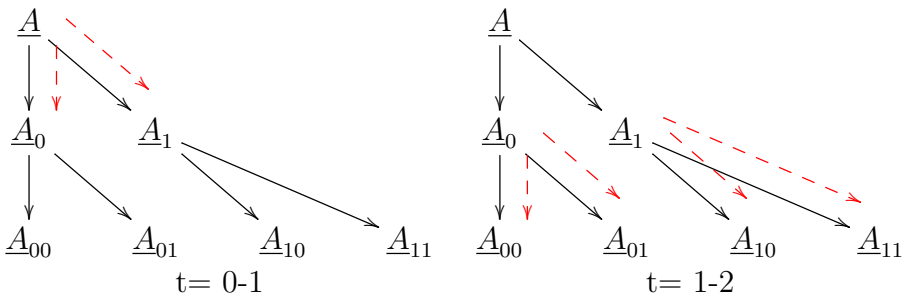


Figure 21.5: The messages passed in Fig.21.4 are indicated by red arrows on the source bnet Fig.21.3. As usual, we use arrows with dashed (resp., dotted) shafts for downstream (resp., upstream) messages. In this diagram, there are only dashed arrows.

If the state of a node is known a priori (i.e., there is a prior "evidence" for that node), then, in the MP procedure, replace that node's TPM by a Kronecker delta function. $BEL(x)$ should now converge to $P(x|$ a priori evidence for all nodes).

In Fig.21.5, the messages passed in the messages-bnet are indicated by red arrows on the source bnet.

Bipartite bnets

By a **bipartite bnet** we will mean a bnet all of whose nodes are root nodes (parentless) or leaf nodes (childless). Pearl MP simplifies when dealing with bipartite bnets. In this section, we will explain how it simplifies. But before doing so, let us explain how the following two types of diagrams can be replaced by equivalent bipartite bnets:

- Factor Graphs
- Tree bnets

Consider a product $g = \prod_\alpha f_\alpha$ of scalar functions $f_\alpha : \mathbb{R}^{nx(\alpha)} \to \mathbb{R}$ for $\alpha = 0, 1, \ldots nf-1$. For instance, consider $g : \mathbb{R}^3 \to \mathbb{R}$ defined by:

$$g(x_0, x_1, x_2) = f_0(x_0) f_1(x_0, x_1) f_2(x_0, x_1) f_3(x_1, x_2) \; . \tag{21.20}$$

The **factor graph** for this function g is given by Fig.21.6.

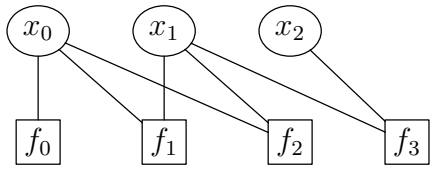


Figure 21.6: Factor graph for function g defined by Eq.(21.20).

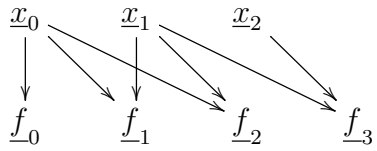


Figure 21.7: Bipartite bnet corresponding to factor graph Fig.21.6.

One can map any factor graph (the "source") to a special bipartite bnet (the "image"), as follows. Replace each x_i by $\underline{x}_i \in \mathbb{R}$ for $i = 0, 1, \ldots, nx-1$ and each f_α by $\underline{f}_\alpha \in \mathbb{R}$ for $\alpha = 0, 1, \ldots, nf-1$. Then replace the connections (edges) of the factor graph by arrows from $\underline{x}_i$ to $\underline{f}_\alpha$. For example, Fig.21.7 is the image of the factor graph Fig.21.6.

Let $\underline{x}^{nx} = (\underline{x}_0, \underline{x}_1, \ldots, \underline{x}_{nx-1})$, $\underline{f}^{nf} = (\underline{f}_0, \underline{f}_1, \ldots, \underline{f}_{nf-1})$, and $f_\alpha = f_\alpha(x_{nb(\underline{f}_\alpha)})$. Here we are using $nb(\underline{f}_\alpha)$ to denote the neighborhood of node $\underline{f}_\alpha$ in the image bipartite bnet, and we are using

x_S to denote $(x_i)_{i\in S}$. Then we define the node TPMs, printed in blue, for the image bipartite bnet to be

$$P(f_\alpha|x_{nx(\underline{f}_\alpha)}) = \mathcal{N}(!x_{nb(\underline{f}_\alpha)})f_\alpha(x_{nb(\underline{f}_\alpha)}) \tag{21.21}$$

for $\alpha = 0, 1, \ldots, nf-1$ and

$$P_{\underline{x}_i}(x'_i) = \delta(x'_i, x_i) \tag{21.22}$$

for $i = 0, 1, \ldots, nx-1$.

Note that

$$P(f^{nf}|x^{nx}) = \mathcal{N}(!x^{nx}) \prod_\alpha f_\alpha(x_{nb(\underline{f}_\alpha)}) \; . \tag{21.23}$$

A **tree bnet** is a bnet for which all nodes have exactly one parent except for the apex root node which has none. A tree bnet is very much like the filing system in a computer.

One can map a tree bnet (the "source") into an equivalent bipartite bnet (the "image") as follows. Replace each arrow

$$\underline{x} \longrightarrow \underline{y} \tag{21.24}$$

of the tree bnet by

$$\underline{x} \longrightarrow \underline{P_{y|x}} \longleftarrow \underline{y} \; . \tag{21.25}$$

For example, the tree bnet Fig.21.8 has the image bipartite bnet given by Fig.21.9. The bnet Fig.21.10 is just a different way of drawing the bnet Fig.21.9.

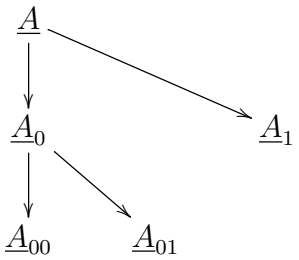


Figure 21.8: Example of a tree bnet.

The node TPMs, printed in blue, for the image bipartite bnet Fig.21.9 are as follows. We express the TPMs of the image bnet in terms of the TPMs of the source bnet Fig.21.8.

$$P(P_{\underline{y|x}}|x, y) = P_{\underline{y|x}}(y|x) \tag{21.26}$$

for all the leaf nodes $P_{\underline{y|x}}$ of the image bipartite bnet. Also

$$P_{\underline{y}}(y') = \delta(y', y) \tag{21.27}$$

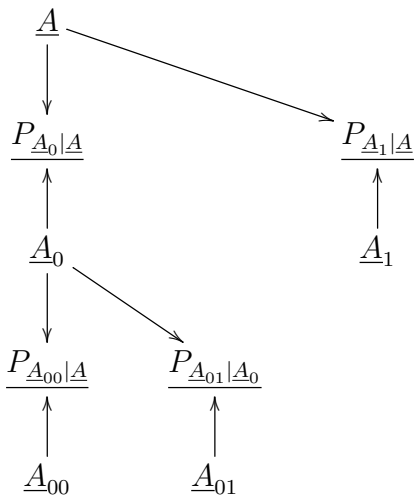


Figure 21.9: Bipartite bnet corresponding to tree bnet Fig.21.8.

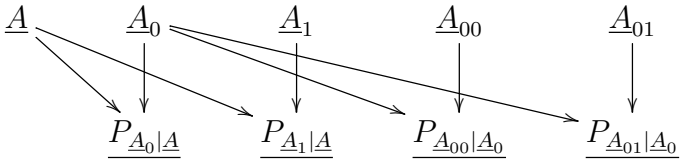


Figure 21.10: Different way of drawing the bnet Fig.21.9.

for all the root nodes $\underline{y}$ of the image bipartite bnet except when $\underline{y}$ corresponds to the root node $\underline{A}$ of the source tree bnet. In that exceptional case,

$$P_{\underline{y}}(y') = P_{\underline{A}}(y') \ . \tag{21.28}$$

MP for bipartite bnets

For a bipartite bnet as defined before, with root nodes $\underline{x}_i$ and leaf nodes $\underline{f}_\alpha$, let

$$nb(i) = \{\alpha : \underline{f}_\alpha \in nb(\underline{x}_i)\} \ , \tag{21.29}$$

$$nb(\alpha) = \{i : \underline{x}_i \in nb(\underline{f}_\alpha)\} \ , \tag{21.30}$$

$$m_{\alpha \leftarrow i}(x_i) = \pi_{\underline{f}_\alpha \overleftarrow{|} \underline{x}_i}(x_i) \ , \tag{21.31}$$

$$m_{i \leftarrow \alpha}(x_i) = \lambda_{\underline{f}_\alpha \setminus \underline{x}_i \rightarrow \underline{x}_i}(x_i) \ , \tag{21.32}$$

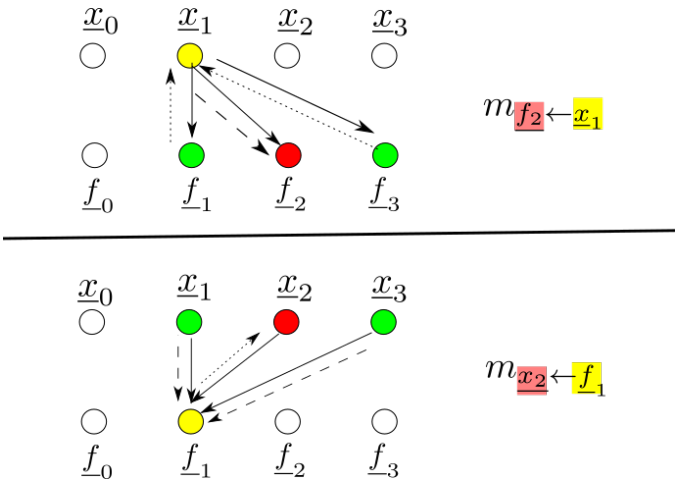


Figure 21.11: This figure is equivalent to Fig.21.2 for the special case of a bipartite bnet. Union of green nodes and the red node = full neighborhood of yellow node. There are two possible cases: the red node is either a parent or a child of the yellow node.

Next we will show how to find $m^{(t)}_{\alpha \leftarrow i}$ and $m^{(t)}_{i \leftarrow \alpha}$ from other messages at times t and $t-1$.

1. **We find $m^{(t)}_{\alpha \leftarrow i}$ as follows.**
   See the $m_{\underline{f}_2 \leftarrow \underline{x}_1}$ panel of Fig.21.11.
   For $i = 0, 1, \ldots, nx-1$, if $\alpha \in nb(i)$, then

$$m^{(t)}_{\alpha \leftarrow i}(x_i) = \prod_{\beta \in nb(i)-\alpha} m^{(t-1)}_{i \leftarrow \beta}(x_i) \ , \tag{21.33}$$

   whereas if $\alpha \notin nb(i)$

$$m^{(t)}_{\alpha \leftarrow i} = m^{(t-1)}_{\alpha \leftarrow i} \ . \tag{21.34}$$

2. **We find $m^{(t)}_{i \leftarrow \alpha}$ as follows.**
   See the $m_{\underline{x}_2 \leftarrow \underline{f}_1}$ panel of Fig.21.11.
   For $\alpha = 0, 1, \ldots, nf-1$, if $j \in nb(\alpha)$, then

$$m^{(t)}_{j \leftarrow \alpha}(x_j) = \sum_{(x_k)_{k \in nb(\alpha)-j}} f_\alpha(x_{nb(\alpha)}) \prod_{k \in nb(\alpha)-j} m^{(t-1)}_{\alpha \leftarrow k}(x_k) \tag{21.35}$$

$$= E^{(t-1)}_{(x_k)_{k \in nb(\alpha)-j}}[f_\alpha(x_{nb(\alpha)})] \ , \tag{21.36}$$

whereas if $j \notin nb(\alpha)$

$$m^{(t)}_{j\leftarrow\alpha}(x_j) = m^{(t-1)}_{j\leftarrow\alpha}(x_j) \ . \tag{21.37}$$

In the above equations for $m^{(t)}_{\alpha\leftarrow i}$ and $m^{(t)}_{i\leftarrow\alpha}$, if the range set of a product is empty, then define the product as 1; i.e., $\prod_{k\in\emptyset} F(k) = 1$.

Claim:

$$P(x_i) = \lim_{t\to\infty} \mathcal{N}(!x_i) \prod_{\alpha\in nb(i)} m^{(t)}_{i\leftarrow\alpha}(x_i) \tag{21.38}$$

and

$$P(x_{nb(\alpha)}) = \lim_{t\to\infty} \mathcal{N}(!x_{nb(\alpha)}) f_\alpha(x_{nb(\alpha)}) \prod_{i\in nb(\alpha)} m^{(t)}_{\alpha\leftarrow i}(x_i) \ . \tag{21.39}$$

Eq.(21.38) produces what is often referred to as a **sum-product decomposition**. I don't like that name because it is unnecessarily confusing and it fails to convey the recursive nature of the decomposition. I prefer to call it a **recursive sum of products (RSOP) decomposition**, and will call it so henceforth in this chapter.

Note that given a function $g : \mathbb{R}^{nx} \to \mathbb{R}$, one can find $\underset{x^{nx}}{\text{argmax}}\, g(x^{nx})$ (or argmin) by replacing all sums over x components by argmax's (or argmin's) in the MP algorithm.

Example of MP for bipartite bnets

Consider the bipartite bnet 21.8. We define its messages-bnet to be Fig.21.12. For that messages-bnet, the node TPMs, printed in blue, are as follows. They come directly from Eqs.(21.33 and (21.36).

$$P(A) = P_{\underline{A}}(A) \tag{21.40}$$

$$P(m^{(t)}_{j\leftarrow\alpha}(\cdot) \,|\, pa[m^{(t)}_{j\leftarrow\alpha}(\cdot)]) = \mathbb{1}(\quad m^{(t)}_{j\leftarrow\alpha}(x_j) = E^{(t-1)}_{(x_k)_{k\in nb(\alpha)-j}}[f_\alpha(x_{nb(\alpha)})] \quad) \tag{21.41}$$

$$P(m^{(t)}_{\alpha\leftarrow i}(\cdot) \,|\, pa[m^{(t)}_{\alpha\leftarrow i}(\cdot)]) = \mathbb{1}(\quad m^{(t)}_{\alpha\leftarrow i}(x_i) = \prod_{\beta\in nb(i)-\alpha} m^{(t-1)}_{i\leftarrow\beta}(x_i) \quad) \tag{21.42}$$

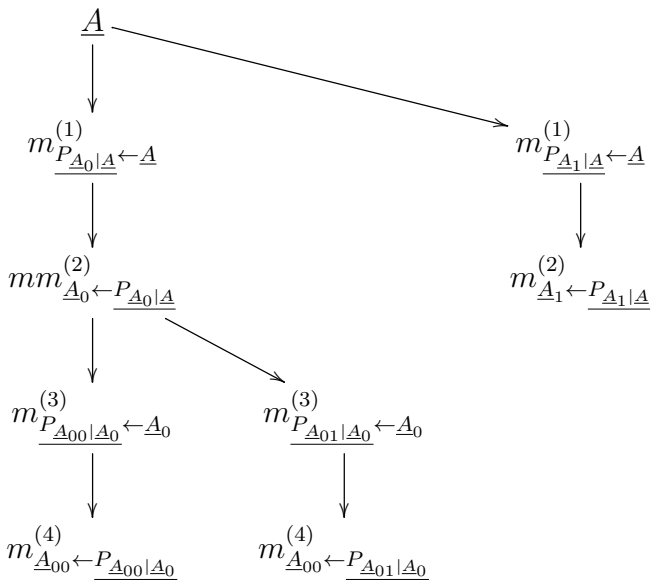


Figure 21.12: messages-bnet corresponding to tree bnet Fig.21.8.

Note that for the tree bnet Fig.21.8, one has

$$P(A_{00}) = \sum_{A,A_0} P(A_{00}|A_0)P(A_0|A)P(A) \tag{21.43}$$

$$= \sum_{A_0} \left\{ P(A_{00}|A_0) \sum_{A} \{P(A_0|A)P(A)\} \right\} \tag{21.44}$$

The right hand side of Eq.(21.44) is expressed as a recursive sum of products (RSOP). In fact, if we consider the path

$$\underline{A}_{00} \longrightarrow \underline{P_{\underline{A}_{00}|\underline{A}_0}} \longleftarrow \underline{A}_0 \longrightarrow \underline{P_{\underline{A}_0|\underline{A}}} \longleftarrow \underline{A} \longrightarrow \underline{P_{\underline{A}}} \tag{21.45}$$

in the bipartite bnet Fig.21.9, we get the same RSOP from the MP algorithm:

$$P(A_{00}) = \mathcal{N}(!A_{00}) m_{\underline{A}_{00}\leftarrow\underline{P_{\underline{A}_{00}|\underline{A}_0}}}(A_{00}) \tag{21.46}$$

$$m_{\underline{A}_{00}\leftarrow\underline{P_{\underline{A}_{00}|\underline{A}_0}}}(A_{00}) = \sum_{A_0} P(A_{00}|A_0) m_{\underline{P_{\underline{A}_{00}|\underline{A}_0}}\leftarrow\underline{A}_0}(A_0) \tag{21.47}$$

$$m_{\underline{P_{\underline{A}_{00}|\underline{A}_0}}\leftarrow\underline{A}_0}(A_0) = m_{\underline{A}_0\leftarrow\underline{P_{\underline{A}_0|\underline{A}}}}(A_0) \tag{21.48}$$

$$m_{\underline{A}_0\leftarrow\underline{P_{\underline{A}_0|\underline{A}}}}(A_0) = \sum_{A} P(A_0|A) m_{\underline{P_{\underline{A}_0|\underline{A}}}\leftarrow\underline{A}}(A) \tag{21.49}$$

$$m_{\underline{P_{\underline{A}_0|\underline{A}}}\leftarrow\underline{A}}(A) = P(A) \tag{21.50}$$

Chapter 22

Monty Hall Problem

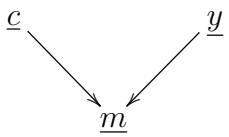


Figure 22.1: Monty Hall Problem.

Mr. Monty Hall, host of the game show "Lets Make a Deal", hides a car behind one of three doors and a goat behind each of the other two. The contestant picks Door No. 1, but before opening it, Mr. Hall opens Door No. 2 to reveal a goat. Should the contestant stick with No. 1 or switch to No. 3?

The Monty Hall problem can be modeled by the bnet Fig.22.1, where

- $\underline{c}$= the door behind which the car actually is.
- $\underline{y}$= the door opened by you (the contestant), on your first selection.
- $\underline{m}$= the door opened by Monty (game host)

We label the doors 1,2,3 so $S_{\underline{c}} = S_{\underline{y}} = S_{\underline{m}} = \{1, 2, 3\}$.
Node matrices printed in blue:

$$P(c) = \frac{1}{3} \text{ for all } c \tag{22.1}$$

$$P(y) = \frac{1}{3} \text{ for all } y \tag{22.2}$$

$$P(m|c, y) = \mathbb{1}(m \neq c) \left[\frac{1}{2}\mathbb{1}(y = c) + \mathbb{1}(y \neq c)\mathbb{1}(m \neq y)\right] \tag{22.3}$$

It's easy to show that the above node probabilities imply that

$$P(c = 1|m = 2, y = 1) = \frac{1}{3} \tag{22.4}$$

$$P(c = 3|m = 2, y = 1) = \frac{2}{3} \tag{22.5}$$

So you are twice as likely to win if you switch your final selection to be the door which is neither your first choice nor Monty's choice.

The way I justify this to myself is: Monty gives you a piece of information. If you don't switch your choice, you are wasting that info, whereas if you switch, you are using the info.

Chapter 23

Naive Bayes

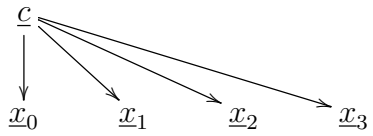


Figure 23.1: bnet for Naive Bayes with 4 features

Class node $\underline{c} \in S_{\underline{c}}$. $|S_{\underline{c}}| = n_{\underline{c}}$= number of classes.
Feature nodes $\underline{x}_i \in S_{\underline{x}_i}$ for $i = 0, 1, 2, \ldots, F-1$. F=number of features.
Define

$$x. = [x_0, x_1, \ldots, x_{F-1}] \; . \tag{23.1}$$

For the bnet of Fig.23.1,

$$P(c, x.) = P(c) \prod_{i=0}^{F-1} P(x_i|c) \; . \tag{23.2}$$

Given $x.$ values, find most likely class $c \in S_{\underline{c}}$.
Maximum a Posteriori (MAP) estimate:

$$\begin{aligned} c^* &= \operatorname*{argmax}_c P(c|x.) && (23.3) \\ &= \operatorname*{argmax}_c \frac{P(c, x.)}{P(x.)} && (23.4) \\ &= \operatorname*{argmax}_c P(c, x.) \; . && (23.5) \end{aligned}$$

Chapter 24

Neural Networks

In this chapter, we discuss Neural Networks (NNs) of the feedforward kind, which is the most popular kind. In their plain, vanilla form, NNs only have deterministic nodes. But the nodes of a bnet can be deterministic too, because the TPM of a node can reduce to a delta function. Hence, NNs should be expressible as bnets. We will confirm this in this chapter.

Henceforth in this chapter, if we replace an index of an indexed quantity by a dot, it will mean the collection of the indexed quantity for all values of that index. For example, $x.$ will mean the array of x_i for all i.

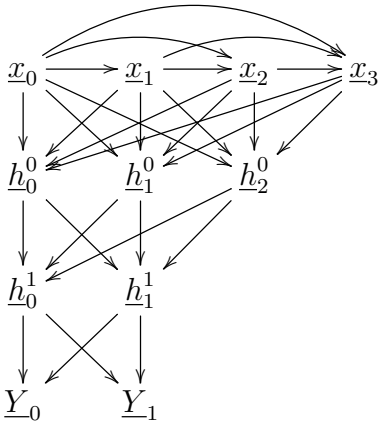


Figure 24.1: Neural Network (feed forward) with 4 layers: input layer $\underline{x}.$, 2 hidden layers $\underline{h}^0.$, $\underline{h}^1.$ and output layer $\underline{Y}$.

Consider Fig.24.1.

$\underline{x}_i \in \{0,1\}$ for $i = 0, 1, 2, \ldots, nx-1$ is the **input layer**.

$\underline{h}^\lambda_i \in \mathbb{R}$ for $i = 0, 1, 2, \ldots, nh(\lambda)-1$ is the **λ-th hidden layer**. $\lambda = 0, 1, 2, \ldots, \Lambda-2$. A NN is said to be **deep** if $\Lambda > 2$; i.e., if it has more than one hidden layer.

$\underline{Y}_i \in \mathbb{R}$ for $i = 0, 1, 2, \ldots, ny-1$ is the **output layer**. We use a upper case y here because in the training phase, we will use pairs $(x.[s], y.[s])$ where $y_i[s] \in \{0,1\}$ for $i = 0, 1, \ldots, ny-1$. $Y = \hat{y}$ is an estimate of y. Note that lower case y is either 0 or 1, but upper case y may be any

real. Often, the activation functions are chosen so that $Y \in [0,1]$.

The number of nodes in each layer and the number of layers are arbitrary. Fig.24.1 is fully connected (aka dense), meaning that every node of a layer is impinged arrow coming from every node of the preceding layer. Later on in this chapter, we will discuss non-dense layers.

Let $w^\lambda_{i|j}, b^\lambda_i \in \mathbb{R}$ be given, for $i \in \mathbb{Z}_{[0,nh(\lambda))}$, $j \in \mathbb{Z}_{[0,nh(\lambda-1))}$, and $\lambda \in \mathbb{Z}_{[0,\Lambda)}$.

These are the TPMs, printed in blue, for the nodes of the bnet Fig.24.1:

$$P(x_i \mid x_{i-1}, x_{i-1}, \ldots, x_0) = \text{given} \tag{24.1}$$

$$P(h^\lambda_i \mid h^{\lambda-1}_\cdot) = \delta\left(h^\lambda_i, \mathcal{A}^\lambda_i(\sum_j w^\lambda_{i|j} h^{\lambda-1}_j + b^\lambda_i)\right) \ , \tag{24.2}$$

where $P(h^0_i|h^{-1}) = P(h^0_i|x)$.

$$P(Y_i \mid h^{\Lambda-2}_\cdot) = \delta\left(Y_i, \mathcal{A}^{\Lambda-1}_i(\sum_j w^{\Lambda-1}_{i|j} h^{\Lambda-2}_j + b^{\Lambda-1}_i)\right) \ . \tag{24.3}$$

Activation Functions $\mathcal{A}^\lambda_i : \mathbb{R} \to \mathbb{R}$

Activation functions must be nonlinear.

- **Step function (Perceptron)**

$$\mathcal{A}(x) = \mathbb{1}(x > 0) \tag{24.4}$$

  Zero for $x \leq 0$, one for $x > 0$.

- **Sigmoid function**

$$\mathcal{A}(x) = \frac{1}{1+e^{-x}} = \text{sig}(x) \tag{24.5}$$

  Smooth, monotonically increasing function. $\text{sig}(-\infty) = 0$,$\text{sig}(0) = 0.5$, $\text{sig}(\infty) = 1$.

$$\begin{aligned} \text{sig}(x) + \text{sig}(-x) &= \frac{1}{1+e^{-x}} + \frac{1}{1+e^{x}} && (24.6)\\ &= \frac{2+e^{x}+e^{-x}}{2+e^{x}+e^{-x}} && (24.7)\\ &= 1 && (24.8) \end{aligned}$$

- **Hyperbolic tangent**

$$\mathcal{A}(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{24.9}$$

Smooth, monotonically increasing function. $\tanh(-\infty) = -1$,$\tanh(0) = 0$, $\tanh(\infty) = 1$.

Odd function:

$$\tanh(-x) = -\tanh(x) \tag{24.10}$$

Whereas $\text{sig}(x) \in [0, 1]$, $\tanh(x) \in [-1, 1]$.

- **ReLU (Rectified Linear Unit)**

$$\mathcal{A}(x) = x\mathbb{1}(x > 0) = \max(0, x) \ . \tag{24.11}$$

Compare this to the step function.

- **Swish**

$$\mathcal{A}(x) = x \ \text{sig}(x) \tag{24.12}$$

- **Softmax**

$$\mathcal{A}(x_i|x.) = \frac{e^{x_i}}{\sum_i e^{x_i}} \tag{24.13}$$

It's called softmax because if we approximate the exponentials, both in the numerator and denominator of Eq.(24.13), by the largest one, we get

$$\mathcal{A}(x_i|x.) \approx \mathbb{1}(x_i = \max_k x_k) \ . \tag{24.14}$$

The softmax definition implies that the bnet nodes within a softmax layer are fully connected by arrows to form a "clique".

For 2 nodes x_0, x_1,

$$\begin{aligned} \mathcal{A}(x_0|x.) &= \frac{e^{x_0}}{e^{x_0} + e^{x_1}} && (24.15) \\ &= \text{sig}(x_0 - x_1) \ , && (24.16) \end{aligned}$$

$$\mathcal{A}(x_1|x.) = \text{sig}(x_1 - x_0) \ . \tag{24.17}$$

Weight optimization via supervised training and gradient descent

The bnet of Fig.24.1 is used for classification of a single data point $x.$. It assumes that the weights $w^{\lambda}_{i|j}, b^{\lambda}_{i}$ are given.

To find the optimum weights via supervised training and gradient descent, one uses the bnet Fig.24.2.

In Fig.24.2, the nodes in Fig.24.1 become sampling space vectors. For example, $\underline{x}.$ becomes $\underline{\vec{x}}.$, where the components of $\underline{\vec{x}}.$ in sampling space are $\underline{x}.[s] \in \{0,1\}^{nx}$ for $s = 0, 1, \ldots, nsam(\vec{x}) - 1$.

$nsam(\vec{x})$ is the number of samples used to calculate the gradient during each **stage (aka iteration)** of Fig.24.2. We will also refer to $nsam(\vec{x})$ as the **mini-batch size**. A **mini-batch** is a subset of the training data set.

To train a bnet with a data set (d-set), the standard procedure is to split the d-set into 3 parts:

1. **training d-set**,
2. **testing1 d-set**, for tuning of hyperparameters like $nsam(\vec{x})$, Λ, and $nunh(i)$ for each i.
3. **testing2 d-set**, for measuring how well the model tuned with the testing1 d-set performs.

The training d-set is itself split into mini-batches. An **epoch** is a pass through all the training d-set.

Define

$$W^{\lambda}_{i|j} = [w^{\lambda}_{i|j}, b^{\lambda}_{i}] \; . \tag{24.18}$$

These are the TPMs, printed in blue, for the nodes of the bnet Fig.24.2:

$$P(x.[s]) = \text{given} \; . \tag{24.19}$$

$$P(y.[s] \mid x.[s]) = \text{given} \; . \tag{24.20}$$

$$P(h^{\lambda}_{i}[s] \mid h^{\lambda-1}_{\cdot}[s]) = \delta \left(h^{\lambda}_{i}[s], \mathcal{A}^{\lambda}_{i}(\sum_{j} w^{\lambda}_{i|j} h^{\lambda-1}_{j}[s] + b^{\lambda}_{i}) \right) \tag{24.21}$$

$$P(Y_{i}[s] \mid h^{\Lambda-2}_{\cdot}[s]) = \delta \left(Y_{i}[s], \mathcal{A}^{\Lambda-1}_{i}(\sum_{j} w^{\Lambda-1}_{i|j} h^{\Lambda-2}_{j}[s] + b^{\Lambda-1}_{i}) \right) \tag{24.22}$$

$$P(W^{\cdot}_{\cdot|\cdot}) = \text{given} \tag{24.23}$$

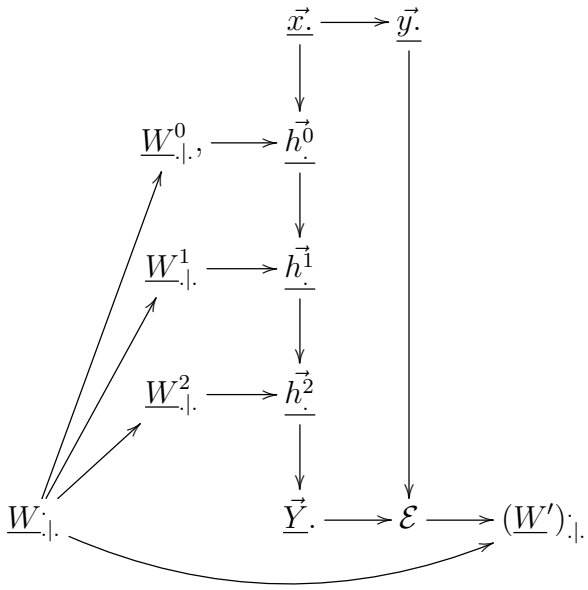


Figure 24.2: bnet for finding optimum weights of the bnet Fig.24.1 via supervised training and gradient descent.

The first time it is used, $W_{\cdot|\cdot}$ is arbitrary. After the first time, it is determined by previous stage.

$$P(W^\lambda_{\cdot|\cdot}|W^{\cdot}_{\cdot|\cdot}) = \delta(W^\lambda_{\cdot|\cdot}, (W^{\cdot}_{\cdot|\cdot})^\lambda) \tag{24.24}$$

$$P(\mathcal{E}|\vec{y}., \vec{Y}.) = \frac{1}{nsam(\vec{x})} \sum_s \sum_i d(y_i[s], Y_i[s]) \ , \tag{24.25}$$

where

$$d(y, Y) = |y - Y|^2 \ . \tag{24.26}$$

If $y, Y \in [0, 1]$, one can use this instead

$$d(y, Y) = XE(y \to Y) = -y \ln Y - (1 - y) \ln(1 - Y) \ . \tag{24.27}$$

$$P((W')^\lambda_{i|j}|\mathcal{E}, W^{\cdot}_{\cdot|\cdot}) = \delta((W')^\lambda_{i|j}, W^\lambda_{i|j} - \eta \partial_{W^\lambda_{i|j}} \mathcal{E}) \tag{24.28}$$

$\eta > 0$ is called the learning rate. This method of minimizing the error $\mathcal{E}$ is called gradient descent. $W' - W = \Delta W = -\eta \partial_W \mathcal{E}$ so $\Delta \mathcal{E} = \frac{-1}{\eta} (\Delta W)^2 < 0$.

Non-dense layers

The TPM for a non-dense layer is of the form:

$$P(h_i^\lambda[s] \mid h_\cdot^{\lambda-1}[s]) = \delta(h_i^\lambda[s], H_i^\lambda[s]) \ , \tag{24.29}$$

where $H_i^\lambda[s]$ will be specified below for each type of non-dense layer.

- **Dropout Layer**

  The dropout layer was invented in Ref.[25]. To dropout nodes from a fixed layer λ: For all i of layer λ, define a new node $\underline{r}_i^\lambda$ with an arrow $\underline{r}_i^\lambda \to \underline{h}_i^\lambda$. For $r \in \{0,1\}$, and some $p \in (0,1)$, define

  $$P(r_i^\lambda = r) = [p]^r[1-p]^{1-r} \text{ (Bernouilli dist.)} \ . \tag{24.30}$$

  Now one has

  $$P(h_i^\lambda[s] \mid h_\cdot^{\lambda-1}[s], r_i^\lambda) = \delta(h_i^\lambda[s], H_i^\lambda[s]) \ , \tag{24.31}$$

  where

  $$H_i^\lambda[s] = \mathcal{A}_i^\lambda(r_i^\lambda \sum_j w_{i|j}^\lambda h_j^{\lambda-1}[s] + b_i^\lambda) \ . \tag{24.32}$$

  This reduces ovefitting. Overfitting might occur if the weights follow too closely several similar minibatches. This dropout procedure adds a random component to each minibatch making groups of similar minibatches less likely.

  The random $\underline{r}_i^\lambda$ nodes that induce dropout are only used in the training bnet Fig.24.2, not in the classification bnet Fig.24.1. We prefer to remove the $\underline{r}_i^\lambda$ stochasticity from classification and for Fig.24.1 to act as an average over sampling space of Fig.24.2. Therefore, if weights $w_{i|j}^\lambda$ are obtained for a dropout layer λ in Fig.24.2, then that layer is used in Fig.24.1 with no $\underline{r}_i^\lambda$ nodes but with weights $\langle r_i^\lambda \rangle w_{i|j}^\lambda = p w_{i|j}^\lambda$.

  Note that dropout adds non-deterministic nodes to a NN, which in their vanilla form only have deterministic nodes.

- **Convolutional Layer**

  - 1-dim
    Filter function $\mathcal{F} : \{0, 1, \ldots, nf-1\} \to \mathbb{R}$.
    σ=stride length

For $i \in \{0, 1, \ldots, nh(\lambda) - 1\}$, let

$$H_i^\lambda[s] = \sum_{j=0}^{nf-1} h_{j+i\sigma}^{\lambda-1}[s]\mathcal{F}(j) \ . \tag{24.33}$$

For the indices not to go out of bounds in Eq.(24.33), we must have

$$nh(\lambda - 1) - 1 = nf - 1 + (nh(\lambda) - 1)\sigma \tag{24.34}$$

so

$$nh(\lambda) = \frac{1}{\sigma}[nh(\lambda - 1) - nf] + 1 \ . \tag{24.35}$$

- 2-dim
  $h_i^\lambda[s]$ becomes $h_{(i,j)}^\lambda[s]$. Do 1-dim convolution along both i and j axes.

- **Pooling Layers (MaxPool, AvgPool)**

  Here each node i of layer λ is impinged by arrows from a subset $Pool(i)$ of the set of all nodes of the previous layer $\lambda - 1$. Partition set $\{0, , 1, \ldots, nh(\lambda - 1) - 1\}$ into $nh(\lambda)$ mutually disjoint, nonempty sets called $Pool(i)$, where $i \in \{0, 1, \ldots, nh(\lambda) - 1\}$.

  - AvgPool
    $$H_i^\lambda[s] = \frac{1}{size(Pool(i))} \sum_{j \in Pool(i)} h_j^{\lambda-1}[s] \tag{24.36}$$
  - MaxPool
    $$H_i^\lambda[s] = \max_{j \in Pool(i)} h_j^{\lambda-1}[s] \tag{24.37}$$

Autoencoder NN

If the sequence

$$nx, nh(0), nh(1), \ldots, nh(\Lambda - 2), ny \tag{24.38}$$

first decreases monotonically up to layer λ_{min}, then increases monotonically until $ny = nx$, then the NN is called an **autoencoder NN**. Autoencoders are useful for unsupervised learning and feature reduction. In this case, Y estimates x. The layers before layer λ_{min} are called the **encoder**, and those after λ_{min} are called the **decoder**. Layer λ_{min} is called the **code**.

Chapter 25

Non-negative Matrix Factorization

Based on Ref.[26].

Given matrix V, factor it into product of two matrices

$$V = WH \ , \tag{25.1}$$

where all 3 matrices have non-negative entries.
$V \in \mathbb{R}_{\geq 0}^{nv \times na}$: visible info matrix
$W \in \mathbb{R}_{\geq 0}^{nv \times nh}$: weight info matrix
$H \in \mathbb{R}_{\geq 0}^{nh \times na}$: hidden info matrix
Usually, $nv > nh < na$ so compression of information (aka dimensional reduction, clustering)

B net interpretation: Express node $\underline{v}$ as a chain of two nodes.

$$\underline{v} \longleftarrow \underline{a} \qquad = \qquad \underline{w} \longleftarrow \underline{h} \longleftarrow \underline{a}$$

Figure 25.1: B net interpretation of non-negative matrix factorization.

Node TPMs, printed in blue, for Fig.25.1.

$$P(\underline{v} = w|a) = \frac{V_{w,a}}{\sum_w V_{w,a}} \tag{25.2}$$

$$P(w|h) = \frac{W_{w,h}}{\sum_w W_{w,h}} \tag{25.3}$$

$$P(h|a) = \frac{\sum_w W_{w,h}}{\sum_w V_{w,a}} H_{h,a} \tag{25.4}$$

Simplest recursive algorithm:

Initialize: Choose nh. Choose $W^{(0)}$ and $H^{(0)}$ that have non-negative entries.

Update: For $n = 0, 1, \ldots,$ do

$$H_{i,j}^{(n+1)} \leftarrow H_{i,j}^{(n)} \frac{[(W^{(n)})^T V]_{i,j}}{[(W^{(n)})^T \underbrace{W^{(n)} H^{(n)}}_{\approx V}]_{i,j}} \tag{25.5}$$

and

$$W_{i,j}^{(n+1)} \leftarrow W_{i,j}^{(n)} \frac{[V(H^{(n+1)})^T]_{i,j}}{[\underbrace{W^{(n)} H^{(n+1)}}_{\approx V}(H^{(n+1)})^T]_{i,j}} \, . \tag{25.6}$$

After each step, record error defined by

$$\mathcal{E}^{(n)} = \| \, V - W^{(n)} H^{(n)} \, \|_2 \ . \tag{25.7}$$

Using 2-norm, aka Frobenius matrix norm. Continue until reach acceptable error.

Can also use Kullback-Lieber divergence for error:

$$\mathcal{E} = \sum_a P(a) D_{KL}(P(\underline{v} = w|a) \parallel \sum_h P(w|h) P(h|a)) \, , \tag{25.8}$$

for some arbitrary choice of prior $P(a)$. For example, can choose $P(a)$ uniform.

Chapter 26

Program evaluation and review technique (PERT)

This chapter is based on Refs.[27] and [28].

PERT diagrams are used for scheduling a project consisting of a series of interdependent activities and estimating how long it will take to finish the project. PERT diagrams were invented by the NAVY in 1958 to manage a submarine project. Nowadays they are taught in many business and management courses.

A **PERT diagram** is a Directed Acyclic Graph (DAG) with the following properties. (See Fig.26.2 for an example of a PERT diagram). The nodes $\underline{E}_i$ for $i = 1, 2, \ldots, ne$ of a PERT diagram are called **events**. The edges $i \to j$ of a PERT diagram are called **activities**. An event represents the starting (kickoff) date of one or more activities. A PERT diagram has a single root node ($i = 1$, start event) and a single leaf node ($i = ne$, end event).

The PERT diagram user must initially provide a **Duration Times (DT) table** which gives $(DO_{i\to j}, DP_{i\to j}, DM_{i\to j})$ for each activity $i \to j$, where

$DO_{i\to j}$= optimistic duration time of activity $i \to j$
$DP_{i\to j}$= pessimistic duration time of activity $i \to j$
$DM_{i\to j}$= median duration time of activity $i \to j$
From the DT table, one calculates:
Duration time of activity $i \to j$

$$D_{i\to j} = \frac{1}{6}(DO_{i\to j} + DP_{i\to j} + 4DM_{i\to j}) \tag{26.1}$$

Duration Variance of activity $i \to j$

$$V_{i\to j} = \left(\frac{DO_{i\to j} - DP_{i\to j}}{DM_{i\to j}}\right)^2 \tag{26.2}$$

Often, it is convenient to define "dummy" edges with $D_{i\to j} = 0$. That is perfectly fine.
Define:
TES_i = Earliest start time for event i
TLS_i = Latest start time for event i

$slack_i = TLS_i - TES_i$ = slack for event i

$TEF_{i\to j} = TES_i + D_{i\to j}$ = Earliest finish time for activity $i \to j$.

$TLF_{i\to j} = TLS_j - D_{i\to j}$ = Latest finish time for activity $i \to j$. See footnote below. [1]

A **critical path** is a directed path (i.e., a chain of connected arrows, all pointing in the same direction) going from the start to the end node, such that slack equals zero at every node visited. In a DAG, the neighbors of a node is the union of its parent and children nodes. A critical path must also have all other nodes as neighbors; i.e, the union of the neighbors of every node in the path plus the nodes in the path itself, equals all nodes in the graph.

GOAL of PERT analysis: The main goal of PERT analysis is to find, based on the data of the DT table, the interval $[TES_i, TLS_i]$ giving a lower and an upper bound to the starting time of each node i. Another goal is to find a critical path for the PERT diagram (which represents an entire project). By adding the $D_{i\to j}$ of each edge of the critical path, one can get the mean value of the total duration of the entire project, and by adding the variances of each edge along the critical path, one can get an estimate of the total variance of the total duration. Knowing the mean and variance of the total duration and assuming a normal distribution, one can predict the probability that the actual duration will deviate by a certain amount from its mean.

To calculate the interval $[TES_i, TLS_i]$, one follows the following two steps.

1. Assume $TES_1 = 0$ and solve

$$TES_i = \max_{a\in pa(i)} (\underbrace{TES_a + D_{a\to i}}_{TEF_{a\to i}}) \tag{26.3}$$

for $i \in [2, ne]$. This recursive equation is solved by what is called "forward propagation", wherein one moves up the list of nodes i in order of increasing i starting at $i = 1$ with $TES_1 = 0$.

2. Assume $TLS_{ne} = TES_{ne}$ and solve

$$TLS_i = \min_{b\in ch(i)} (\underbrace{TLS_b - D_{i\to b}}_{TLF_{i\to b}}) \tag{26.4}$$

for $i \in [1, ne-1]$. This recursive equation is solved by what is called "backward propagation", wherein one moves down the list of nodes i in order of decreasing i starting at $i = ne$ with $TLS_{ne} = TES_{ne}$. TES_{ne} is known from step 1.

Eqs.(26.3) and (26.4) are illustrated in Fig.26.1.

[1] In the popular educational literature, the edge variables $TEF_{i\to j}$ and $TLF_{i\to j}$ are sometimes associated with the nodes, but they are clearly edge variables. This makes things confusing. The reason this is done is that some software draws PERT diagrams as trees whereas other software draws them as DAGs. For trees, storing $TEF_{i\to j}$ and $TLF_{i\to j}$ in a node makes some sense but not for DAGs. You will notice that giving specific names to the variables $TEF_{i\to j}$ and $TLF_{i\to j}$ is unnecessary. It is possible to delete all mention of their names from this chapter without losing any details. I only declare their names in this chapter so as tell the reader what they are in case he/she hears them mentioned and wonders what they are equal to in our notation.

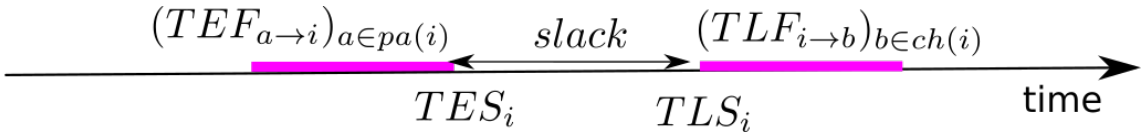


Figure 26.1: TES_i defined from info received from parents of i and TLS_i defined from info received from children of i.

Example

To illustrate PERT analysis, we end with an example. We present the example in the form of an exercise question and then provide the answer. This example comes from Ref.[27], except for part (e) about bnets, which is our own.

Question: For the PERT diagram of Fig.26.2, calculate the following:

(a) Interval $[TES_i, TLS_i]$ for all i.

(b) A critical path for this PERT diagram.

(c) The mean and variance of the total duration of the critical path.

(d) The probability that the total duration will be 225 days or less.

(e) A bnet interpretation of this problem.

Answer to (a) $[TES_i, TLS_i]$ are given by Fig.26.3.

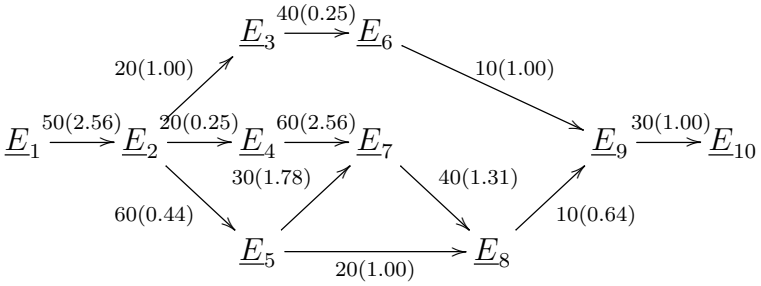
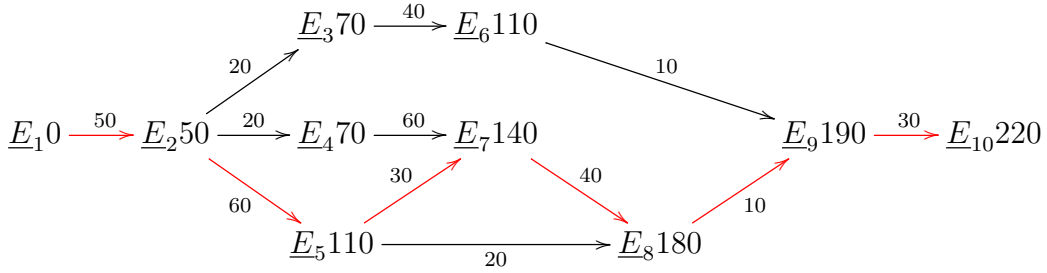
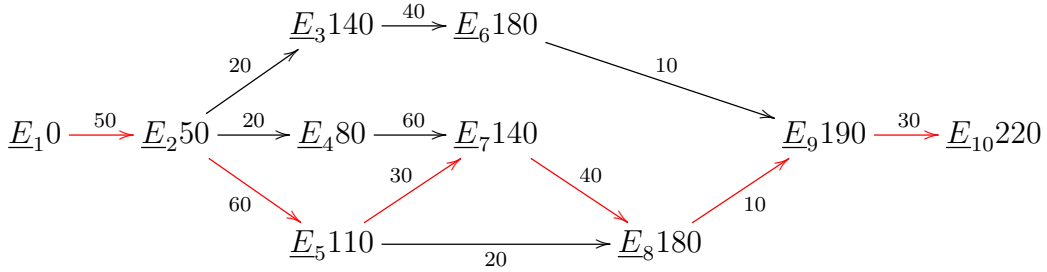


Figure 26.2: Example of a PERT diagram. The numbers attached to the arrows are the duration times $D_{i\to j}$ in days followed by, enclosed in parentheses, the variance $V_{i\to j}$ of that duration. The info given in this PERT diagram was derived from a DT table in Ref.[27]. The info in this PERT diagram is sufficient for calculating TES_i and TLS_i for each node i. The results of that calculation are given in Fig.26.3.

Answer to (b) The critical path is given in red in Fig.26.3. Note that this path does indeed have zero slack at each node it visits and the union of its neighborhood and the path itself encompasses all nodes.



TES_i (given after the node name) for node $\underline{E}_i$ for all i



TLS_i (given after the node name) for node $\underline{E}_i$ for all i

Figure 26.3: Results of calculating TES_i for all i via a forward pass, followed by calculating TLS_i for all i via a backward pass. Critical path indicated in red.

Answer to (c) The mean and variance of the total duration are calculated in Table 26.1.

Answer to (d)

$$\begin{aligned} P(x < 225) &= p\left[\frac{x-\mu}{\sigma} \leq \frac{225-220}{\sqrt{7.73}}\right] && (26.5)\\ &= P[z \leq 1.80] && (26.6)\\ &= 0.9641 && (26.7)\end{aligned}$$

Answer to (e) Define 2 bnets.

1. The first PERT bnet is for calculating TES_i for all i and is given by Fig.26.4.
   The node TPMs, printed in blue, for the bnet Fig.26.4 are given by (this equation is to be evaluated recursively by a forward pass through the bnet):

$$P(TES_i|(TES_a)_{a\in pa(i)}) = \delta(TES_i, \max_{a\in pa(i)}(TES_a + D_{a\to i})) \quad (26.8)$$

2. The second PERT bnet is for calculating TLS_i for all i and is given by Fig.26.5. Note that the directions of all the arrows in the PERT diagram Fig.26.2 have been reversed so Fig.26.5 is a time reversed graph.

edge $i \to j$	duration $D_{i\to j}$	variance $V_{i\to j}$
A $(1 \to 2)$	50	2.56
D $(2 \to 5)$	60	0.44
G $(5 \to 7)$	30	1.78
J $(7 \to 8)$	40	1.31
K $(8 \to 9)$	10	0.64
L $(9 \to 10)$	30	1.00
Total	220	7.73

Table 26.1: Calculation of mean and variance of total duration along critical path.

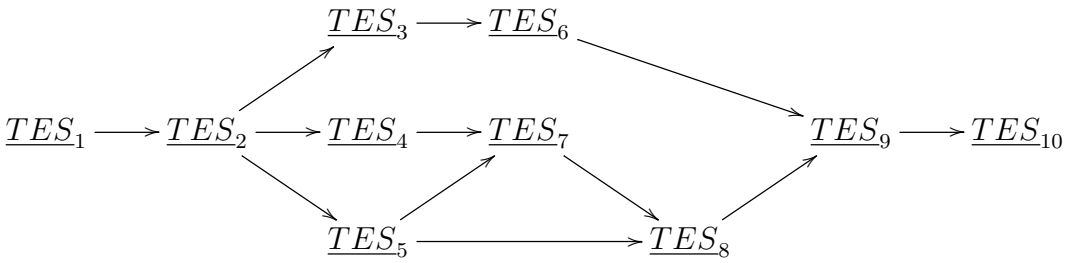


Figure 26.4: bnet for TES_i calculation.

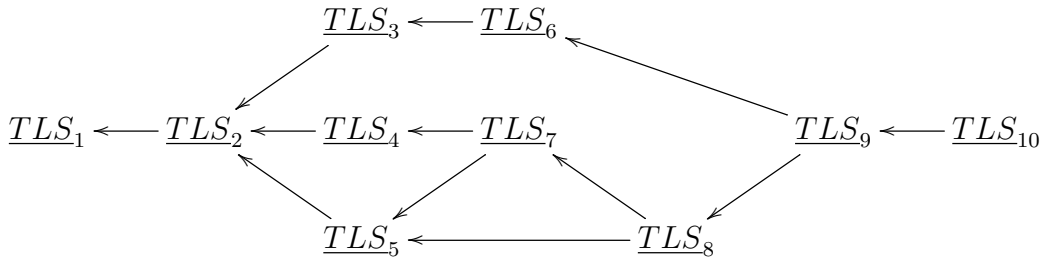


Figure 26.5: bnet for TLS_i calculation.

The node TPMs, printed in blue, for the bnet Fig.26.5 are given by (this equation is to be evaluate recursively by a backward pass through the bnet):

$$P(TLS_i|(TLS_b)_{b\in pa(i)}) = \delta(TLS_i, \min_{b\in pa(i)}(TLS_b - D^T_{b\to i})) \ , \tag{26.9}$$

where $D^T_{i\to j} = D_{j\to i}$.

Chapter 27

Recurrent Neural Networks

This chapter is mostly based on Ref.[29].

This chapter assumes you are familiar with the material and notation of Chapter 24 on plain Neural Nets.

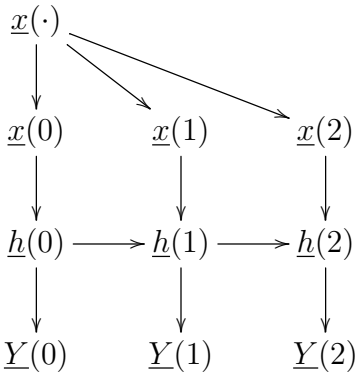


Figure 27.1: Simple example of RNN witb $T = 3$

Suppose
T is a positive integer.
$t = 0, 1, \ldots, T-1$,
$\underline{x}_i(t) \in \mathbb{R}$ for $i = 0, 1, \ldots, numx - 1$,
$\underline{h}_i(t) \in \mathbb{R}$ for $i = 0, 1, \ldots, numh - 1$,
$\underline{Y}_i(t) \in \mathbb{R}$ for $i = 0, 1, \ldots, numy - 1$,
$W^{h|x} \in \mathbb{R}^{numh \times numx}$,
$W^{h|h} \in \mathbb{R}^{numh \times numh}$,
$W^{y|h} \in \mathbb{R}^{numy \times numh}$,
$b^y \in \mathbb{R}^{numy}$,
$b^h \in \mathbb{R}^{numh}$.
Henceforth, $x(\cdot)$ will mean the array of $x(t)$ for all t.

The simplest kind of recurrent neural network (RNN) has the bnet Fig.27.1 with arbitrary T. The node TPMs, printed in blue, for this bnet, are as follows.

$$P(x(\cdot)) = \text{given} \tag{27.1}$$

$$P(x(t)) = \delta(x(t), [x(\cdot)]_t) \tag{27.2}$$

$$P(h(t) \mid h(t-1), x(t)) = \delta(h(t), \mathcal{A}(W^{h|x} x(t) + W^{h|h} h(t-1) + b^h)) \ , \tag{27.3}$$

where $h(-1) = 0$.

$$P(Y(t) \mid h(t)) = \delta(Y(t), \mathcal{A}(W^{y|h} h(t) + b^y)) \tag{27.4}$$

Define

$$W^h = [W^{h|x}, W^{h|h}, b^h] \ , \tag{27.5}$$

and

$$W^y = [W^{y|h}, b^y] \ . \tag{27.6}$$

The bnet of Fig.27.1 can be used for classification once its parameters W^h and W^y have been optimized. To optimize those parameters via gradient descent, one can use the bnet of Fig.27.2.

Let $s = 0, 1, \ldots, nsam(\vec{x}) - 1$ be the labels for a minibatch of samples. The node TPMs, printed in blue, for bnet Fig.27.2, are as follows.

$$P(x(\cdot)[s]) = \text{given} \tag{27.7}$$

$$P(x(t)[s]) = \delta(x(t)[s], [x(\cdot)]_t[s]) \tag{27.8}$$

$$P(h(t)[s] \mid h(t-1)[s], x(t)[s]) = \delta(h(t)[s], \mathcal{A}(W^{h|x} x(t)[s] + W^{h|h} h(t-1)[s] + b^h) \tag{27.9}$$

$$P(Y(t)[s] \mid h(t-1)[s]) = \delta(Y(t)[s], \mathcal{A}(W^{y|h} h(t-1)[s] + b^y) \tag{27.10}$$

$$P(y(\cdot)[s] \mid x(\cdot)[s]) = \text{given} \tag{27.11}$$

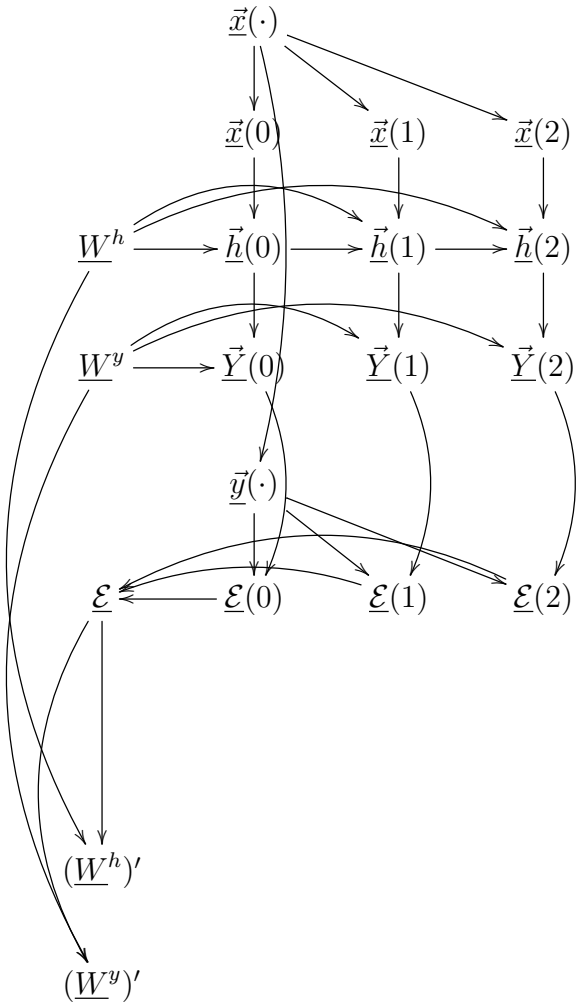


Figure 27.2: RNN bnet used to optimize parameters W^h and W^y of RNN bnet Fig.27.1.

$$P(\mathcal{E}(t) \mid \vec{y}(\cdot), \vec{Y}(t)) = \frac{1}{nsam(\vec{x})} \sum_s d(y(t)[s], Y(t)[s]) \ , \tag{27.12}$$

where

$$d(y, Y) = |y - Y|^2 \ . \tag{27.13}$$

If $y, Y \in [0, 1]$, one can use this instead

$$d(y, Y) = XE(y \to Y) = -y \ln Y - (1 - y) \ln(1 - Y) \ . \tag{27.14}$$

$$P(\mathcal{E} \mid [\mathcal{E}(t)]_{\forall t}) = \delta(\mathcal{E}, \sum_t \mathcal{E}(t)) \tag{27.15}$$

For $a = h, y$,

$$P(W^a) = \text{given} \ . \tag{27.16}$$

The first time it is used, W^a is fairly arbitrary. Afterwards, it is determined by previous horizontal stage.

$$P((W^a)'|\mathcal{E}, W^a) = \delta((W^a)', W^a - \eta^a \partial_{W^a} \mathcal{E}) \ . \tag{27.17}$$

$\eta^a > 0$ is the learning rate for W^a.

Language Sequence Modeling

Figs.27.1, and 27.2 with arbirary T can be used as follows to do Language Sequence Modeling.

For this usecase, one must train with the following TPM for node $\vec{y}(\cdot)$:

$$P(y(\cdot)[s] \mid x(\cdot)[s]) = \prod_t \mathbb{1}(\quad y(t)[s] = P(x(t)[s] \mid [x(t')[s]]_{t'<t}) \quad) \tag{27.18}$$

With such training, one gets

$$P(Y(t)|h(t)) = \mathbb{1}(\quad Y(t) = P(x(t) \mid [x(t')]_{t'<t}) \quad) \ . \tag{27.19}$$

Therefore,

$$Y(0) = P(x(0)) \ , \tag{27.20}$$

$$Y(1) = P(x(1)|x(0)) \ , \tag{27.21}$$

$$Y(2) = P(x(2)|x(0), x(1)) \ , \tag{27.22}$$

and so on.

We can use this to:

- predict the probability of a sentence,

  example: Get $P(x(0), x(1), x(2))$.

- predict the most likely next word in a sentence,

  example: Get $P(x(2)|x(0), x(1))$.

- generate fake sentences.

  example:

  Get $x(0) \sim P(x(0))$.

  Next get $x(1) \sim P(x(1)|x(0))$.

  Next get $x(2) \sim P(x(2)|x(0), x(1))$.

Other types of RNN

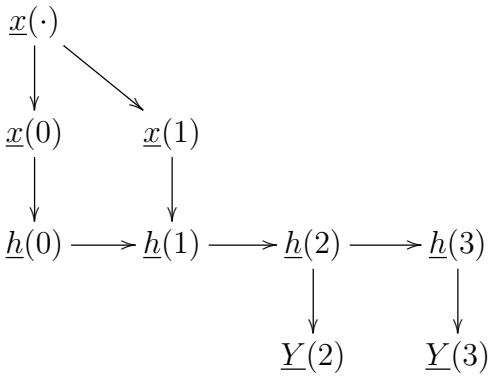


Figure 27.3: RNN bnet of the many to many kind. This one can be used for translation. $x(0)$ and $x(1)$ might denote two words of an English sentence, and $Y(2)$ and $Y(3)$ might be their Italian translation.

Let $\mathcal{T} = \{0, 1, \ldots, T-1\}$, and $\mathcal{T}^x, \mathcal{T}^y \subset \mathcal{T}$. Above, we assumed that $\underline{x}(t)$ and $\underline{Y}(t)$ were both defined for all $t \in \mathcal{T}$. More generally, they might be defined only for subsets of $\mathcal{T}$: $\underline{x}(t)$ for $t \in \mathcal{T}^x$ and $\underline{Y}(t)$ for $t \in \mathcal{T}^y$. If $|\mathcal{T}^x| = 1$ and $|\mathcal{T}^y| > 1$, we say the RNN bnet is of the 1 to many kind. In general, can have **1 to 1, 1 to many, many to 1, many to many** RNN bnets.

Plain RNNs can suffer from the **vanishing or exploding gradients problem**. There are various ways to mitigate this (good choice of initial W^h and W^y, good choice of activation functions, regularization). Or by using GRU or LSTM (discussed below). **GRU and LSTM** were designed to mitigate the vanishing or exploding gradients problem. They are very popular in NLP (Natural Language Processing).

Long Short Term Memory (LSTM) unit (1997)

This section is based on Wikipedia article Ref.[30]. In this section, $\odot$ will denote the Hadamard matrix product (elementwise product).

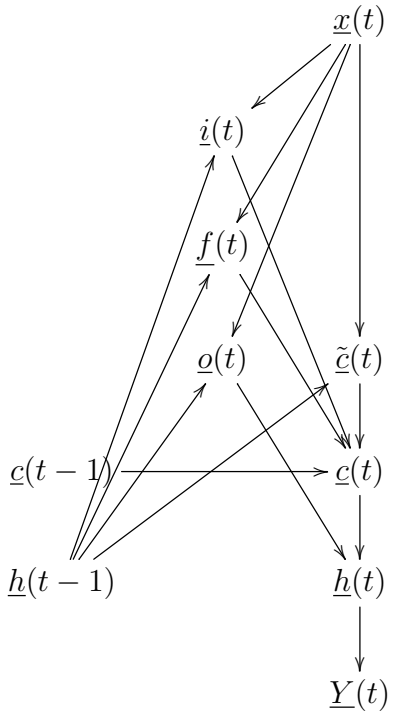


Figure 27.4: bnet for a Long Short Term Memory (LSTM) unit.

Let
$\underline{x}(t) \in \mathbb{R}^{numx}$: input vector to the LSTM unit
$\underline{f}(t) \in \mathbb{R}^{numh}$: forget gate's activation vector
$\underline{i}(t) \in \mathbb{R}^{numh}$: input/update gate's activation vector
$\underline{o}(t) \in \mathbb{R}^{numh}$: output gate's activation vector
$\underline{h}(t) \in \mathbb{R}^{numh}$: hidden state vector also known as output vector of the LSTM unit
$\underline{\tilde{c}}(t) \in \mathbb{R}^{numh}$: cell input activation vector
$\underline{c}(t) \in \mathbb{R}^{numh}$: cell state vector
$\underline{Y}(t) \in \mathbb{R}^{numy}$: classification of $x(t)$.
$W \in \mathbb{R}^{numh \times numx}$, $U \in \mathbb{R}^{numh \times numh}$ and $b \in \mathbb{R}^{numh}$: weight matrices and bias vectors, parameters learned by training.
$\mathcal{W}^{y|h} \in \mathbb{R}^{numy \times numh}$: weight matrix
Fig.27.4 is a bnet net for a LSTM unit. The node TPMs, printed in blue, for this bnet, are as follows.

$$P(f(t)|x(t), h(t-1)) = \mathbb{1}(\quad f(t) = \text{sig}(W^{f|x}x(t) + U^{f|h}h(t-1) + b^f) \quad), \tag{27.23}$$

where $h(-1) = 0$.

$$P(i(t)|x(t), h(t-1)) = \mathbb{1}(\quad i(t) = \text{sig}(W^{i|x}x(t) + U^{i|h}h(t-1) + b^i) \quad) \tag{27.24}$$

$$P(o(t)|x(t), h(t-1)) = \mathbb{1}(\quad o(t) = \text{sig}(W^{o|x}x(t) + U^{o|h}h(t-1) + b^o) \quad) \tag{27.25}$$

$$P(\tilde{c}(t)|x(t), h(t-1)) = \mathbb{1}(\quad \tilde{c}(t) = \tanh(W^{c|x}x(t) + U^{c|h}h(t-1) + b^c) \quad) \tag{27.26}$$

$$P(c(t)|f(t), c(t-1), i(t), \tilde{c}(t)) = \mathbb{1}(\quad c(t) = f(t) \odot c(t-1) + i(t) \odot \tilde{c}(t) \quad) \tag{27.27}$$

$$P(h(t)|o(t), c(t)) = \mathbb{1}(\quad h(t) = o(t) \odot \tanh(c(t)) \quad) \tag{27.28}$$

$$P(Y(t)|h(t)) = \mathbb{1}(\quad Y(t) = \mathcal{A}(\mathcal{W}^{y|h}h(t) + b^y) \quad) \tag{27.29}$$

Gated Recurrence Unit (GRU) (2014)

This section is based on Wikipedia article Ref.[31]. In this section, $\odot$ will denote the Hadamard matrix product (elementwise product).

GRU is a more recent (17 years later) attempt at simplifying LSTM unit.

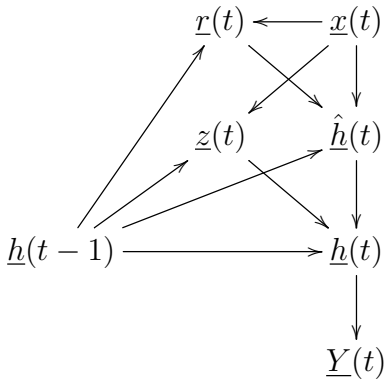


Figure 27.5: bnet for a Gated Recurrent Unit (GRU).

Let
$\underline{x}(t) \in \mathbb{R}^{numx}$: input vector
$\underline{h}(t) \in \mathbb{R}^{numh}$: output vector
$\underline{\hat{h}}(t) \in \mathbb{R}^{numh}$: candidate activation vector
$\underline{z}(t) \in \mathbb{R}^{numh}$: update gate vector
$\underline{r}(t) \in \mathbb{R}^{numh}$: reset gate vector
$\underline{Y}(t) \in \mathbb{R}^{numy}$: classification of $x(t)$.
$W \in \mathbb{R}^{numh \times numx}$, $U \in \mathbb{R}^{numh \times numh}$ and $b \in \mathbb{R}^{numh}$: weight matrices and bias vectors, parameters learned by training.
$\mathcal{W}^{y|h} \in \mathbb{R}^{numy \times numh}$: weight matrix

Fig.27.5 is a bnet net for a GRU. The node TPMs, printed in blue, for this bnet, are as follows.

$$P(z(t)|x(t), h(t-1)) = \mathbb{1}(\quad z(t) = \text{sig}(W^{z|x} x(t) + U^{z|h} h(t-1) + b^z) \quad) , \tag{27.30}$$

where $h(-1) = 0$.

$$P(r(t)|x(t), h(t-1)) = \mathbb{1}(\quad r(t) = \text{sig}(W^{r|x} x(t) + U^{r|h} h(t-1) + b^r) \quad) \tag{27.31}$$

$$P(\hat{h}(t)|x(t), r(t), h(t-1)) = \mathbb{1}(\quad \hat{h}(t) = \tanh(W^{h|x} x(t) + U^{h|h} (r(t) \odot h(t-1)) + b^h) \quad) \tag{27.32}$$

$$P(h(t)|z(t), h(t-1), \hat{h}(t)) = \mathbb{1}(\quad h(t) = (1 - z(t)) \odot h(t-1) + z(t) \odot \hat{h}(t) \quad) \tag{27.33}$$

$$P(Y(t)|h(t)) = \mathbb{1}(\quad Y(t) = \mathcal{A}(\mathcal{W}^{y|h} h(t) + b^y) \quad) \tag{27.34}$$

Chapter 28

Reinforcement Learning (RL)

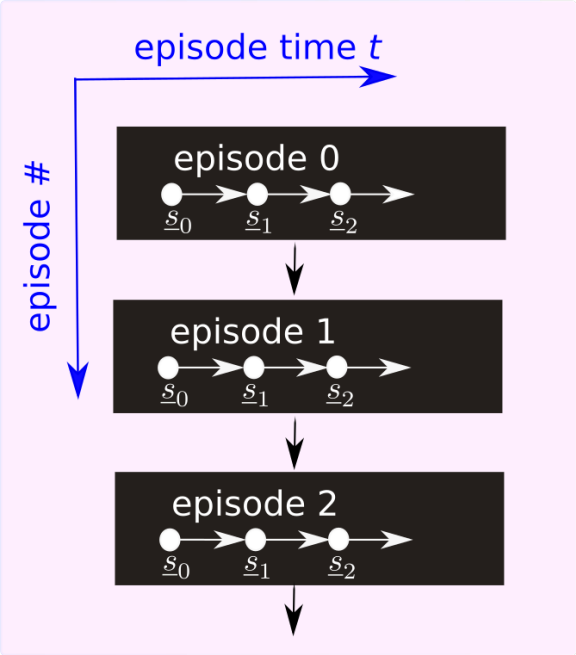


Figure 28.1: Axes for episode time and episode number.

I based this chapter on the following references. Refs.[32][33]

In RL, we consider an "agent" or robot that is learning.

Let $T \in \mathbb{Z}_{>0}$ be the duration time of an **episode** of learning. If $T = \infty$, we say that the episode has an infinite time horizon. A learning episode will evolve towards the right, for times $t = 0, 1, \ldots, T-1$. We will consider multiple learning episodes. The episode number will evolve from top to bottom. This is illustrated in Fig.28.1.

Let $\underline{s}_t \in S_{\underline{s}}$ for $t \in \mathbb{Z}_{[0,T-1]}$ be random variables that record the **state** of the agent at various times t.

Let $\underline{a}_t \in S_{\underline{a}}$ for $t \in \mathbb{Z}_{[0,T-1]}$ be random variables that record the **action** of the agent at various times t.

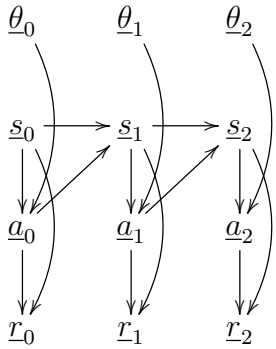


Figure 28.2: State-Action-Reward dynamical bnet

Let $\underline{\theta}_t \in S_{\underline{\theta}}$ for $t \in \mathbb{Z}_{[0,T-1]}$ be random variables that record the **policy parameters** at various times t.

For $\underline{X} \in \{\underline{s}, \underline{a}, \underline{\theta}\}$, define $\underline{X}$ followed by a dot to be the vector

$$\underline{X}. = [\underline{X}_0, \underline{X}_1, \ldots, \underline{X}_{T-1}] \,. \tag{28.1}$$

Also let

$$\underline{X}_{\geq t} = [\underline{X}_t, \underline{X}_{t+1}, \ldots, \underline{X}_{T-1}] \,. \tag{28.2}$$

Fig.28.2 shows the basic State-Action-Reward bnet for an agent that is learning. The TPMs for the nodes of Fig.28.2 are given in blue below:

$$P(a_t|s_t, \theta_t) = \text{given.} \tag{28.3}$$

$P(a_t|s_t, \theta_t)$ is called a **policy with parameter θ_t.**

$$P(s_t|s_{t-1}, a_{t-1}) = \text{given.} \tag{28.4}$$

$P(s_t|s_{t-1}, a_{t-1})$ is called the **TPM of the model**. $P(s_t|s_{t-1}, a_{t-1})$ reduces to $P(s_0)$ when $t = 0$.

$$P(r_t|s_t, a_t) = \delta(r_t, r(s_t, a_t))) \,. \tag{28.5}$$

$r : S_{\underline{s}} \times S_{\underline{a}} \to \mathbb{R}$ is a given **one-time reward function**.

Note that

$$P(s., a.|\theta.) = \prod_{t=0}^{T-1} \{P(s_t|s_{t-1}, a_{t-1}) P(a_t|s_t, \theta_t)\} \,. \tag{28.6}$$

Define the **all times reward** Σ by

$$\Sigma(s., a.) = \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t) \,. \tag{28.7}$$

Here $0 < \gamma < 1$. γ, called the **discount rate**, is included to assure convergence of Σ when $T \to \infty$. If $r(s_t, a_t) < K$ for all t, then $\Sigma < K\frac{1}{1-\gamma}$.

Define the **objective (i.e. goal) function** $E\Sigma(\theta.)$ by

$$E\Sigma(\theta.) = E_{\underline{s}.,\underline{a}.|\theta.}\Sigma(\underline{s}., \underline{a}.) = \sum_{s.,a.} P(s., a.|\theta.)\Sigma(s., a.) \tag{28.8}$$

The goal of RL is to maximize the objective function over its parameters $\theta.$. The parameters $\theta^*.$ that maximize the objective function are the optimum strategy:

$$\theta.^* = \underset{\theta.}{\operatorname{argmax}}\, E\Sigma(\theta.) \tag{28.9}$$

Define a **future reward** for times $\geq t$ as:

$$\Sigma_{\geq t}((s_{t'}, a_{t'})_{t' \geq t}) = \sum_{t'=t}^{T-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) \tag{28.10}$$

Define the following **expected conditional future rewards** (rewards for times $\geq t$, conditioned on certain quantities having given values):

$$\begin{aligned} v_t &= v(s_t, a_t; \theta.) = E_{\underline{s}.,\underline{a}.|s_t,a_t,\theta.}[\Sigma_{\geq t}] && (28.11)\\ V_t &= V(s_t; \theta.) = E_{\underline{s}.,\underline{a}.|s_t,\theta.}[\Sigma_{\geq t}] = E_{\underline{a}_t|s_t,\theta.}[v(s_t, \underline{a}_t; \theta.)] && (28.12) \end{aligned}$$

v is usually called Q in the literature. We will refer to Q as v in order to follow a convention wherein an $\underline{a}_t$-average changes a lower case letter to an upper case one.

We will sometimes write $v(s_t, a_t)$ instead of $v(s_t, a_t; \theta.)$.

Since $E\Sigma_{\geq t}$ only depends on $\theta_{\geq t}$, $v(s_t, a_t; \theta.) = v(s_t, a_t; \theta_{\geq t})$, and $V(s_t; \theta.) = V(s_t; \theta_{\geq t})$.

Note that the objective function $E\Sigma$ can be expressed in terms of v_0 by averaging over its unaveraged parameters:

$$E\Sigma(\theta.) = E_{\underline{s}_0,\underline{a}_0|\theta_0} v(\underline{s}_0, \underline{a}_0; \theta.) \tag{28.13}$$

Define a **one-time reward** and an **expected conditional one-time reward** as:

$$\begin{aligned} r_t &= r(s_t, a_t) && (28.14)\\ R_t &= R(s_t; \theta_t) = E_{\underline{a}_t|s_t,\theta_t}[r(s_t, \underline{a}_t)] \ . && (28.15) \end{aligned}$$

Note that

$$\begin{aligned} \Sigma_{\geq t} &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{T-1-t} r_{t+(T-1-t)} && (28.16)\\ &= r_t + \gamma \Sigma_{\geq t+1}; . && (28.17) \end{aligned}$$

If we take $E_{\underline{s}.,\underline{a}.|s_t,a_t,\theta.}[\cdot]$ of both sides of Eq.(28.17), we get

$$v_t = r_t + \gamma E_{\underline{s}_{t+1},\underline{a}_{t+1}|\theta.}[v_{t+1}] \ . \tag{28.18}$$

If we take $E_{\underline{s}_\cdot,\underline{a}_\cdot|s_t,\theta_\cdot}[\cdot]$ of both sides of Eq.(28.17), we get

$$V_t = R_t + \gamma E_{\underline{s}_{t+1}|\theta_\cdot}[V_{t+1}] \ . \tag{28.19}$$

Note that

$$\begin{aligned} \Delta r_t &= r_t - R_t && (28.20)\\ &= r_t - (V_t - \gamma E_{\underline{s}_{t+1}|\theta_\cdot}[V_{t+1}]) && (28.21)\\ &= r_t + \gamma E_{\underline{s}_{t+1}|\theta_\cdot}[V_{t+1}] - V_t \ . && (28.22) \end{aligned}$$

Define

$$\Delta v_t = v_t - V_t \ . \tag{28.23}$$

Note that

$$\Delta v_t = \Delta r_t \ . \tag{28.24}$$

Next, we will discuss 3 RL bnets

- exact RL bnet (exact, assumes policy is known)
- Actor-Critic RL bnet (approximate, assumes policy is known)
- Q function learning RL bnet (approximate, assumes policy is NOT known)

Exact RL bnet

An exact RL bnet is given by Fig.28.3.

Fig.28.3 is the same as Fig.28.2 but with more nodes added in order to optimize the policy parameters. Here are the TPMs, printed in blue, for the nodes not already discussed in connection to Fig.28.2.

$$P(\theta_t|\theta_\cdot) = \delta(\theta_t, (\theta_\cdot)_t) \tag{28.25}$$

$$\forall (s_t, a_t): \quad P(v_t(s_t,a_t)|r_t, v_{t+1}(\cdot), \theta_\cdot) = \delta(v_t(s_t,a_t), r_t + \gamma E_{\underline{s}_{t+1},\underline{a}_{t+1}|\theta_\cdot}[v_{t+1}]) \tag{28.26}$$

$$P(\theta_\cdot'|\theta_\cdot, v_0(\cdot)) = \delta(\theta_\cdot', \theta_\cdot + \alpha \partial_{\theta_\cdot} \underbrace{E_{\underline{s}_0,\underline{a}_0|\theta_0} v(\underline{s}_0, \underline{a}_0; \theta_\cdot)}_{E\Sigma(\theta_\cdot)}) \tag{28.27}$$

$\alpha > 0$ is called the **learning rate**. This method of improving $\theta_\cdot$ is called gradient ascent.

Concerning the gradient of the objective function, note that

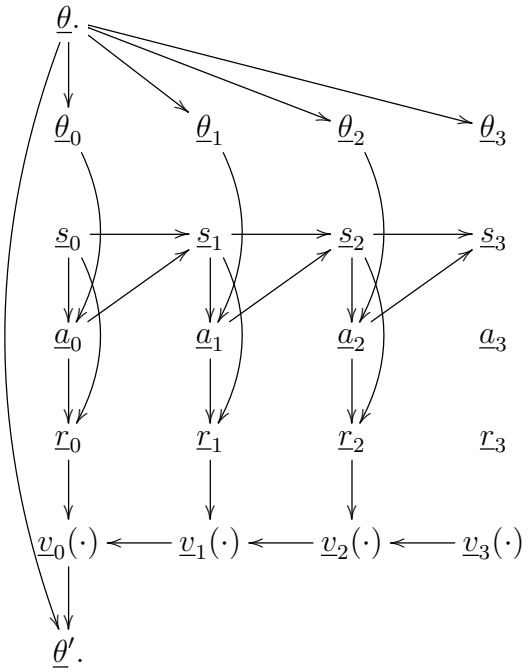


Figure 28.3: Exact RL bnet. $v_t(\cdot)$ means the array $[v_t(s_t, a_t)]_{\forall s_t, a_t}$ The following arrows are implicit: for all t, arrow from $\underline{\theta}. \to \underline{v}_t(\cdot)$. We did not draw those arrows so as not to clutter the diagram.

$$\partial_{\theta_t} E\Sigma(\theta.) = \sum_{s.,a.} \partial_{\theta_t} P(s.,a.|\theta.)\Sigma(s.,a.) \tag{28.28}$$

$$= \sum_{s.,a.} P(s.,a.|\theta.)\partial_{\theta_t} \ln P(s.,a.|\theta.)\Sigma(s.,a.) \tag{28.29}$$

$$= E_{\underline{s}.,\underline{a}.|\theta.} \{\partial_{\theta_t} \ln P(a_t|s_t,\theta_t)\Sigma(s.,a.)\} \ . \tag{28.30}$$

If we run the agent $nsam(\vec{s}_t)$ times and obtain samples $s_t[i], a_t[i]$ for all t and for $i = 0, 1, \ldots, nsam(\vec{s}_t)-1$, we can express this gradient as follows:

$$\partial_{\theta_t} E\Sigma(\theta.) \approx \frac{1}{nsam(\vec{s}_t)} \sum_i \sum_{t=0}^{T-1} \partial_{\theta_t} \ln P(a_t[i] \mid s_t[i],\theta_t) r(s_t[i],a_t[i]) \ . \tag{28.31}$$

The exact RL bnet Fig.28.3 is difficult to use to calculate the optimum parameters θ^*.. The problem is that $\underline{s}_t$ propagates towards the future and the $\underline{v}_t(\cdot)$ propagates towards the past, so we don't have a Markov Chain with a chain link for each t (i.e., a dynamical bnet) in the episode time direction. Hence, people have come up with approximate RL bnets that are doubly dynamical (i.e., dynamical along the episode time and episode number axes.) We discuss some of those approximate RL bnets next.

Actor-Critic RL bnet

For the actor-critic RL bnet, we approximate Eq.(28.31) by

$$\partial_{\theta_t} E\Sigma(\theta.) \approx \frac{1}{nsam(\vec{s})} \sum_i \sum_{t=0}^{T-1} \underbrace{\partial_{\theta_t} \ln P(a_t[i] \mid s_t[i],\theta_t)}_{Actor} \underbrace{\Delta r_t(s_t[i],a_t[i])}_{Critic} \tag{28.32}$$

The actor-critic RL bnet is given by Fig.28.4. This bnet is approximate and assumes that the policy is known. The TPMs for its nodes are given in blue below.

$$P(\theta_t) = \text{given} \tag{28.33}$$

$$P(s_t[i] \mid s_{t-1}[i], a_{t-1}[i]) = \text{ given} \tag{28.34}$$

$$P(a_t[i] \mid s_t[i],\theta_t) = \text{ given} \tag{28.35}$$

$$P(r_t[i] \mid s_t[i],a_t[i]) = \delta(r_t[i], r(s_t[i],a_t[i])) \tag{28.36}$$

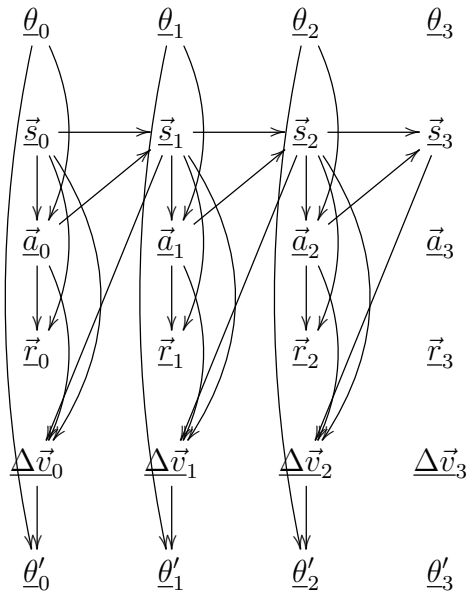


Figure 28.4: Actor-Critic RL bnet.

$r : S_{\underline{s}} \times S_{\underline{a}} \to \mathbb{R}$ is given.

$$P(\Delta v_t[i] \mid s_t[i], a_t[i], s_{t+1}[i]) = \delta(\Delta v_t[i], r(s_t[i], a_t[i]) + \gamma \hat{V}(s_{t+1}[i]; \phi') - \hat{V}(s_t[i]); \phi) \ . \tag{28.37}$$

$$P(\theta'.) = \delta(\theta'., \theta_t + \alpha \partial_{\theta_t} \sum_i \ln P(a_t[i] \mid s_t[i], \theta_t) \Delta v_t[i]) \tag{28.38}$$

$\hat{V}(s_t[i]); \phi)$ is obtained by curve fitting (see Chapter 2) using samples $(s_t[i], a_t[i])\ \forall t, i$ with

$$y[i] = \sum_{t'=t}^{T} r(s_{t'}[i], a_{t'}[i]) \tag{28.39}$$

and

$$\hat{y}[i] = \hat{V}(s_t[i]; \phi) \ . \tag{28.40}$$

Eq.(28.39) is an approximation because $(s_{t'}, a_{t'})_{t'>t}$ are averaged over in the exact expression for $V(s_t)$. $\hat{V}(s_{t+1}[i]); \phi')$ is obtained in the same way as $\hat{V}(s_t[i]); \phi)$ but with t replaced by $t+1$ and ϕ by ϕ'.

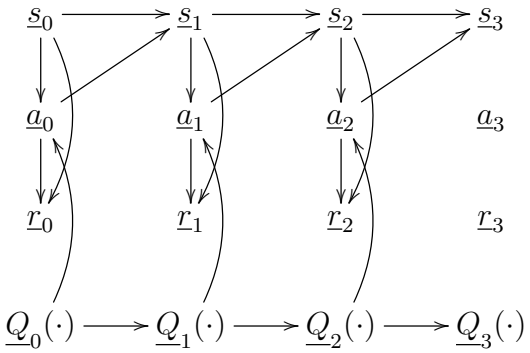


Figure 28.5: Q function learning RL bnet.

Q function learning RL bnet

The Q-function learning RL bnet is given by Fig.28.5. This bnet is approximate and assumes that the policy is NOT known. The TPMs for its nodes are given in blue below. (Remember that $Q = v$).

$$P(s_t|s_{t-1}, a_{t-1}) = \text{ given} \tag{28.41}$$

$$P(a_t|s_t, v_t(\cdot)) = \delta(a_t, \underset{a}{\text{argmax}}\, v_t(s_t, a)) \tag{28.42}$$

$$P(r_t|s_t, a_t) = \delta(r_t, r(s_t, a_t)) \tag{28.43}$$

$r : S_{\underline{s}} \times S_{\underline{a}} \to \mathbb{R}$ is given.

$$\begin{aligned} \forall (s_t, a_t): \quad & P(v_t(s_t, a_t)|v_{t-1}(\cdot)) = \\ & = \delta(v_t(s_t, a_t), r(s_t, a_t) + \gamma \text{max}_a E_{\underline{s}_{t+1}|s_t, a_t} v_{t-1}(\underline{s}_{t+1}, a)) \end{aligned} \tag{28.44}$$

This value for $v_t(s_t, a_t)$ approximates $v_t = r_t + \gamma E_{\underline{s}_{t+1}, \underline{a}_{t+1}} v_{t+1}$.

Some people use the bnet of Fig.28.6) instead of Fig.28.5 and replace Eq.(28.44) by

$$\begin{aligned} \forall (s_t, a_t): \quad & P(v_t(s_t, a_t)|s_{t+1}, v_{t-1}(\cdot)) = \\ & = \delta(v_t(s_t, a_t), r(s_t, a_t) + \gamma \text{max}_a v_{t-1}(s_{t+1}, a)) \ . \end{aligned} \tag{28.45}$$

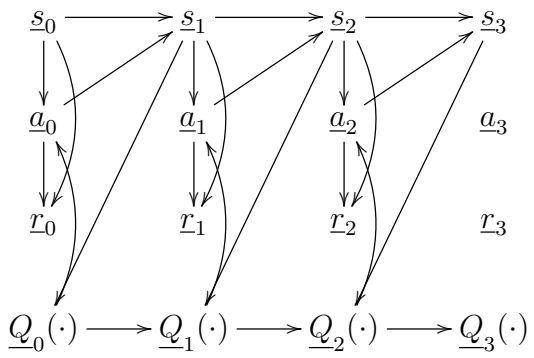


Figure 28.6: Q function learning RL bnet. Same as Fig.28.5 but with new arrow passing s_t to Q_{t-1}.

Chapter 29

Reliability Box Diagrams and Fault Tree Diagrams

This chapter is based on Refs.[34] and [35].

In this chapter, we assume that reader is familiar with Boolean Algebra. See the Notational Conventions Chapter 0.2 for a quick review of what we recommend that you know about Boolean Algebra to fully appreciate this chapter.

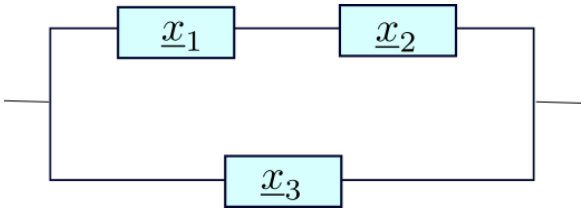


Figure 29.1: Example of rbox diagram.

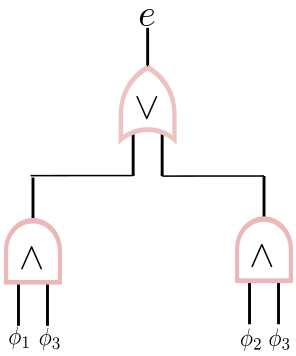


Figure 29.2: An ftree diagram equivalent to Fig.29.1. It represents $e = (\phi_1 \wedge \phi_3) \vee (\phi_2 \wedge \phi_3)$.

Complicated devices with a large number of components such as airplanes or rockets can fail in many ways. If their performance depends on some components working in series and one of the

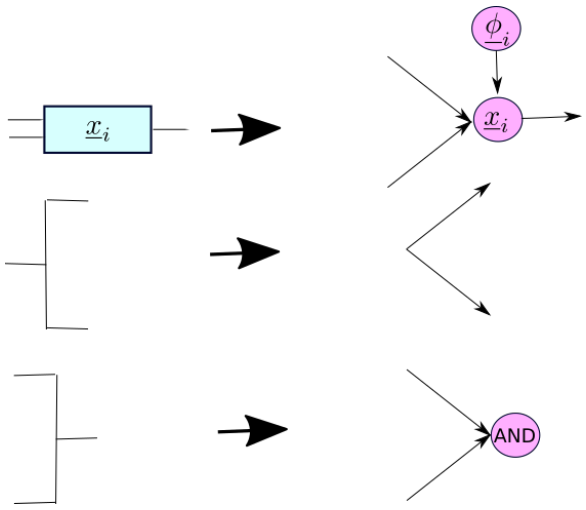


Figure 29.3: How to map an rbox diagram to a bnet.

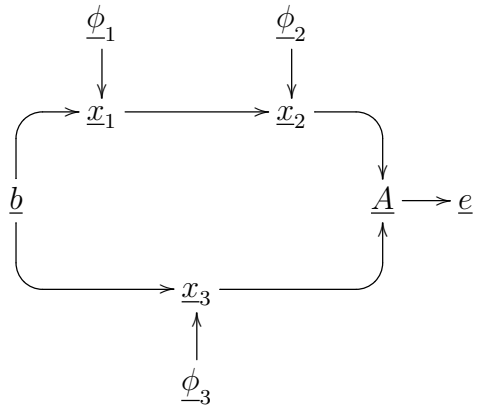


Figure 29.4: bnet corresponding the rbox diagram Fig.29.1.

components in the series fails, this may lead to catastrophic failure. To avert such disasters, engineers use equivalent components connected in parallel instead of in series, thus providing multiple backup systems. They analyze the device to find its weak points and add backup capabilities there. They also estimate the average time to failure for the device.

The two most popular diagrams for finding the failure modes and their rates for large complicated devices are

- rbox diagrams = Reliability Box diagrams. See Fig.29.1 for an example.
- ftree diagrams = Fault Tree Diagrams. See Fig.29.2 for an example.

In an ftree diagram, several nodes might stand for the same component of a physical device. In an rbox diagram, on the other hand, each node represents a distinct component in a device. Hence,

rbox diagrams resemble the device they are addressing whereas ftree diagrams don't. Henceforth, we will refer to this desirable property as **physical resemblance**.

As we will show below with an example, it is pretty straightforward to translate an rbox to an ftree diagram. Going the other way, translating an ftree to an rbox diagram is much more difficult.

Next we will define a new kind of bnet that we will call a failure bnet that has physical resemblance. Then we will describe a simple method of translating (i.e., mapping) any rbox diagram to a failure bnet. Then we will show how a failure bnet can be used to do all the calculations that are normally done with an rbox or an ftree diagram. In that sense, failure bnets seem to afford all the benefits of both ftree and rbox diagrams.

A **failure bnet** contains nodes of 5 types, labeled $\underline{b}$, $\underline{e}$, $\underline{x}_i$, $\underline{\phi}_i$, and $\underline{A}_i$. All nodes have only two possible states $S = Success = 0$, $F = Failure = 1$.

1. The bnet has a beginning node labeled $\underline{b}$ which is always set to success. The $\underline{b}$ node and the $\underline{\phi}_i$ nodes are the only root nodes of the bnet.

2. The bnet has a single leaf node, the end node, labeled $\underline{e}$. $\underline{e}$ is fixed. In rbox diagrams, $\underline{e} = S$ whereas in ftree diagrams, $\underline{e} = F$.

3. $\underline{x}^{nx} = (\underline{x}_0, \underline{x}_1, \ldots, \underline{x}_{nx-1})$. $\underline{x}_i \in \{S, F\}$ for all i.

   Suppose $\underline{x}_i$ has parents $\underline{\phi}_i$ and $\underline{a}^{na} = (\underline{a}_0, \underline{a}_1, \ldots \underline{a}_{na-1})$. Then the TPM of node $\underline{x}_i$ is defined to be

$$P(x_i|\phi_i, a^{na}) = \delta(x_i, \phi_i \vee \vee_{i=0}^{na-1} a_i) \tag{29.1}$$

4. For each node $\underline{x}_i$, the bnet has a "performance" root node $\underline{\phi}_i \in \{0, 1\}$ with an arrow pointing from it to $\underline{x}_i$ (i.e, $\underline{\phi}_i \rightarrow \underline{x}_i$). For all i,

$$P(\phi_i) = \epsilon_i \delta(\phi_i, F) + \overline{\epsilon}_i \delta(\phi_i, S) \; . \tag{29.2}$$

   ϵ_i is the failure probability and $\overline{\epsilon}_i = 1 - \epsilon_i$ is the success probability. We name the failure probability ϵ_i because it is normally very small. It is usually set to $1 - e^{-\lambda_i t} \approx \lambda_i t$ when $\lambda_i t << 1$, where λ_i is the failure rate for node $\underline{x}_i$ and t stands for time. The rblock literature usually calls $\overline{\epsilon_i} = R_i$ the **reliability** of node $\underline{x}_i$, and $\epsilon_i = (1 - R_i) = F_i$ its **unreliability**.

5. The nodes $\underline{A}_i \in \{0, 1\}$ are simply AND gates. If $\underline{A}_i$ has inputs $\underline{y}^{ny} = (\underline{y}_0, \underline{y}_1, \ldots, \underline{y}_{ny-1})$, then the TPM of $\underline{A}_i$ is

$$P(A_i|y^{ny}) = \delta(A_i, \wedge_{i=0}^{ny-1} y_i) \; . \tag{29.3}$$

An instance (instantiation) of a bnet is the bnet with all nodes set to a specific state. A **realizable instance (r-instance)** of a bnet is one which has non-zero probability.

Fig.29.3 shows how to translate any rbox diagram to a failure bnet. To illustrate this procedure, we translated the rbox diagram Fig.29.1 into the failure bnet Fig.29.4.

For the failure bnet Fig.29.4, one has:

$$\begin{array}{l} P(b) = \mathbb{1}(b=0) \\ P(x_1|\phi_1, b) = \mathbb{1}(x_1 = \phi_1 \vee b) \\ P(x_2|\phi_2, x_1) = \mathbb{1}(x_2 = \phi_2 \vee x_1) \\ P(x_3|\phi_3, b) = \mathbb{1}(x_3 = \phi_3 \vee b) \\ P(A|x_2, x_3)e = \mathbb{1}(x_2 \wedge x_3) \\ P(e|A) = \mathbb{1}(e = A) \end{array} \quad . \tag{29.4}$$

Therefore, all r-instances of this bnet must satisfy

$$\begin{aligned} e &= (\phi_1 \vee \phi_2) \wedge \phi_3 && (29.5) \\ &= (\phi_1 \wedge \phi_3) \vee (\phi_2 \wedge \phi_3) \ . && (29.6) \end{aligned}$$

Eq.(29.6) proves that Fig.29.2 is indeed a representation of Fig.29.1.

Next, we consider r-instances of this bnet for two cases: $e = S$ and $e = F$.

- **rblock analysis:** $e = S = 0$**.**
  Table 29.1 shows the probability of all possible r-instances that end in success for the failure bnet Fig.29.4. (These r-instances are the main focus of rblock analysis). The first 4 of those probabilities (those with $\phi_3 = 0$) sum to $\bar{\epsilon}_3$ so the sum $P(e = S)$ of all 5 is

  $$P(e = S) = \bar{\epsilon}_3 + \bar{\epsilon}_1\bar{\epsilon}_2\epsilon_3 \ , \tag{29.7}$$

  or, expressing it in reliability language in which $\bar{\epsilon} = R$,

  $$P(e = S) = R_3 + R_1R_2\overline{R}_3 \ . \tag{29.8}$$

- **ftree analysis:** $e = F = 1$**.**
  Table 29.2 shows the probability of all possible r-instances that end in failure for the failure bnet Fig.29.4. (These r-instances are the main focus of ftree analysis). If we set $\epsilon_i = \epsilon$ and $\bar{\epsilon}_i \approx 1$ for $i = 1, 2, 3$, then the first two of those r-instances have probabilities of $order(\epsilon^2)$ and the third has probability of $order(\epsilon^3)$. The two lowest order ($order(\epsilon^2)$) r-instances are called the "minimal cut sets" of the ftree. We will have more to say about minimal cut sets later on. For now, just note from Eq.(29.6) that the ftree Fig.29.2 is just the result of joining together with ORs two expressions, one for each of the two minimal cut sets.

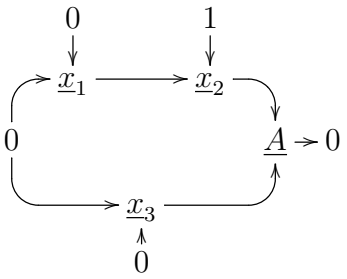
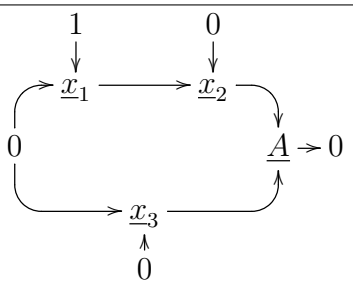
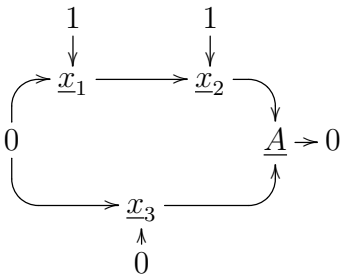
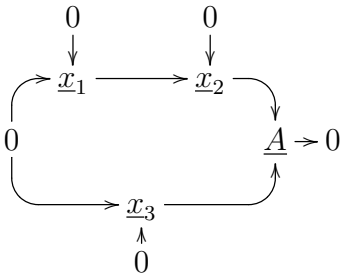
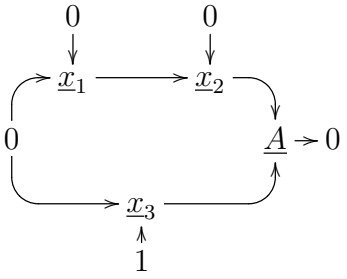
instance	probability
	$\overline{\epsilon}_1\epsilon_2\overline{\epsilon}_3$
	$\epsilon_1\overline{\epsilon}_2\overline{\epsilon}_3$
	$\epsilon_1\epsilon_2\overline{\epsilon}_3$
	$\overline{\epsilon}_1\overline{\epsilon}_2\overline{\epsilon}_3$
	$\overline{\epsilon}_1\overline{\epsilon}_2\epsilon_3$

Table 29.1: Probabilities of all possible r-instances with $e = S = 0$ for failure bnet Fig.29.4.

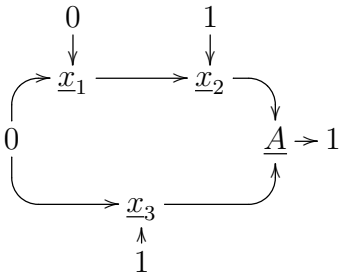
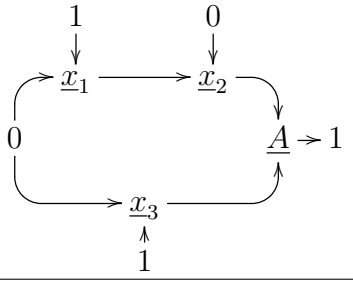
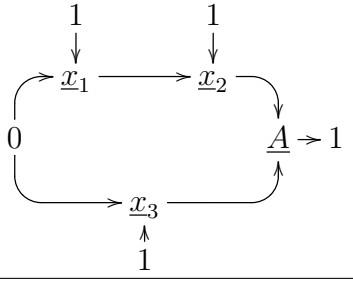
instance	probability
	$\overline{\epsilon}_1\epsilon_2\epsilon_3$
	$\epsilon_1\overline{\epsilon}_2\epsilon_3$
	$\epsilon_1\epsilon_2\epsilon_3$

Table 29.2: Probabilities of all possible r-instances with $e = F = 1$ for the failure bnet Fig.29.4.

More general $\underline{x}_i$.

Failure bnets can actually accommodate $\underline{x}_i$ nodes of a more general kind than what we first stipulated. Here are some possibilities:

For any $a^n \in \{0,1\}^n$, let

$$len(a^n) = \sum_i a_i \tag{29.9}$$

- **OR gate**

$$\begin{aligned} P(x_i|\phi_i, a^{na}) &= \delta(x_i, \phi_i \vee \vee_j a_j) && (29.10)\\ &= \delta(x_i, \phi_i \vee \mathbb{1}(len(a^{na}) > 0)) && (29.11)\end{aligned}$$

- **AND gate**

$$\begin{aligned} P(x_i|\phi_i, a^{na}) &= \delta(x_i, \phi_i \vee \wedge_j a_j) && (29.12)\\ &= \delta(x_i, \phi_i \vee \mathbb{1}(len(a^{na}) = na)) && (29.13)\end{aligned}$$

- **Fail if least K failures (less than K successes)**

$$P(x_i|\phi_i, a^{na}) = \delta(x_i, \phi_i \vee \mathbb{1}(len(a^{na}) \geq K)) \tag{29.14}$$

- **Fail if less than K failures (at least K successes)**

$$P(x_i|\phi_i, a^{na}) = \delta(x_i, \phi_i \vee \mathbb{1}(len(a^{na}) < K)) \tag{29.15}$$

- **Fail if exactly one failure**

$$P(x_i|\phi_i, a^{na}) = \delta(x_i, \phi_i \vee \mathbb{1}(len(a^{na}) = 1)) \tag{29.16}$$

  This equals an XOR (exclusive OR) gate when $na = 2$.

- **General gate**
  $f : \{0,1\}^{na} \to \{0,1\}$

$$P(x_i|\phi_i, a^{na}) = \delta(x_i, \phi_i \vee f(a^{na})) \tag{29.17}$$

Minimal Cut Sets

Suppose $x \in \{0,1\}$ and $f : \{0,1\} \to \{0,1\}$. Then by direct evaluation, we see that

$$f(x) = [\overline{x}f(0)] \vee [xf(1)] \; . \tag{29.18}$$

Let

$$\begin{array}{l} !x = 1 - x, \\ !^0 x = x, \\ !^1 x = !x \end{array} \tag{29.19}$$

Then Eq.29.18 can be rewritten as

$$f(x) = \vee_{a \in \{0,1\}} [(!^{\overline{a}} x) f(a)] \; . \tag{29.20}$$

Now suppose $x^n \in \{0,1\}^n$ and $f : \{0,1\}^n \to \{0,1\}$. Eq.(29.20) generalizes to

$$f(x^n) = \vee_{a^n \in \{0,1\}^n} [\prod_i (!^{\overline{a}_i} x_i) f(a^n)] \; . \tag{29.21}$$

Eq.(29.21) is called an **ors-of-ands** normal form expansion. There is also an **ands-of-ors** normal form expansion obtained by swapping multiplication and $\vee$ in Eq.(29.21), but we won't need it here.

A **cut set** is a set of ϕ_i's such that if they are all equal to F, then $e = F$ for all the r-instances. A **minimal cut set** is a cut set such that there are no larger cut sets that contain it. From the failure bnet, we can always find a function $f : \{0,1\}^{nx} \to \{0,1\}$ such that $e = f(\phi^{nx})$ for all the r-instances. We did that for our example failure bnet and obtained Eq.(29.6). We can then express $f(\phi^{nx})$ as an ors-of-ands expansion to find all the minimal cut sets. The ands terms in that ors-of-ands expansion each gives a different minimal cut set, after some simplification. The ors-of-ands expression is not unique and it may be necessary to simplify (using the Boolean Algebra identities given in Chapter 0.2) to remove those redundancies.

Chapter 30

Restricted Boltzmann Machines

In what follows, we will abbreviate "restricted Boltzmann machine' by rebo.

Let
$v \in \{0,1\}^{numv}$
$h \in \{0,1\}^{numh}$
$b \in \mathbb{R}^{numv}$ (mnemonic, v and b sound the same)
$a \in \mathbb{R}^{numh}$
$W^{v|h} \in \mathbb{R}^{numv \times numh}$
Energy:

$$E(v,h) = -(b^T v + a^T h + v^T W^{v|h} h) \tag{30.1}$$

Boltzmann distribution:

$$P(v,h) = \frac{e^{-E(v,h)}}{Z} \tag{30.2}$$

Partition function:

$$Z = \sum_{v,h} e^{-E(v,h)} = Z(a,b,W^{v|h}) \tag{30.3}$$

$$\begin{aligned} P(v|h) &= \frac{e^{b^T v + a^T h + v^T W^{v|h} h}}{\sum_v e^{b^T v + a^T h + v^T W^{v|h} h}} && (30.4)\\ &= \frac{e^{b^T v + v^T W^{v|h} h}}{\sum_v e^{b^T v + v^T W^{v|h} h}} && (30.5)\\ &= \prod_i \frac{e^{v_i (b_i + \sum_j W_{i,j}^{v|h} h_j)}}{\sum_{v_i=0,1} e^{v_i (b_i + \sum_j W_{i,j}^{v|h} h_j)}} && (30.6)\\ &= \prod_i P(v_i|h) && (30.7) \end{aligned}$$

$$P(v_i|h) = \frac{e^{v_i (b_i + \sum_j W_{i,j}^{v|h} h_j)}}{Z_i(h)} \tag{30.8}$$

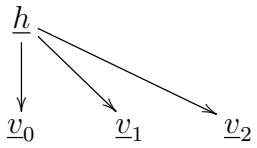


Figure 30.1: bnet for a Restricted Boltzmann Machine (rebo) with $numv = 3$

Eq.30.8 implies that a rebo can be represented by the bnet Fig.30.1.
Let

$$x_i = b_i + \sum_j W_{ij}^{v|h} h_j \ . \tag{30.9}$$

Then

$$\begin{aligned} P(v_i = 1|h) &= \frac{e^{x_i}}{1 + e^{x_i}} && (30.10) \\ &= \frac{1}{1 + e^{-x_i}} && (30.11) \\ &= \text{sig}(x_i) \ . && (30.12) \end{aligned}$$

One could also expand the node $\underline{h}$ in Fig.30.1 into $numh$ nodes. But note that $P(h) \neq \prod_j P(h_j)$ so there would be arrows among the h_j nodes.

Note that the rebo bnet is a special case of Naive Bayes (See Chapter 23) with $v_i, h_i \in \{0, 1\}$ and specific $P(h)$ and $P(v_i|h)$ node matrices.

Chapter 31

Simpson's Paradox

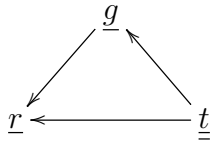


Figure 31.1: bnet for a simple case of Simpson's paradox.

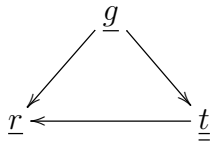


Figure 31.2: Equivalent to Fig.31.1

I wrote an article about this in 2020 for my blog "Quantum Bayesian Networks". See Ref.[36].

Chapter 32

Turbo Codes

This chapter is based on Ref.[37].

In this chapter, vectors with n components will be indicated by an n superscript. For example, $a^n = (a_0, a_1, \ldots, a_{n-1})$.

Consider an n-letter message $u^n = (u_0, u_1, \ldots, u_{n-1})$, where for all i, $u_i \in \mathcal{A}$ is an element of an alphabet $\mathcal{A}$, and where for all i, the $\underline{u}_i$ are i.i.d.. Suppose u^n is encoded deterministically in two different ways, $e_1(u^n)$ and $e_2(u^n)$. After passing through the same memoryless channel, the variables u^n, e_1, e_2 become $\tilde{u}^n, \tilde{e}_1, \tilde{e}_2$, respectively. The letter u stands for unencoded, and e for encoded. Quantities with a tilde $\tilde{u}^n, \tilde{e}_1, \tilde{e}_2$ occur after channel passage and are visible (measurable). Quantities without a tilde u^n, e_1, e_2 are hidden (unmeasurable).

The situation just described can be represented by the bnet Fig.32.1, or by its abridged version Fig.32.2. But note that the abridged version does not show explicitly that the u_i are i.i.d. or that the channel is memoryless (i.e., that the u_i for all i pass independently through the channel).

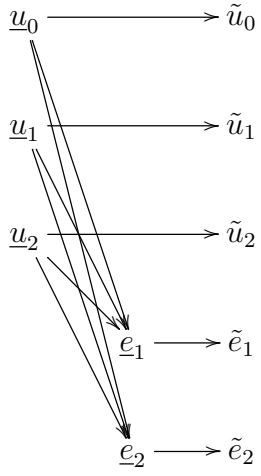


Figure 32.1: Turbo coding B net representing a message being encoded two different ways and then the original message and the 2 encodings pass through a memoryless channel.

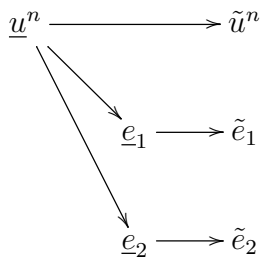


Figure 32.2: Abridged version of Fig.32.1.

Define

$$x = (u^n, e_1, e_2) \tag{32.1}$$

and

$$\tilde{x} = (\tilde{u}^n, \tilde{e}_1, \tilde{e}_2) \ . \tag{32.2}$$

Fig.32.1 implies that

$$P(x, \tilde{x}) = P(\tilde{u}^n|u^n) \left[\prod_{r=1,2} P(\tilde{e}_r|e_r) P(e_r|u^n) \right] P(u^n) \ . \tag{32.3}$$

Because the u^n are i.i.d.,

$$P(u^n) = \prod_i P(u_i) \ . \tag{32.4}$$

Because the channel is memoryless,

$$P(\tilde{u}^n|u^n) = \prod_i P(\tilde{u}_i|u_i) \ . \tag{32.5}$$

Because the encoding is deterministic, we must have for $r = 1, 2$

$$P(e_r|u^n) = \delta(e_r, e_r(u^n)) \ . \tag{32.6}$$

Define the belief functions

$$BEL_i = BEL_i(\underline{u}_i = a) = P(\underline{u}_i = a|\tilde{x}) \ . \tag{32.7}$$

The best estimate of u_j given all visible evidence $\tilde{x}$ is

$$\hat{u}_i = \underset{u_i}{\text{argmax}} \, BEL_i(u_i) \ . \tag{32.8}$$

Define the probability functions

$$\pi_i = \pi_i(u_i) = P(u_i) \ , \tag{32.9}$$

and the likelihood functions

$$\lambda_i = \lambda_i(u_i) = P(\tilde{u}_i|u_i) \ . \tag{32.10}$$

For $r = 1, 2$, define the Kernel functions

$$K_r = K_r(u^n) = P(\tilde{e}_r|e_r = e_r(u^n)) \ . \tag{32.11}$$

In this book, $\mathcal{N}(!a)$ denotes a normalization constant that does not depend on a. Define

$$\mathcal{N}_i = \mathcal{N}(!u_i) \ . \tag{32.12}$$

Claim 5

$$BEL_i = \mathcal{N}_i \lambda_i \pi_i \mathcal{T}_i^{K_1 K_2} [\prod_{j \neq i} \lambda_j \pi_j] \ , \tag{32.13}$$

where $\mathcal{T}_i^K(\cdot)$ with $K = K_1 K_2$ is an operator (transform) that acts on functions of u^n:

$$\mathcal{T}_i^K(\cdot) = \sum_{u^n} \delta(u_i, a) K(u^n)(\cdot) \ . \tag{32.14}$$

proof:

$$\begin{aligned}
P(\underline{u}_i = a|\tilde{x}) = & \\
&= \sum_x \delta(u_i, a) P(x|\tilde{x}) && (32.15)\\
&= \sum_x \delta(u_i, a) \frac{P(\tilde{x}|x)P(x)}{P(\tilde{x})} && (32.16)\\
&= \mathcal{N}(!a) \sum_x \delta(u_i, a) P(\tilde{x}|x) P(x) && (32.17)\\
&= \mathcal{N}(!a) \sum_x \delta(u_i, a) P(u^n) \left[\prod_{r=1,2} P(\tilde{e}_r|e_r) \delta(e_r, e_r(u^n)) \right] \prod_j P(\tilde{u}_j|u_j) && (32.18)\\
&= \mathcal{N}(!a) \lambda_i(a) \pi_i(a) R \ , && (32.19)
\end{aligned}$$

where

$$
\begin{aligned}
R &= \sum_{u^n} \delta(u_i, a) \left[\prod_{r=1,2} P(\tilde{e}_r | e_r(u^n)) \right] \prod_{j \neq i} P(\tilde{u}_j | u_j) P(u_j) && (32.20) \\
&= \sum_{u^n} \delta(u_i, a) \left[\prod_{r=1,2} K_r(u^n) \right] \prod_{j \neq i} \lambda_j(u_j) \pi_j(u_j) && (32.21) \\
&= \mathcal{T}_i^{K_1 K_2} [\prod_{j \neq i} \lambda_j(u_j) \pi_j(u_j)] \; . && (32.22)
\end{aligned}
$$

Hence

$$
BEL_i(a) = \mathcal{N}(!a) \lambda_i(a) \pi_i(a) \mathcal{T}_i^{K_1 K_2} [\prod_{j \neq i} \lambda_j(u_j) \pi_j(u_j)] \; . \tag{32.23}
$$

QED

Decoding Algorithm

The Turbo algorithm for decoding the encode message is as follows. For $m = 0$, let

$$
\pi_j^{(0)}(u_j) = \frac{1}{n_{\underline{u}_j}} \; . \tag{32.24}
$$

Then for $m = 1, 2, \ldots,$ let

$$
\pi_i^{(m)} = \mathcal{N}_i \mathcal{T}_i^{K_{m\%2}} [\prod_{j \neq i} \lambda_j \pi_j^{(m-1)}] \; , \tag{32.25}
$$

where $m\%2 = 1$ if m is odd and $m\%2 = 2$ if m is even. Furthermore, for $m > 0$, let

$$
\begin{aligned}
BEL_i^{(m)} &= \mathcal{N}_i \lambda_i \pi_i^{(m-1)} \pi_i^{(m)} && (32.26) \\
&= \mathcal{N}_i \lambda_i \pi_i^{(m-1)} \mathcal{T}_i^{K_{m\%2}} [\prod_{j \neq i} \lambda_j \pi_j^{(m-1)}] \; . && (32.27)
\end{aligned}
$$

As $m \to \infty$, $BEL_i^{(m)}$ given by Eq.(32.27) is expected to converge to the the exact BEL_i given by Eq.(32.13).

Turbo decoding can be represented by the bnets Figs.32.3 and 32.4.

The node TPMs, printed in blue, for Fig.32.3, are given by:

$$
P(d_i^{(m)} = a \mid \tilde{u}^n, \tilde{e}_{m\%2}) = BEL_i^{(m)}(a) \; . \tag{32.28}
$$

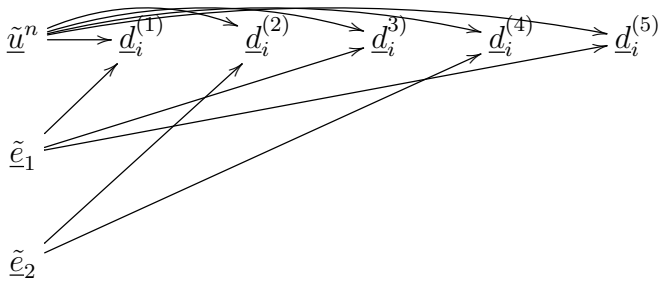


Figure 32.3: B net describing Turbo code generation of $BEL_i^{(m)}(a)$ for $m = 1, 2, \ldots$.

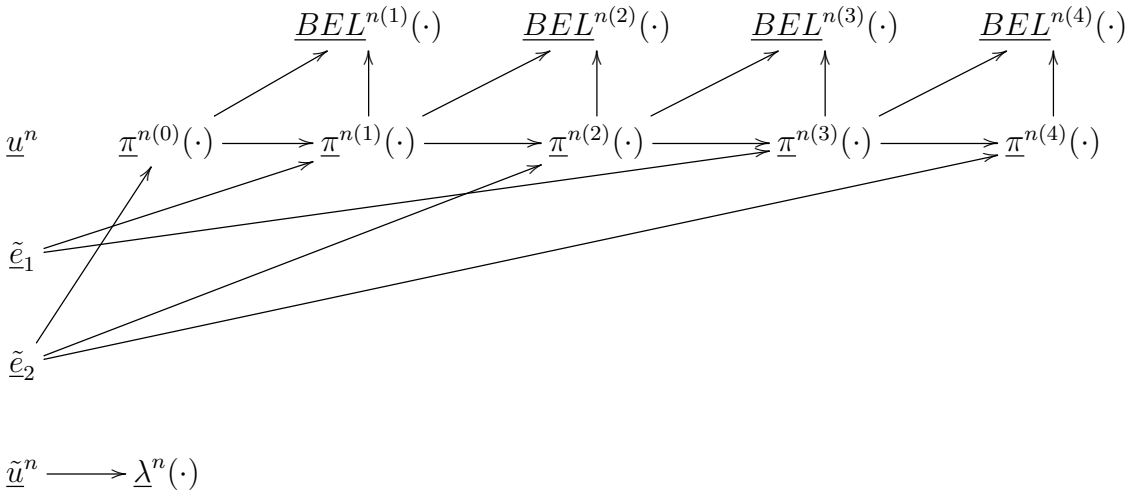


Figure 32.4: B net describing Turbo code generation of $BEL^{n(m)}(\cdot)$ and $\pi^{n(m)}(\cdot)$ for $m = 0, 1, 2 \ldots$. The following arrows were not drawn so as not to unduly clutter the diagram: Arrows pointing from node $\underline{\lambda}^n(\cdot)$ to nodes $\underline{\pi}^{n(m)}(\cdot)$ and $\underline{BEL}^{n(m)}(\cdot)$ for $m = 0, 1, 2, \ldots$.

The TPMs, printed in blue, for Fig.32.4, are given by:

$$P((\lambda^n)'(\cdot)|\tilde{u}^n) = \delta((\lambda^n)'(\cdot), \lambda^n(\cdot)) \tag{32.29}$$

$$P(\pi^{n(m)}(\cdot)|\lambda^n(\cdot), \pi^{n(m-1)}(\cdot), \tilde{e}_{m\%2}) = \prod_i \prod_{u_i} \delta(\pi_i^{(m)}(u_i), \mathcal{N}_i \mathcal{T}_i^{K_{m\%2}}[\prod_{j \neq i} \lambda_j \pi_j^{(m-1)}]) \tag{32.30}$$

$$P(BEl^{n(m)}(\cdot)|\lambda^n(\cdot), \pi^{n(m)}(\cdot), \pi^{n(m-1)}(\cdot)) = \prod_i \prod_{u_i} \delta(BEL_i(u_i), \mathcal{N}_i \lambda_i \pi_i^{(m-1)} \pi_i^{(m)}) \tag{32.31}$$

Message Passing Interpretation of Decoding Algorithm

Ref.[37] shows that the Turbo code decoding algo can be interpreted as an application of Message Passing. We leave all talk of Message Passing to a separate chapter, Chapter 21.

Chapter 33

Variational Bayesian Methods: COMING SOON

Bibliography

[1] Wikipedia. Boolean algebra. https://en.wikipedia.org/wiki/Boolean_algebra.

[2] Robert R. Tucci. Bell's inequalities for Bayesian statisticians. blog post in blog Quantum Bayesian Networks, https://qbnets.wordpress.com/2008/09/19/bells-inequaties-for-bayesian-statistician/.

[3] Wikipedia. Binary decision diagram. https://en.wikipedia.org/wiki/Binary_decision_diagram.

[4] Wikipedia. Expectation maximization. https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm.

[5] Wikipedia. k-means clustering. https://en.wikipedia.org/wiki/K-means_clustering.

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, David Warde-Farley Bing Xu, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. https://arxiv.org/abs/1406.2661.

[7] Wikipedia. Hidden Markov model. https://en.wikipedia.org/wiki/Hidden_Markov_model.

[8] Gregory Nuel. Tutorial on exact belief propagation in Bayesian networks: from messages to algorithms. https://arxiv.org/abs/1201.4724.

[9] Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):157–194, 1988. http://www.eecis.udel.edu/~shatkay/Course/papers/Lauritzen1988.pdf.

[10] Wikipedia. Junction tree algorithm. https://en.wikipedia.org/wiki/Junction_tree_algorithm.

[11] Cecil Huang and Adnan Darwiche. Inference in belief networks: A procedural guide. *International journal of approximate reasoning*, 15(3):225–263, 1996. http://www.ar-tiste.com/Huang-Darwiche1996.pdf.

[12] Robert R. Tucci. Quantum Fog. https://github.com/artiste-qb-net/quantum-fog.

[13] Wikipedia. Kalman filter. https://en.wikipedia.org/wiki/Kalman_filter.

[14] Wikipedia. Markov blanket. https://en.wikipedia.org/wiki/Markov_blanket.

[15] Judea Pearl. *Probabilistic Inference in Intelligent Systems*. Morgan Kaufmann, 1988.

[16] Wikipedia. Monte Carlo methods. https://en.wikipedia.org/wiki/Category:Monte_Carlo_methods.

[17] Wikipedia. Inverse transform sampling. https://en.wikipedia.org/wiki/Inverse_transform_sampling.

[18] Wikipedia. Rejection sampling. https://en.wikipedia.org/wiki/Rejection_sampling.

[19] Dan Bendel. Metropolis-Hastings: A comprehensive overview and proof. https://similarweb.engineering/mcmc/.

[20] Wikipedia. Metropolis-Hastings method. https://en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_algorithm.

[21] Wikipedia. Gibbs sampling. https://en.wikipedia.org/wiki/Gibbs_sampling.

[22] Wikipedia. Importance sampling. https://en.wikipedia.org/wiki/Importance_sampling.

[23] Judea Pearl. Reverend Bayes on inference engines: A distributed hierarchical approach. https://www.aaai.org/Papers/AAAI/1982/AAAI82-032.pdf, 1982.

[24] Wikipedia. Belief propagation. https://en.wikipedia.org/wiki/Belief_propagation.

[25] Nitish Srivastava, G E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf.

[26] Wikipedia. Non-negative matrix factorization. https://en.wikipedia.org/wiki/Non-negative_matrix_factorization.

[27] theinvestorsbook.com. Pert analysis. https://theinvestorsbook.com/pert-analysis.html.

[28] Wikipedia. Program evaluation and review technique. https://en.wikipedia.org/wiki/Program_evaluation_and_review_technique.

[29] Andrew Ng. Lecture at deeplearning.ai on recurrent neural networks. http://www.ar-tiste.com/ng-lec-rnn.pdf.

[30] Wikipedia. Long short term memory. https://en.wikipedia.org/wiki/Long_short-term_memory.

[31] Wikipedia. Gated recurrent unit. https://en.wikipedia.org/wiki/Gated_recurrent_unit.

[32] Charles Fox, Neil Girdhar, and Kevin Gurney. A causal bayesian network view of reinforcement learning. https://www.aaai.org/Papers/FLAIRS/2008/FLAIRS08-030.pdf".

[33] Sergey Levine. Course CS 285 at UC Berkeley, Deep reinforcement learning. http://rail.eecs.berkeley.edu/deeprlcourse/.

[34] ReliaSoft. System analysis reference. http://reliawiki.org/index.php/System_Analysis_Reference.

[35] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. Fault tree handbook nureg-0492. https://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/.

[36] Robert R. Tucci. Simpson's paradox, the bane of clinical trials. blog post in blog Quantum Bayesian Networks https://qbnets.wordpress.com/2020/07/09/simpsons-paradox-the-bane-of-clinical-trials/.

[37] Robert J. McEliece, David J. C. MacKay, and Jung-Fu Cheng. Turbo decoding as an instance of Pearls belief propagation algorithm. http://authors.library.caltech.edu/6938/1/MCEieeejstc98.pdf.