Bayesuvius,
a small visual dictionary of Bayesian Networks

Robert R. Tucci
www.ar-tiste.xyz

July 27, 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

bnet=B net=Bayesian Network

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 $\vec{x}$. $\underline{x}[i]$ are i.d.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.

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

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

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

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

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

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

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

- Entropy:

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

- Kullback-Liebler divergence:

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

- Cross entropy:

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

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

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

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

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

$$E_{\underline{x}|\underline{y}}[f(\underline{x}, \underline{y})] = \sum_x P(x|\underline{y}) f(x, \underline{y}) \ , \tag{27}$$

$$E_{\underline{x}|\underline{y}=y}[f(\underline{x}, y)] = E_{\underline{x}|y}[f(\underline{x}, y)] = \sum_x P(x|y) f(x, y) \ . \tag{28}$$

Note that

$$\begin{aligned} E_{\underline{y}}[E_{\underline{x}|\underline{y}}[f(\underline{x}, \underline{y})]] &= \sum_{x,y} P(x|y) P(y) f(x, y) && (29) \\ &= \sum_{x,y} P(x, y) f(x, y) && (30) \\ &= E_{\underline{x},\underline{y}}[f(\underline{x}, \underline{y})] \ . && (31) \end{aligned}$$

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

$$\mathrm{sig}(x) = \frac{1}{1 + e^{-x}} \tag{32}$$

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

Chapter 1

Back Propagation (Auto Differentiation): COMING SOON

Chapter 2

Basic Curve Fitting Using Gradient Descent

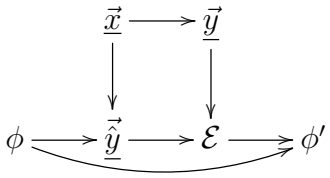


Figure 2.1: Basic curve fitting bnet.

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

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

The node transition matrices 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

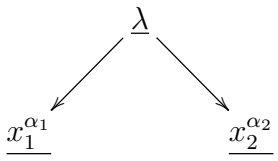


Figure 3.1: bnet used to discuss Bell and Clauser-Horne inequalities in Quantum Mechanics.

I wrote an article about this in 2008 for my blog "Quantum Bayesian Networks". See Ref.[1].

Chapter 4

Do-Calculus: COMING SOON

Chapter 5

D-Separation: COMING SOON

Chapter 6

Hidden Markov Model

A Hidden Markov Model (HMM) is a generalization of a Kalman Filter. Their bnets are the same. The only difference is that a KF assumes special node transition matrices. Kalman Filters are discussed in Chapter 8.

See Wikipedia article Ref.[2] to learn about the history and many uses of HMMs. This chapter is based on Ref.[3].

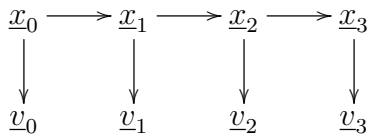


Figure 6.1: HMM bnet with $n = 4$.

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.6.1, one has

$$P(x^n, v^n) = \prod_{i=0}^{n-1} P(x_i|x_{i-1})P(v_i|x_i) \ , \tag{6.1}$$

where $x_{-1} = 0$.

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

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

$$\mathcal{F}_i(x_i) = P(v_{>i}|x_i) \tag{6.2}$$

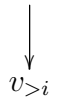
$\overline{\mathcal{F}}_i$= past and present measurements probability

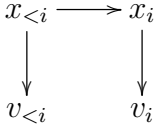
$$\overline{\mathcal{F}}_i(x_i) = P(v_{<i}, v_i, x_i) \tag{6.3}$$

λ_i= present measurement probability

$$\lambda_i(x_i) = P(v_i|x_i) \tag{6.4}$$

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

$\mathcal{F}_i(x_i) = \frac{1}{P(x_i)} \sum_{x_{>i}}$  (6.5)

$\overline{\mathcal{F}}_i(x_i) = \sum_{x_{<i}}$  (6.6)

$\lambda_i(x_i) = \frac{1}{P(x_i)}$  (6.7)

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

$$P(x_i, v^n) = \overline{\mathcal{F}}_i(x_i)\mathcal{F}_i(x_i) \ . \tag{6.8}$$

For $i > 0$,

$$P(x_{i-1}, x_i, v^n) = \overline{\mathcal{F}}_{i-1}(x_{i-1})\lambda_i(x_i)P(x_i|x_{i-1})\mathcal{F}_i(x_i) \ . \tag{6.9}$$

proof:

$$\begin{aligned}
P(x_i, v^n) &= \sum_{x_{<i}} \sum_{x_{>i}} P(x^n, v^n) && (6.10)\\
&= \sum_{x_{<i}} \sum_{x_{>i}} P(x^n, v^n|x_i)P(x_i) && (6.11)\\
&= \sum_{x_{<i}} \sum_{x_{>i}} P(x_{<i}, v_{<i}, v_i|x_i)P(x_{>i}, v_{>i}|x_i)P(x_i) && (6.12)\\
&= P(v_{<i}, v_i|x_i)P(v_{>i}|x_i)P(x_i) && (6.13)\\
&= \overline{\mathcal{F}}_i(x_i)\mathcal{F}_i(x_i) && (6.14)
\end{aligned}$$

$$
\begin{aligned}
P(x_{i-1}, x_i, v^n) &= \sum_{x_{<i-1}} \sum_{x_{>i}} P(x^n, v^n) && (6.15)\\
&= \sum_{x_{<i-1}} \sum_{x_{>i}} P(x^n, v^n | x_{i-1}, x_i) P(x_{i-1}, x_i) && (6.16)\\
&= \sum_{x_{<i-1}} \sum_{x_{>i}} P(x_{<i-1}, v_{<i-1}, v_{i-1} | x_{i-1}) P(v_i | x_i) P(x_{i-1}, x_i) P(x_{>i}, v_{>i} | x_i) && (6.17)\\
&= P(v_{<i-1}, v_{i-1} | x_{i-1}) P(v_i | x_i) P(x_{i-1}, x_i) P(v_{>i} | x_i) && (6.18)\\
&= \overline{\mathcal{F}}_{i-1}(x_{i-1}) \lambda_i(x_i) P(x_i | x_{i-1}) \mathcal{F}_i(x_i) && (6.19)
\end{aligned}
$$

QED

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

$$\overline{\mathcal{F}}_i(x_i) = \sum_{x_{i-1}} \overline{\mathcal{F}}_{i-1}(x_{i-1}) \lambda_i(x_i) P(x_i | x_{i-1}) \tag{6.20}$$

$$\mathcal{F}_{i-1}(x_{i-1}) = \sum_{x_i} \lambda_i(x_i) P(x_i | x_{i-1}) \mathcal{F}_i(x_i) \tag{6.21}$$

proof:

$$
\begin{aligned}
\overline{\mathcal{F}}_i(x_i) \mathcal{F}_i(x_i) &= P(x_i, v^n) && (6.22)\\
&= \sum_{x_{i-1}} P(x_{i-1}, x_i, v^n) && (6.23)\\
&= \sum_{x_{i-1}} \overline{\mathcal{F}}_{i-1}(x_{i-1}) \lambda_i(x_i) P(x_i | x_{i-1}) \mathcal{F}_i(x_i) && (6.24)
\end{aligned}
$$

$$
\begin{aligned}
\overline{\mathcal{F}}_{i-1}(x_{i-1}) \mathcal{F}_{i-1}(x_{i-1}) &= P(x_{i-1}, v^n) && (6.25)\\
&= \sum_{x_i} P(x_{i-1}, x_i, v^n) && (6.26)\\
&= \sum_{x_i} \overline{\mathcal{F}}_{i-1}(x_{i-1}) \lambda_i(x_i) P(x_i | x_{i-1}) \mathcal{F}_i(x_i) && (6.27)
\end{aligned}
$$

QED

Claim 3

$$P(x_i | x_{i-1}, v^n) = \frac{\lambda_i(x_i) \mathcal{F}_i(x_i)}{\mathcal{F}_{i-1}(x_{i-1})} P(x_i | x_{i-1}) \tag{6.28}$$

$$P(x_{i-1} | x_i, v^n) = \frac{\lambda_i(x_i) \overline{\mathcal{F}}_{i-1}(x_{i-1})}{\overline{\mathcal{F}}_i(x_i)} P(x_i | x_{i-1}) \tag{6.29}$$

proof:

$$P(x_i|x_{i-1},v^n) = \frac{P(x_{i-1},x_i,v^n)}{P(x_{i-1},v^n)} \tag{6.30}$$

$$= \frac{\overline{\mathcal{F}}_{i-1}(x_{i-1})\lambda_i(x_i)P(x_i|x_{i-1})\mathcal{F}_i(x_i)}{\overline{\mathcal{F}}_{i-1}(x_{i-1})\mathcal{F}_{i-1}(x_{i-1})} \tag{6.31}$$

Analogous proof for Eq.(6.29).
QED

Chapter 7

Generative Adversarial Networks (GANs)

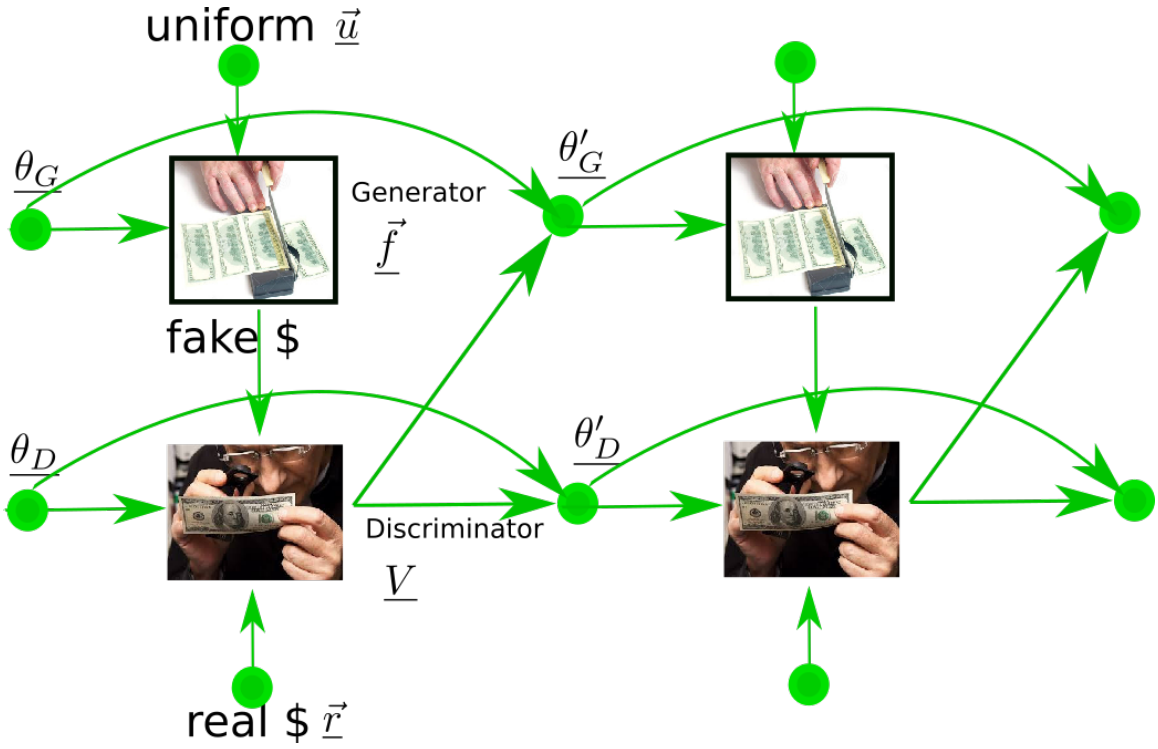


Figure 7.1: Generative Adversarial Network (GAN)

Original GAN, Ref.[4](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

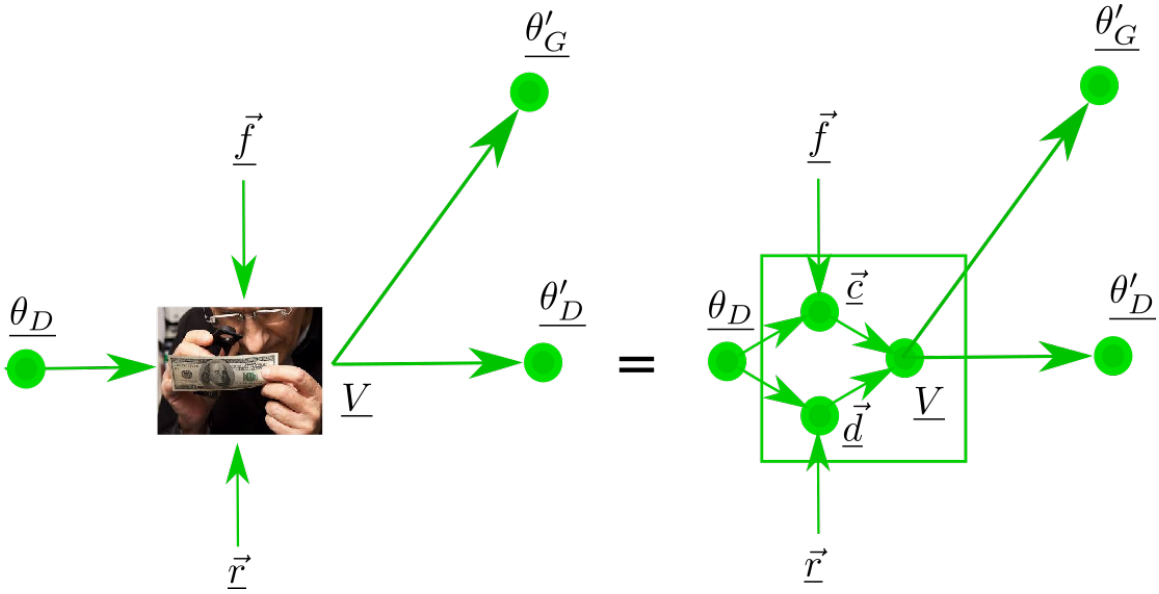


Figure 7.2: Discriminator node $\underline{V}$ in Fig.7.1 can be split into 3 nodes $\underline{\vec{c}}$, $\underline{\vec{d}}$ and $\underline{V}$.

$V(\theta_G, \theta_D)$ 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 $x \in \mathcal{D}$,

$$0 \leq D(x, \theta_D) \leq 1 \; . \tag{7.1}$$

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

$$\sum_{x \in S} D(x, \theta_D) = \lambda(S, \theta_D) \; . \tag{7.2}$$

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

$$\overline{D}(f, \theta_D) = 1 - D(f, \theta_D) \; . \tag{7.3}$$

Note that

$$0 \leq \overline{D}(f, \theta_D) \leq 1 \; . \tag{7.4}$$

Define:

$$V(\theta_G, \theta_D) = \sum_r P(r) \log D(r, \theta_D) + \sum_u P(u) \log \overline{D}(G(u, \theta_G), \theta_D) \; . \tag{7.5}$$

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

$$\delta V(\theta_G, \theta_D) = 0 \; . \tag{7.6}$$

This implies

$$\partial_{\theta_G} V(\theta_G, \theta_D) = \partial_{\theta_D} V(\theta_G, \theta_D) = 0 \tag{7.7}$$

and

$$V_{opt} = \min_{\theta_G} \max_{\theta_D} V(\theta_G, \theta_D) \ . \tag{7.8}$$

Node transition probability matrices for Figs.7.1 and 7.2 are given next in blue:

$$P(\theta_G) = \text{ given} \tag{7.9}$$

$$P(\theta_D) = \text{ given} \tag{7.10}$$

$$P(\vec{u}) = \prod_i P(u[i]) \ \text{ (usually uniform distribution)} \tag{7.11}$$

$$P(\vec{r}) = \prod_i P(r[i]) \tag{7.12}$$

$$P(f[i] \mid \vec{u}, \theta_G) = \delta[f[i], G(u[i], \theta_G)] \tag{7.13}$$

$$P(c[i] \mid \vec{f}, \theta_D) = \delta(c[i], \overline{D}(f[i], \theta_D)) \tag{7.14}$$

$$P(d[j] \mid \vec{r}, \theta_D) = \delta(d[j], D(r[j], \theta_D)) \tag{7.15}$$

$$P(V|\vec{d}, \vec{c}) = \delta(V, \frac{1}{N} \log \prod_{i,j} (c[i]d[j])) \tag{7.16}$$

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.

$$P(\theta'_G|V, \theta_G) = \delta(\theta'_G, \theta_G - \eta_G \partial_{\theta_G} V) \tag{7.17}$$

$$P(\theta'_D|V, \theta_D) = \delta(\theta'_D, \theta_D + \eta_D \partial_{\theta_D} V) \tag{7.18}$$

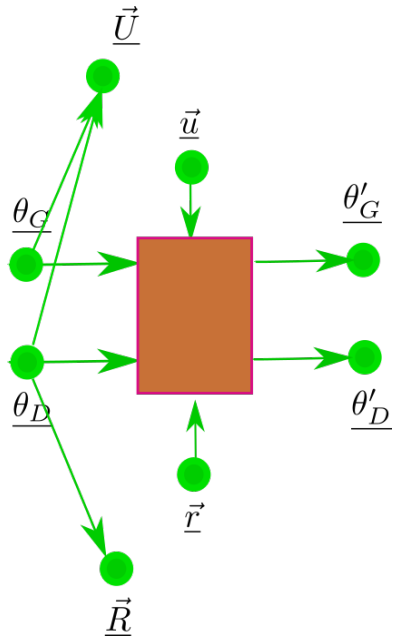


Figure 7.3: GAN, Constraining Bayesian Network

Constraining B net given in Fig.7.3. It adds 2 new nodes, namely $\underline{\vec{U}}$ and $\underline{\vec{R}}$, to the bnet of Fig.7.1. The purpose of these 2 barren (childrenless) nodes is to constrain certain functions to be probability distributions.

Node transition probabilities for the 2 new nodes given next in blue.

$$P(U[i] \mid \theta_G) = \frac{\overline{D}(G(U[i], \theta_G), \theta_D))}{\overline{\lambda}(\theta_G, \theta_D)} \tag{7.19}$$

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

$$P(R[i] \mid \theta_G, \theta_D) = \frac{D(R[i], \theta_D)}{\lambda(\theta_D)} \tag{7.20}$$

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

$$P(V|\vec{u}, \vec{r}) = \delta(V, \frac{1}{N} \log \prod_{i,j} (P(\underline{R[i]} = r[i] \mid \theta_G, \theta_D) P(\underline{U[i]} = u[j] \mid \theta_G))) \tag{7.21}$$

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

$\mathcal{L}$ = likelihood

$$\mathcal{L} = P(\vec{r}, \vec{u}|\theta_G, \theta_D) \tag{7.22}$$

$$= \prod_{i,j} \left[\frac{D(r[i], \theta_D)}{\lambda(\theta_D)} \frac{\overline{D}(G(u[j], \theta_G), \theta_D))}{\overline{\lambda}(\theta_G, \theta_D)} \right] \tag{7.23}$$

$$\log \mathcal{L} = N[V(\theta_G, \theta_D) - \log \lambda(\theta_D) - \log \overline{\lambda}(\theta_G, \theta_D)] \tag{7.24}$$

Chapter 8

Kalman Filter

A Kalman Filter is a special case of a Hidden Markov Model. HMMs are discussed in Chapter 6.

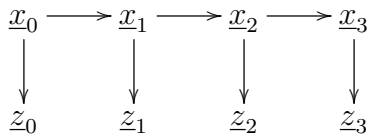


Figure 8.1: Kalman Filter bnet with $T = 4$.

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.8.1 has the following node probability transition matrices, printed in blue:

$$P(x_t|x_{t-1}) = \mathcal{N}(F_t x_{t-1} + B_t u_t, Q_t) \ , \tag{8.1}$$

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

$$P(z_t|x_t) = \mathcal{N}(H_t x_t, R_t) \ , \tag{8.2}$$

where H_t, R_t are given.
Define

$$\underline{Z}_t = (\underline{z}_{t'})_{t' \le t} \ . \tag{8.3}$$

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

$$P(x_t|Z_t) = \mathcal{N}(\hat{x}_t, P_t) \ . \tag{8.4}$$

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.8.2. For that figure,

$$P(\hat{x}_t, P_t | z_t, \hat{x}_{t-1}, P_{t-1}) = \delta(\hat{x}_t, ?)\delta(P_t, ?) \ . \tag{8.5}$$

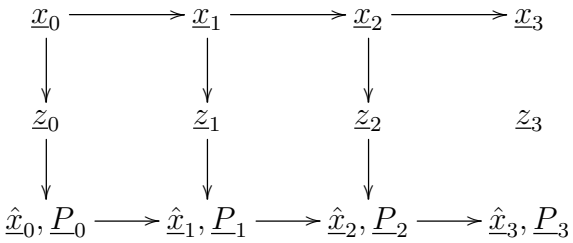


Figure 8.2: Kalman Filter bnet with deterministic nodes for $\hat{x}_t, P_t$.

Solution copied from Wikipedia Ref.[5]:

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

- **Predict**

  Predicted (a priori) state estimate

$$\hat{x}_{t|t-1} = F_t \hat{x}_{t-1|t-1} + B_t u_t \tag{8.6}$$

  Predicted (a priori) estimate covariance

$$P_{t|t-1} = F_t P_{t-1|t-1} F_t^{\mathsf{T}} + Q_t \tag{8.7}$$

- **Update**

  Innovation (or measurement pre-fit residual)

$$\tilde{y}_{t|t-1} = z_t - H_t \hat{x}_{t|t-1} \tag{8.8}$$

  Innovation (or pre-fit residual) covariance

$$S_t = H_t P_{t|t-1} H_t^{\mathsf{T}} + R_t \tag{8.9}$$

Optimal Kalman gain

$$K_t = P_{t|t-1} H_t^{\mathsf{T}} S_t^{-1} \tag{8.10}$$

Updated (a posteriori) state estimate

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t \tilde{y}_t \tag{8.11}$$

Updated (a posteriori) estimate covariance

$$P_{t|t} = (I - K_t H_t) P_{t|t-1} \tag{8.12}$$

Measurement post-fit residual

$$\tilde{y}_{t|t} = z_t - H_t \hat{x}_{t|t} \tag{8.13}$$

Chapter 9

Linear and Logistic Regression

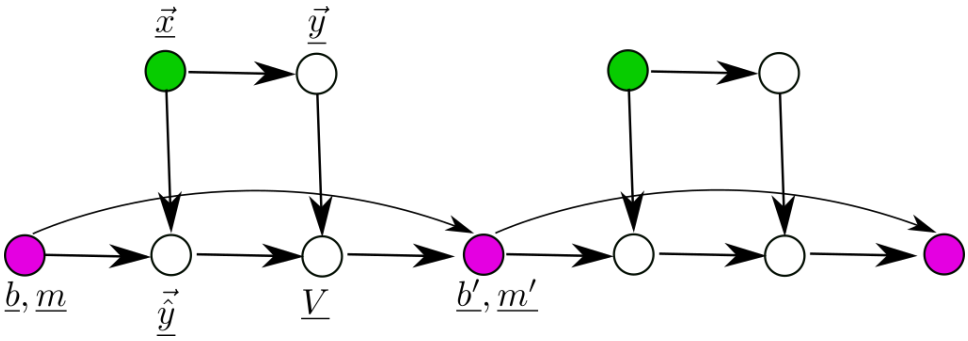


Figure 9.1: Linear Regression

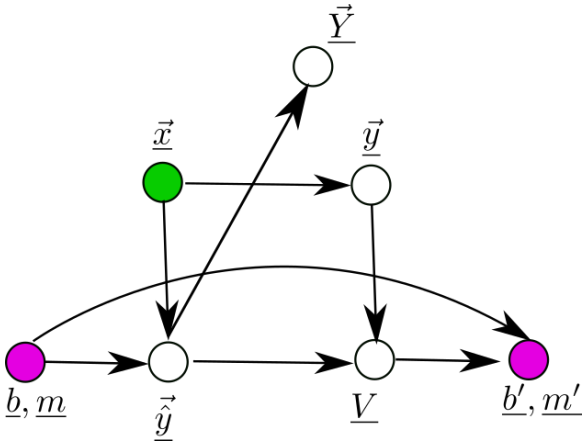


Figure 9.2: B net of Fig.9.1 with new $\vec{Y}$ node.

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.

$$\hat{y}(x; b, m) = b + mx \tag{9.1}$$

- **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.

$$\hat{y}(x; b, m) = \text{sig}(b + mx) \tag{9.2}$$

Define

$$V(b,m) = \sum_{x,y} P(x,y)|y - \hat{y}(x;b,m)|^2 \ . \tag{9.3}$$

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

Node transition probabilities of B net of Fig.9.1 given next in blue.

$$P(b,m) = \text{given} \tag{9.4}$$

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

Let

$$P_{\underline{x},\underline{y}}(x,y) = \frac{1}{nsam(\vec{x})} \sum_i \mathbb{1}(x = x[i], y = y[i]) \ . \tag{9.5}$$

$$P(\vec{x}) = \prod_i P(x[i]) \tag{9.6}$$

$$P(\vec{y}|\vec{x}) = \prod_i P(y[i] \mid x[i]) \tag{9.7}$$

$$P(\vec{\hat{y}}|\vec{x}, b, m) = \prod_i \delta(\hat{y}[i], \hat{y}(x[i], b, m)) \tag{9.8}$$

$$P(V|\vec{\hat{y}}, \vec{y}) = \delta(V, \frac{1}{nsam(\vec{x})} \sum_i |y[i] - \hat{y}[i]|^2) \tag{9.9}$$

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".

$$P(b'|V, b) = \delta(b', b - \eta_b \partial_b V) \tag{9.10}$$

$$P(m'|V, m) = \delta(m', m - \eta_m \partial_m V) \tag{9.11}$$

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:

$$x[i] = (x_0[i], x_1[i], x_2[i], \ldots x_{n-1}[i]) \ . \tag{9.12}$$

Slope m is replaced by weights

$$w = (w_0, w_1, w_3, , \ldots, w_{n-1}) \ , \tag{9.13}$$

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

$$p[i](1) = y[i], \quad p[i](0) = 1 - y[i] \tag{9.14}$$

$$\hat{p}[i](1) = \hat{y}[i], \quad \hat{p}[i](0) = 1 - \hat{y}[i] \tag{9.15}$$

Then for logistic regression, the following 2 cost functions $V(b, m)$ can be used as alternatives to the cost function Eq.(9.3) previously given.

$$V(b, m) = \frac{1}{nsam(\vec{x})} \sum_i D_{KL}(p[i] \parallel \hat{p}[i]) \tag{9.16}$$

and

$$\begin{aligned}
V(b, m) &= \frac{1}{nsam(\vec{x})} \sum_i CE(p[i] \to \hat{p}[i]) && (9.17)\\
&= \frac{-1}{nsam(\vec{x})} \sum_i \{y[i] \log \hat{y}[i] + (1 - y[i]) \log(1 - \hat{y}[i])\} && (9.18)\\
&= \frac{-1}{nsam(\vec{x})} \sum_i \log \left\{\hat{y}[i]^{y[i]} (1 - \hat{y}[i])^{(1-y[i])}\right\} && (9.19)\\
&= \frac{-1}{nsam(\vec{x})} \sum_i \log P(\underline{Y} = y[i] \mid \hat{y} = \hat{y}[i]) && (9.20)\\
&= -\sum_{x,y} P(x, y) \log P(\underline{Y} = y | \hat{y} = \hat{y}(x, b, m)) && (9.21)
\end{aligned}$$

Above, we used

$$P(\underline{Y} = Y | \hat{y}) = \hat{y}^Y [1 - \hat{y}]^{1-Y} \tag{9.22}$$

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

There is no node corresponding to $\underline{Y}$ in the B net of Fig.9.1. Fig.9.2 shows a new B net that has a new node called $\underline{\vec{Y}}$ compared to the B net of Fig.9.1. One defines the transition probabilities for all nodes of Fig.9.2 except $\underline{\vec{Y}}$ and $\underline{V}$ the same as for Fig.9.1. For $\underline{\vec{Y}}$ and $\underline{V}$, one defines

$$P(Y[i] \mid \vec{\hat{y}}) = P(\underline{Y} = Y[i] \mid \hat{y}[i]) \tag{9.23}$$

$$P(V|\vec{Y}, \vec{y}) = \delta(V, \frac{-1}{nsam(\vec{x})} \log \mathcal{L}) \ , \tag{9.24}$$

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

Chapter 10

Markov Blankets: COMING SOON

Chapter 11

Markov Chain Monte Carlo (MCMC): COMING SOON

Chapter 12

Message Passing (Belief Propagation): COMING SOON

Chapter 13

Monty Hall Problem

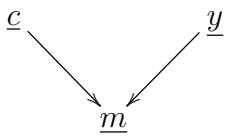


Figure 13.1: Monty Hall Problem.

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.13.1, 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:

$$P(c) = \frac{1}{3} \text{ for all } c \tag{13.1}$$

$$P(y) = \frac{1}{3} \text{ for all } y \tag{13.2}$$

$$P(m|c, y) = \mathbb{1}(m \neq c) \left[\frac{1}{2}\mathbb{1}(y = c) + \mathbb{1}(y \neq c)\mathbb{1}(m \neq y)\right] \tag{13.3}$$

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

$$P(c = 1|m = 2, y = 1) = \frac{1}{3} \tag{13.4}$$

$$P(c = 3|m = 2, y = 1) = \frac{2}{3} \tag{13.5}$$

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 14

Naive Bayes

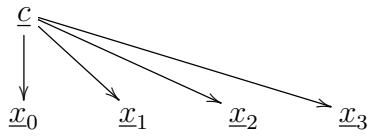


Figure 14.1: bnet for Naive Bayes with 4 features

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

$$x. = [x_0, x_1, \ldots, x_{F-1}] \; . \tag{14.1}$$

For the bnet of Fig.14.1,

$$P(c, x.) = P(c) \prod_{i=0}^{F-1} P(x_i|c) \; . \tag{14.2}$$

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

$$\begin{aligned} c^* &= \text{argmax}_c P(c|x.) && (14.3) \\ &= \text{argmax}_c \frac{P(c, x.)}{P(x.)} && (14.4) \\ &= \text{argmax}_c P(c, x.) \; . && (14.5) \end{aligned}$$

Chapter 15

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 transition probability matrix 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.

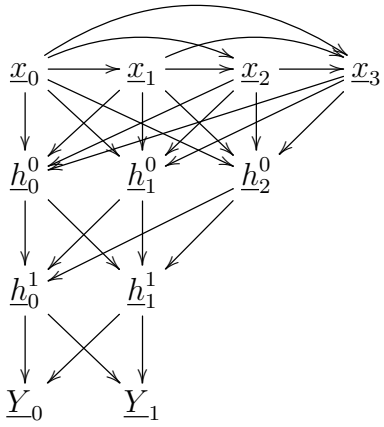


Figure 15.1: Neural Network (feed forward) with 4 layers: input layer $\underline{x}.$, 2 hidden layers $\underline{h}^0.$, $\underline{h}^1.$ and output layer $\underline{Y}$.

Consider Fig.15.1.

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

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

$\underline{Y}_i \in \mathbb{R}$ for $i = 0, 1, 2, \ldots, numy - 1$ 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 $i = 0, 1, \ldots, numy - 1$. $Y = \hat{y}$ 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.15.1 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 $i \in \mathbb{Z}_{[0,numh(\lambda))}$, $j \in \mathbb{Z}_{[0,numh(\lambda-1))}$, and $\lambda \in \mathbb{Z}_{[0,\Lambda)}$.

These are the transition probability matrices, printed in blue, for the nodes of the bnet Fig.15.1:

$$P(x_i \mid x_{i-1}, x_{i-1}, \ldots, x_0) = \text{given} \tag{15.1}$$

$$P(h^{\lambda}_i \mid h^{\lambda-1}_{\cdot}) = \delta\left(h^{\lambda}_i, \mathcal{A}^{\lambda}_i(\sum_j w^{\lambda-1}_{i|j} h^{\lambda-1}_j + b^{\lambda-1}_i)\right) \tag{15.2}$$

$$P(Y_i \mid h^{\Lambda-1}_{\cdot}) = \delta\left(Y_i, \mathcal{A}^{\Lambda}_i(\sum_j w^{\Lambda-1}_{i|j} h^{\Lambda-1}_j + b^{\Lambda-1}_i)\right) \tag{15.3}$$

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

Activation functions must be nonlinear.

- **Step function (Perceptron)**

$$\mathcal{A}(x) = \mathbb{1}(x > 0) \tag{15.4}$$

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

- **Sigmoid function**

$$\mathcal{A}(x) = \frac{1}{1+e^{-x}} = \text{sig}(x) \tag{15.5}$$

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

$$\begin{aligned} \text{sig}(x) + \text{sig}(-x) &= \frac{1}{1+e^{-x}} + \frac{1}{1+e^{x}} && (15.6)\\ &= \frac{2+e^{x}+e^{-x}}{2+e^{x}+e^{-x}} && (15.7)\\ &= 1 && (15.8) \end{aligned}$$

- **Hyperbolic tangent**

$$\mathcal{A}(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{15.9}$$

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

Odd function:

$$\tanh(-x) = -\tanh(x) \tag{15.10}$$

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

- **ReLU (Rectified Linear Unit)**

$$\mathcal{A}(x) = x\mathbb{1}(x > 0) = \max(0, x) \; . \tag{15.11}$$

Compare this to the step function.

- **Swish**

$$\mathcal{A}(x) = x \; \text{sig}(x) \tag{15.12}$$

- **Softmax**

$$\mathcal{A}(x_i|x.) = \frac{e^{x_i}}{\sum_i e^{x_i}} \tag{15.13}$$

It's called softmax because if we approximate the exponentials, both in the numerator and denominator of Eq.(15.13), by the largest one, we get

$$\mathcal{A}(x_i|x.) \approx \mathbb{1}(x_i = \max_k x_k) \; . \tag{15.14}$$

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,

$$\begin{aligned} \mathcal{A}(x_0|x.) &= \frac{e^{x_0}}{e^{x_0} + e^{x_1}} & (15.15) \\ &= \text{sig}(x_0 - x_1) \; , & (15.16) \end{aligned}$$

$$\mathcal{A}(x_1|x.) = \text{sig}(x_1 - x_0) \; . \tag{15.17}$$

Weight optimization via supervised training and gradient descent

The bnet of Fig.15.1 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.15.2.

In Fig.15.2, the nodes in Fig.15.1 become sampling space vectors. For example, $\underline{x}.$ becomes $\underline{\vec{x}}.$, where the components of $\underline{\vec{x}}.$ in sampling space are $\underline{x}.[s] \in \{0,1\}^{numx}$ 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.15.2. 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

$$W^{\lambda}_{i|j} = [w^{\lambda}_{i|j}, b^{\lambda}_{i}] \; . \tag{15.18}$$

These are the transition probability matrices, printed in blue, for the nodes of the bnet Fig.15.2:

$$P(x.[s]) = \text{given} \; . \tag{15.19}$$

$$P(y.[s] \mid x.[s]) = \text{given} \; . \tag{15.20}$$

$$P(h^{\lambda}_{i}[s] \mid h^{\lambda-1}_{\cdot}[s]) = \delta\left(h^{\lambda}_{i}[s], \mathcal{A}^{\lambda}_{i}(\sum_{j} w^{\lambda-1}_{i|j} h^{\lambda-1}_{j}[s] + b^{\lambda-1}_{i})\right) \tag{15.21}$$

$$P(Y_{i}[s] \mid h^{\Lambda-1}_{\cdot}[s]) = \delta\left(Y_{i}[s], \mathcal{A}^{\Lambda}_{i}(\sum_{j} w^{\Lambda-1}_{i|j} h^{\Lambda-1}_{j}[s] + b^{\Lambda-1}_{i})\right) \tag{15.22}$$

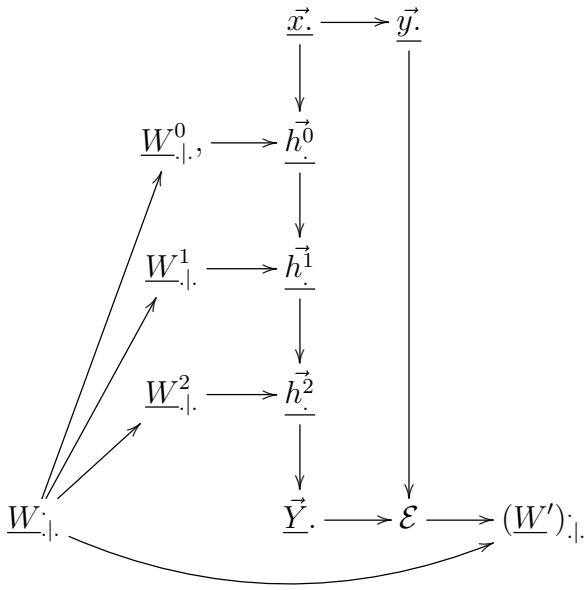


Figure 15.2: bnet for finding optimum weights of the bnet Fig.15.1 via supervised training and gradient descent.

$$P(W^{\cdot}_{\cdot|\cdot}) = \text{given} \tag{15.23}$$

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

$$P(W^{\lambda}_{\cdot|\cdot}|W^{\cdot}_{\cdot|\cdot}) = \delta(W^{\lambda}_{\cdot|\cdot}, (W^{\cdot}_{\cdot|\cdot})^{\lambda}) \tag{15.24}$$

$$P(\mathcal{E}|\vec{y}_{\cdot}, \vec{Y}_{\cdot}) = \frac{1}{nsam(\vec{x})} \sum_s \sum_i d(y_i[s], Y_i[s]) \ , \tag{15.25}$$

where

$$d(y, Y) = |y - Y|^2 \ . \tag{15.26}$$

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

$$d(y, Y) = XE(y \to Y) = -y \log Y - (1 - y) \log(1 - Y) \ . \tag{15.27}$$

$$P((W')^{\lambda}_{i|j}|\mathcal{E}, W^{\cdot}_{\cdot|\cdot}) = \delta((W')^{\lambda}_{i|j}, W^{\lambda}_{i|j} - \eta \partial_{W^{\lambda}_{i|j}} \mathcal{E}) \tag{15.28}$$

$\eta > 0$ 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 transition probability matrix for a non-dense layer is of the form:

$$P(h_i^\lambda[s] \mid h_\cdot^{\lambda-1}[s]) = \delta(h_i^\lambda[s], H_i^\lambda[s]) \ , \tag{15.29}$$

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.[6]. 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

  $$P(r_i^\lambda = r) = [p]^r[1-p]^{1-r} \text{ (Bernouilli dist.)} \ . \tag{15.30}$$

  Now one has

  $$P(h_i^\lambda[s] \mid h_\cdot^{\lambda-1}[s], r_i^\lambda) = \delta(h_i^\lambda[s], H_i^\lambda[s]) \ , \tag{15.31}$$

  where

  $$H_i^\lambda[s] = \mathcal{A}_i^\lambda(r_i^\lambda \sum_j w_{i|j}^\lambda h_j^{\lambda-1}[s] + b_i^\lambda) \ . \tag{15.32}$$

  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.15.2, not in the classification bnet Fig.15.1. We prefer to remove the $\underline{r}_i^\lambda$ stochasticity from classification and for Fig.15.1 to act as an average over sampling space of Fig.15.2. Therefore, if weights $w_{i|j}^\lambda$ are obtained for a dropout layer λ in Fig.15.2, then that layer is used in Fig.15.1 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 $\mathcal{F} : \{0, 1, \ldots, numf - 1\} \to \mathbb{R}$.
    σ=stride length

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

$$H_i^\lambda[s] = \sum_{j=0}^{numf-1} h_{j+i\sigma}^{\lambda-1}[s]\mathcal{F}(j) \ . \tag{15.33}$$

For the indices not to go out of bounds in Eq.(15.33), we must have

$$numh(\lambda - 1) - 1 = numf - 1 + (numh(\lambda) - 1)\sigma \tag{15.34}$$

so

$$numh(\lambda) = \frac{1}{\sigma}[numh(\lambda - 1) - numf] + 1 \ . \tag{15.35}$$

- 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 $\{0, , 1, \ldots, numh(\lambda - 1) - 1\}$ into $numh(\lambda)$ mutually disjoint, nonempty sets called $Pool(i)$, where $i \in \{0, 1, \ldots, numh(\lambda) - 1\}$.

  - AvgPool
  $$H_i^\lambda[s] = \frac{1}{size(Pool(i))} \sum_{j \in Pool(i)} h_j^{\lambda-1}[s] \tag{15.36}$$

  - MaxPool
  $$H_i^\lambda[s] = \max_{j \in Pool(i)} h_j^{\lambda-1}[s] \tag{15.37}$$

Autoencoder NN

If the sequence

$$numx, numh(0), numh(1), \ldots, numh(\Lambda - 1), numy \tag{15.38}$$

first decreases monotonically up to layer λ_{min}, then increases monotonically until $numy = numx$, 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 16

Non-negative Matrix Factorization

Based on Ref.[7].

Given matrix V, factor it into product of two matrices

$$V = WH \ , \tag{16.1}$$

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 16.1: B net interpretation of non-negative matrix factorization.

Node transition matrices, printed in blue, for Fig.16.1.

$$P(\underline{v} = w|a) = \frac{V_{w,a}}{\sum_w V_{w,a}} \tag{16.2}$$

$$P(w|h) = \frac{W_{w,h}}{\sum_w W_{w,h}} \tag{16.3}$$

$$P(h|a) = \frac{\sum_w W_{w,h}}{\sum_w V_{w,a}} H_{h,a} \tag{16.4}$$

Simplest recursive algorithm:

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

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

$$H_{i,j}^{(n+1)} \leftarrow H_{i,j}^{(n)} \frac{[(W^{(n)})^T V]_{i,j}}{[(W^{(n)})^T \underbrace{W^{(n)} H^{(n)}}_{\approx V}]_{i,j}} \tag{16.5}$$

and

$$W_{i,j}^{(n+1)} \leftarrow W_{i,j}^{(n)} \frac{[V(H^{(n+1)})^T]_{i,j}}{[\underbrace{W^{(n)} H^{(n+1)}}_{\approx V}(H^{(n+1)})^T]_{i,j}} \, . \tag{16.6}$$

After each step, record error defined by

$$\mathcal{E}^{(n)} = \| \, V - W^{(n)} H^{(n)} \, \|_2 \ \ . \tag{16.7}$$

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

Can also use Kullback-Lieber divergence for error:

$$\mathcal{E} = \sum_a P(a) D_{KL}(P(\underline{v} = w|a) \parallel \sum_h P(w|h) P(h|a)) \, , \tag{16.8}$$

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

Chapter 17

Recurrent Neural Networks

This chapter is mostly based on Ref.[8].

This chapter assumes you are familiar with the material and notation of Chapter 15 on plain Neural Nets.

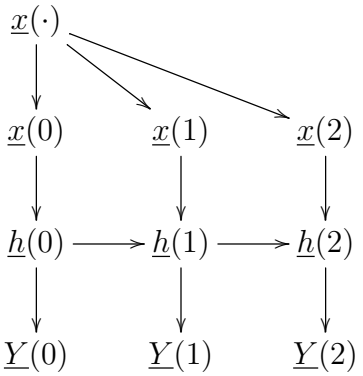


Figure 17.1: Simple example of RNN witb $T = 3$

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.17.1 with arbitrary T. The node transition matrices, printed in blue, for this bnet, are as follows.

$$P(x(\cdot)) = \text{given} \tag{17.1}$$

$$P(x(t)) = \delta(x(t), [x(\cdot)]_t) \tag{17.2}$$

$$P(h(t) \mid h(t-1), x(t)) = \delta(h(t), \mathcal{A}(W^{h|x} x(t) + W^{h|h} h(t-1) + b^h)) \ , \tag{17.3}$$

where $h(-1) = 0$.

$$P(Y(t) \mid h(t)) = \delta(Y(t), \mathcal{A}(W^{y|h} h(t) + b^y)) \tag{17.4}$$

Define

$$W^h = [W^{h|x}, W^{h|h}, b^h] \ , \tag{17.5}$$

and

$$W^y = [W^{y|h}, b^y] \ . \tag{17.6}$$

The bnet of Fig.17.1 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.17.2.

Let $s = 0, 1, \ldots, nsam(\vec{x}) - 1$ be the labels for a minibatch of samples. The node transition matrices, printed in blue, for bnet Fig.17.2, are as follows.

$$P(x(\cdot)[s]) = \text{given} \tag{17.7}$$

$$P(x(t)[s]) = \delta(x(t)[s], [x(\cdot)]_t[s]) \tag{17.8}$$

$$P(h(t)[s] \mid h(t-1)[s], x(t)[s]) = \delta(h(t)[s], \mathcal{A}(W^{h|x} x(t)[s] + W^{h|h} h(t-1)[s] + b^h) \tag{17.9}$$

$$P(Y(t)[s] \mid h(t-1)[s]) = \delta(Y(t)[s], \mathcal{A}(W^{y|h} h(t-1)[s] + b^y) \tag{17.10}$$

$$P(y(\cdot)[s] \mid x(\cdot)[s]) = \text{given} \tag{17.11}$$

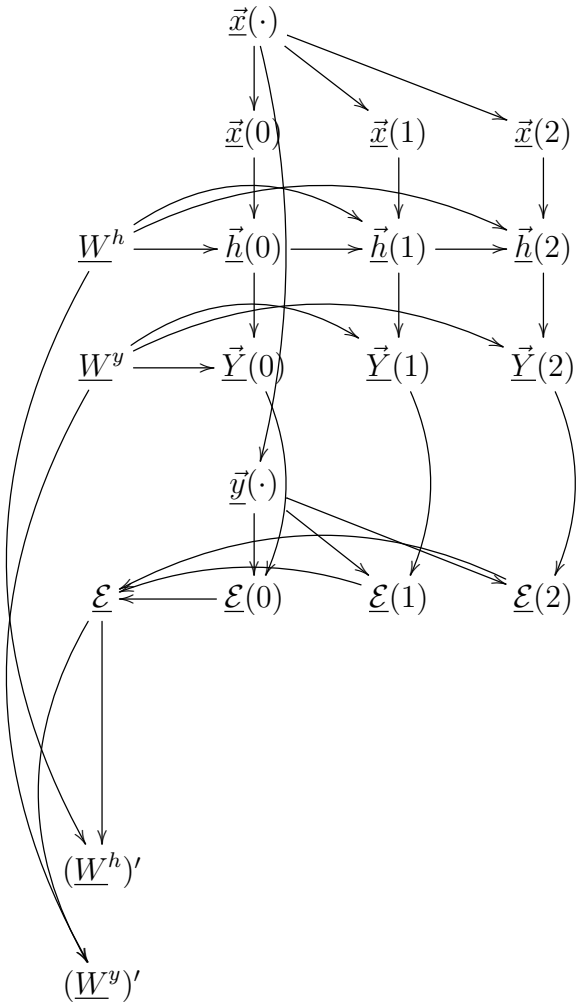


Figure 17.2: RNN bnet used to optimize parameters W^h and W^y of RNN bnet Fig.17.1.

$$P(\mathcal{E}(t) \mid \vec{y}(\cdot), \vec{Y}(t)) = \frac{1}{nsam(\vec{x})} \sum_s d(y(t)[s], Y(t)[s]) \ , \tag{17.12}$$

where

$$d(y, Y) = |y - Y|^2 \ . \tag{17.13}$$

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

$$d(y, Y) = XE(y \to Y) = -y \log Y - (1 - y) \log(1 - Y) \ . \tag{17.14}$$

$$P(\mathcal{E} \mid [\mathcal{E}(t)]_{\forall t}) = \delta(\mathcal{E}, \sum_t \mathcal{E}(t)) \tag{17.15}$$

For $a = h, y$,

$$P(W^a) = \text{given} \ . \tag{17.16}$$

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

$$P((W^a)'|\mathcal{E}, W^a) = \delta((W^a)', W^a - \eta^a \partial_{W^a} \mathcal{E}) \ . \tag{17.17}$$

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

Language Sequence Modeling

Figs.17.1, and 17.2 with arbirary T can be used as follows to do Language Sequence Modeling.

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

$$P(y(\cdot)[s] \mid x(\cdot)[s]) = \prod_t \mathbb{1}(\quad y(t)[s] = P(x(t)[s] \mid [x(t')[s]]_{t'<t}) \quad) \tag{17.18}$$

With such training, one gets

$$P(Y(t)|h(t)) = \mathbb{1}(\quad Y(t) = P(x(t) \mid [x(t')]_{t'<t}) \quad) \ . \tag{17.19}$$

Therefore,

$$Y(0) = P(x(0)) \ , \tag{17.20}$$

$$Y(1) = P(x(1)|x(0)) \ , \tag{17.21}$$

$$Y(2) = P(x(2)|x(0), x(1)) \ , \tag{17.22}$$

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

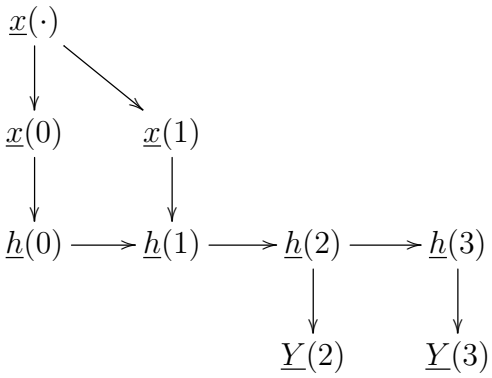


Figure 17.3: RNN bnet of the many to many kind. This one can be used for translation. $x(0)$ and $x(1)$ might denote two words of an English sentence, and $Y(2)$ and $Y(3)$ might be their Italian translation.

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.[9]. In this section, $\odot$ will denote the Hadamard matrix product (elementwise product).

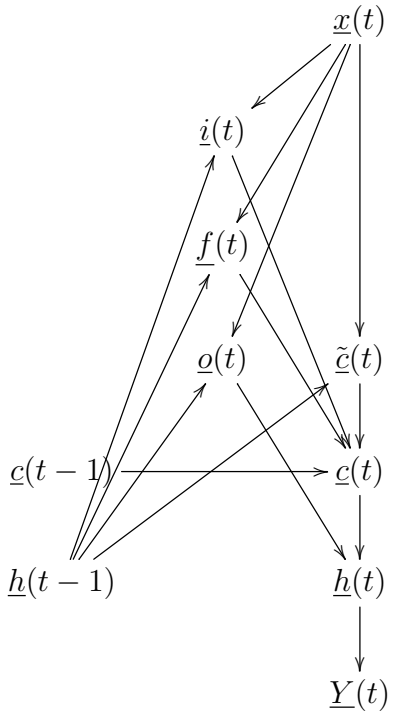


Figure 17.4: bnet for a Long Short Term Memory (LSTM) unit.

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.17.4 is a bnet net for a LSTM unit. The node transition matrices, printed in blue, for this bnet, are as follows.

$$P(f(t)|x(t),h(t-1)) = \mathbb{1}(\quad f(t) = \text{sig}(W^{f|x}x(t) + U^{f|h}h(t-1) + b^f) \quad)\,, \tag{17.23}$$

where $h(-1) = 0$.

$$P(i(t)|x(t),h(t-1)) = \mathbb{1}(\quad i(t) = \text{sig}(W^{i|x}x(t) + U^{i|h}h(t-1) + b^i) \quad) \tag{17.24}$$

$$P(o(t)|x(t),h(t-1)) = \mathbb{1}(\quad o(t) = \text{sig}(W^{o|x}x(t) + U^{o|h}h(t-1) + b^o) \quad) \tag{17.25}$$

$$P(\tilde{c}(t)|x(t),h(t-1)) = \mathbb{1}(\quad \tilde{c}(t) = \tanh(W^{c|x}x(t) + U^{c|h}h(t-1) + b^c) \quad) \tag{17.26}$$

$$P(c(t)|f(t),c(t-1),i(t),\tilde{c}(t)) = \mathbb{1}(\quad c(t) = f(t) \odot c(t-1) + i(t) \odot \tilde{c}(t) \quad) \tag{17.27}$$

$$P(h(t)|o(t),c(t)) = \mathbb{1}(\quad h(t) = o(t) \odot \tanh(c(t)) \quad) \tag{17.28}$$

$$P(Y(t)|h(t)) = \mathbb{1}(\quad Y(t) = \mathcal{A}(\mathcal{W}^{y|h}h(t) + b^y) \quad) \tag{17.29}$$

Gated Recurrence Unit (GRU) (2014)

This section is based on Wikipedia article Ref.[10]. 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.

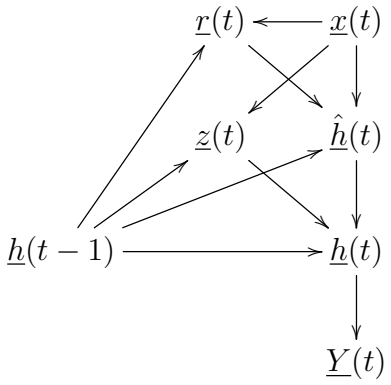


Figure 17.5: bnet for a Gated Recurrent Unit (GRU).

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.17.5 is a bnet net for a GRU. The node transition matrices, printed in blue, for this bnet, are as follows.

$$P(z(t)|x(t), h(t-1)) = \mathbb{1}(\quad z(t) = \text{sig}(W^{z|x}x(t) + U^{z|h}h(t-1) + b^z) \quad) \,, \tag{17.30}$$

where $h(-1) = 0$.

$$P(r(t)|x(t), h(t-1)) = \mathbb{1}(\quad r(t) = \text{sig}(W^{r|x}x(t) + U^{r|h}h(t-1) + b^r) \quad) \tag{17.31}$$

$$P(\hat{h}(t)|x(t), r(t), h(t-1)) = \mathbb{1}(\quad \hat{h}(t) = \tanh(W^{h|x}x(t) + U^{h|h}(r(t) \odot h(t-1)) + b^h) \quad) \tag{17.32}$$

$$P(h(t)|z(t), h(t-1), \hat{h}(t)) = \mathbb{1}(\quad h(t) = (1 - z(t)) \odot h(t-1) + z(t) \odot \hat{h}(t) \quad) \tag{17.33}$$

$$P(Y(t)|h(t)) = \mathbb{1}(\quad Y(t) = \mathcal{A}(\mathcal{W}^{y|h} h(t) + b^y) \quad) \tag{17.34}$$

Chapter 18

Reinforcement Learning (RL)

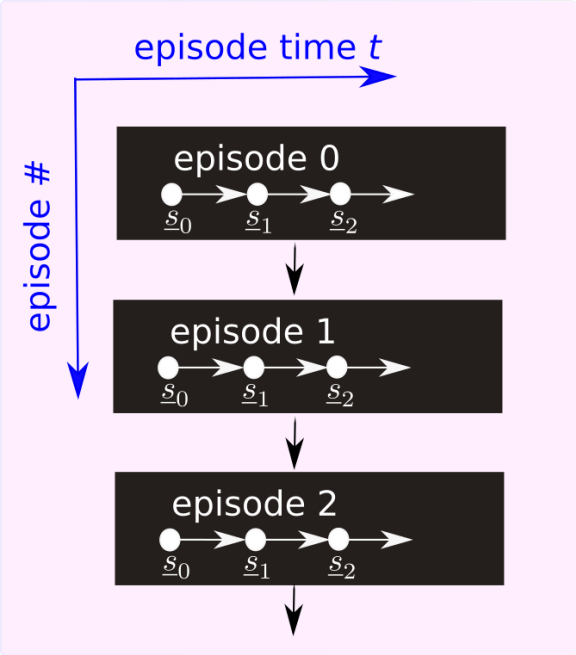


Figure 18.1: Axes for episode time and episode number.

I based this chapter on the following references. Refs.[11][12]

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.18.1.

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.

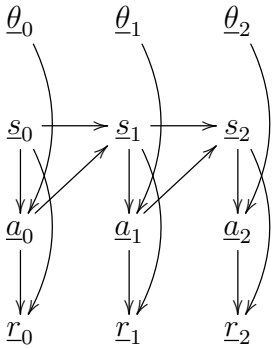


Figure 18.2: State-Action-Reward dynamical bnet

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

$$\underline{X}_\cdot = [\underline{X}_0, \underline{X}_1, \ldots, \underline{X}_{T-1}] \ . \tag{18.1}$$

Also let

$$\underline{X}_{\geq t} = [\underline{X}_t, \underline{X}_{t+1}, \ldots, \underline{X}_{T-1}] \ . \tag{18.2}$$

Fig.18.2 shows the basic State-Action-Reward bnet for an agent that is learning. The transition probabilities for the nodes of Fig.18.2 are given in blue below:

$$P(a_t|s_t, \theta_t) = \text{given.} \tag{18.3}$$

$P(a_t|s_t, \theta_t)$ is called a **policy with parameter θ_t.**

$$P(s_t|s_{t-1}, a_{t-1}) = \text{given.} \tag{18.4}$$

$P(s_t|s_{t-1}, a_{t-1})$ is called the **transition matrix of the model**. $P(s_t|s_{t-1}, a_{t-1})$ reduces to $P(s_0)$ when $t = 0$.

$$P(r_t|s_t, a_t) = \delta(r_t, r(s_t, a_t))) \ . \tag{18.5}$$

$r : S_{\underline{s}} \times S_{\underline{a}} \to \mathbb{R}$ is a given **one-time reward function**.

Note that

$$P(s_\cdot, a_\cdot|\theta_\cdot) = \prod_{t=0}^{T-1} \{P(s_t|s_{t-1}, a_{t-1}) P(a_t|s_t, \theta_t)\} \ . \tag{18.6}$$

Define the **all times reward** Σ by

$$\Sigma(s_\cdot, a_\cdot) = \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t) \ . \tag{18.7}$$

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

$$E\Sigma(\theta.) = E_{\underline{s}.,\underline{a}.|\theta.}\Sigma(\underline{s}., \underline{a}.) = \sum_{s.,a.} P(s., a.|\theta.)\Sigma(s., a.) \tag{18.8}$$

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:

$$\theta.^* = \text{argmax}_{\theta.} E\Sigma(\theta.) \tag{18.9}$$

Define a **future reward** for times $\geq t$ as:

$$\Sigma_{\geq t}((s_{t'}, a_{t'})_{t' \geq t}) = \sum_{t'=t}^{T-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) \tag{18.10}$$

Define the following **expected conditional future rewards** (rewards for times $\geq t$, conditioned on certain quantities having given values):

$$\begin{aligned} v_t &= v(s_t, a_t; \theta.) = E_{\underline{s}.,\underline{a}.|s_t,a_t,\theta.}[\Sigma_{\geq t}] && (18.11) \\ V_t &= V(s_t; \theta.) = E_{\underline{s}.,\underline{a}.|s_t,\theta.}[\Sigma_{\geq t}] = E_{\underline{a}_t|s_t,\theta.}[v(s_t, \underline{a}_t; \theta.)] && (18.12) \end{aligned}$$

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:

$$E\Sigma(\theta.) = E_{\underline{s}_0,\underline{a}_0|\theta_0} v(\underline{s}_0, \underline{a}_0; \theta.) \tag{18.13}$$

Define a **one-time reward** and an **expected conditional one-time reward** as:

$$\begin{aligned} r_t &= r(s_t, a_t) && (18.14) \\ R_t &= R(s_t; \theta_t) = E_{\underline{a}_t|s_t,\theta_t}[r(s_t, \underline{a}_t)] \ . && (18.15) \end{aligned}$$

Note that

$$\begin{aligned} \Sigma_{\geq t} &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{T-1-t} r_{t+(T-1-t)} && (18.16) \\ &= r_t + \gamma \Sigma_{\geq t+1}; . && (18.17) \end{aligned}$$

If we take $E_{\underline{s}.,\underline{a}.|s_t,a_t,\theta.}[\cdot]$ of both sides of Eq.(18.17), we get

$$v_t = r_t + \gamma E_{\underline{s}_{t+1},\underline{a}_{t+1}|\theta.}[v_{t+1}] \ . \tag{18.18}$$

If we take $E_{\underline{s}_\cdot,\underline{a}_\cdot|s_t,\theta_\cdot}[\cdot]$ of both sides of Eq.(18.17), we get

$$V_t = R_t + \gamma E_{\underline{s}_{t+1}|\theta_\cdot}[V_{t+1}] \; . \tag{18.19}$$

Note that

$$\begin{aligned} \Delta r_t &= r_t - R_t && (18.20)\\ &= r_t - (V_t - \gamma E_{\underline{s}_{t+1}|\theta_\cdot}[V_{t+1}]) && (18.21)\\ &= r_t + \gamma E_{\underline{s}_{t+1}|\theta_\cdot}[V_{t+1}] - V_t \; . && (18.22) \end{aligned}$$

Define

$$\Delta v_t = v_t - V_t \; . \tag{18.23}$$

Note that

$$\Delta v_t = \Delta r_t \; . \tag{18.24}$$

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.18.3.

Fig.18.3 is the same as Fig.18.2 but with more nodes added in order to optimize the policy parameters. Here are the transition matrices, in blue, for the nodes not already discussed in connection to Fig.18.2.

$$P(\theta_t|\theta_\cdot) = \delta(\theta_t, (\theta_\cdot)_t) \tag{18.25}$$

$$\forall (s_t, a_t): \quad P(v_t(s_t,a_t)|r_t, v_{t+1}(\cdot), \theta_\cdot) = \delta(v_t(s_t,a_t), r_t + \gamma E_{\underline{s}_{t+1},\underline{a}_{t+1}|\theta_\cdot}[v_{t+1}]) \tag{18.26}$$

$$P(\theta_\cdot'|\theta_\cdot, v_0(\cdot)) = \delta(\theta_\cdot', \theta_\cdot + \alpha \partial_{\theta_\cdot} \underbrace{E_{\underline{s}_0,\underline{a}_0|\theta_0} v(\underline{s}_0, \underline{a}_0; \theta_\cdot)}_{E\Sigma(\theta_\cdot)}) \tag{18.27}$$

$\alpha > 0$ is called the **learning rate**. This method of improving $\theta_\cdot$ is called gradient ascent.

Concerning the gradient of the objective function, note that

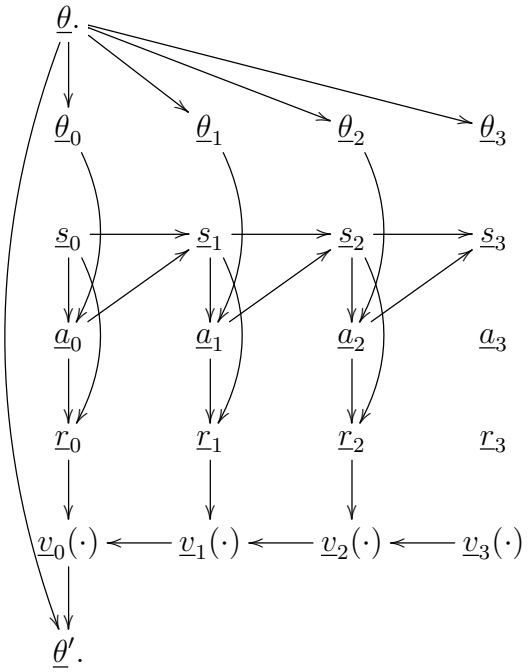


Figure 18.3: Exact RL bnet. $v_t(\cdot)$ means the array $[v_t(s_t, a_t)]_{\forall s_t, a_t}$ The following arrows are implicit: for all t, arrow from $\underline{\theta}. \to \underline{v}_t(\cdot)$. We did not draw those arrows so as not to clutter the diagram.

$$\partial_{\theta_t} E\Sigma(\theta.) = \sum_{s.,a.} \partial_{\theta_t} P(s.,a.|\theta.)\Sigma(s.,a.) \tag{18.28}$$

$$= \sum_{s.,a.} P(s.,a.|\theta.)\partial_{\theta_t} \log P(s.,a.|\theta.)\Sigma(s.,a.) \tag{18.29}$$

$$= E_{\underline{s}.,\underline{a}.|\theta.} \left\{\partial_{\theta_t} \log P(a_t|s_t,\theta_t)\Sigma(s.,a.)\right\} \ . \tag{18.30}$$

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:

$$\partial_{\theta_t} E\Sigma(\theta.) \approx \frac{1}{nsam(\vec{s}_t)} \sum_i \sum_{t=0}^{T-1} \partial_{\theta_t} \log P(a_t[i] \mid s_t[i],\theta_t) r(s_t[i],a_t[i]) \ . \tag{18.31}$$

The exact RL bnet Fig.18.3 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.(18.31) by

$$\partial_{\theta_t} E\Sigma(\theta.) \approx \frac{1}{nsam(\vec{s})} \sum_i \sum_{t=0}^{T-1} \underbrace{\partial_{\theta_t} \log P(a_t[i] \mid s_t[i],\theta_t)}_{Actor} \underbrace{\Delta r_t(s_t[i],a_t[i])}_{Critic} \tag{18.32}$$

The actor-critic RL bnet is given by Fig.18.4. This bnet is approximate and assumes that the policy is known. The transition matrices for its nodes are given in blue below.

$$P(\theta_t) = \text{given} \tag{18.33}$$

$$P(s_t[i] \mid s_{t-1}[i], a_{t-1}[i]) = \text{ given} \tag{18.34}$$

$$P(a_t[i] \mid s_t[i],\theta_t) = \text{ given} \tag{18.35}$$

$$P(r_t[i] \mid s_t[i], a_t[i]) = \delta(r_t[i], r(s_t[i],a_t[i])) \tag{18.36}$$

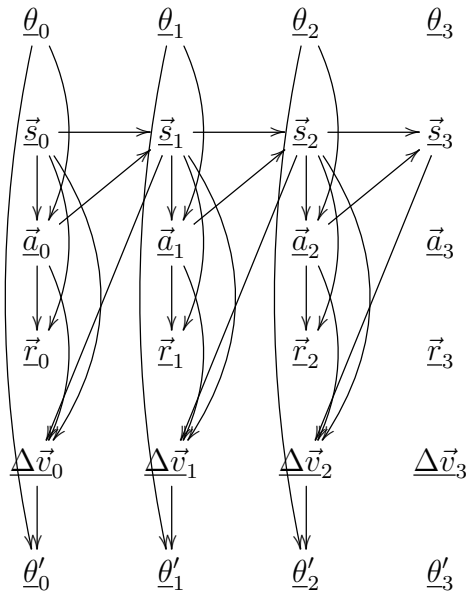


Figure 18.4: Actor-Critic RL bnet.

$r : S_{\underline{s}} \times S_{\underline{a}} \rightarrow \mathbb{R}$ is given.

$$P(\Delta v_t[i] \mid s_t[i], a_t[i], s_{t+1}[i]) = \delta(\Delta v_t[i], r(s_t[i], a_t[i]) + \gamma \hat{V}(s_{t+1}[i]; \phi') - \hat{V}(s_t[i]); \phi) \; . \tag{18.37}$$

$$P(\theta'.) = \delta(\theta'., \theta_t + \alpha \partial_{\theta_t} \sum_i \log P(a_t[i] \mid s_t[i], \theta_t) \Delta v_t[i]) \tag{18.38}$$

$\hat{V}(s_t[i]); \phi)$ is obtained by curve fitting (see Chapter 2) using samples $(s_t[i], a_t[i])$ $\forall t, i$ with

$$y[i] = \sum_{t'=t}^{T} r(s_{t'}[i], a_{t'}[i]) \tag{18.39}$$

and

$$\hat{y}[i] = \hat{V}(s_t[i]; \phi) \; . \tag{18.40}$$

Eq.(18.39) 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'.

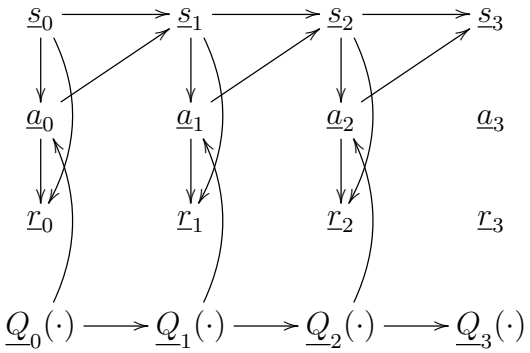


Figure 18.5: Q function learning RL bnet.

Q function learning RL bnet

The Q-function learning RL bnet is given by Fig.18.5. This bnet is approximate and assumes that the policy is NOT known. The transition matrices for its nodes are given in blue below. (Remember that $Q = v$).

$$P(s_t|s_{t-1}, a_{t-1}) = \text{ given} \tag{18.41}$$

$$P(a_t|s_t, v_t(\cdot)) = \delta(a_t, \text{argmax}_a v_t(s_t, a)) \tag{18.42}$$

$$P(r_t|s_t, a_t) = \delta(r_t, r(s_t, a_t)) \tag{18.43}$$

$r : S_{\underline{s}} \times S_{\underline{a}} \to \mathbb{R}$ is given.

$$\begin{aligned} \forall (s_t, a_t): \quad & P(v_t(s_t, a_t)|v_{t-1}(\cdot)) = \\ & = \delta(v_t(s_t, a_t), r(s_t, a_t) + \gamma \text{max}_a E_{\underline{s}_{t+1}|s_t, a_t} v_{t-1}(\underline{s}_{t+1}, a)) \end{aligned} \tag{18.44}$$

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.18.6) instead of Fig.18.5 and replace Eq.(18.44) by

$$\begin{aligned} \forall (s_t, a_t): \quad & P(v_t(s_t, a_t)|s_{t+1}, v_{t-1}(\cdot)) = \\ & = \delta(v_t(s_t, a_t), r(s_t, a_t) + \gamma \text{max}_a v_{t-1}(s_{t+1}, a)) \ . \end{aligned} \tag{18.45}$$

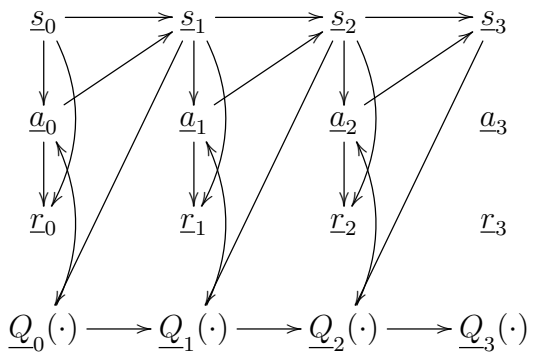


Figure 18.6: Q function learning RL bnet. Same as Fig.18.5 but with new arrow passing s_t to Q_{t-1}.

Chapter 19

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:

$$E(v,h) = -(b^T v + a^T h + v^T W^{v|h} h) \tag{19.1}$$

Boltzmann distribution:

$$P(v,h) = \frac{e^{-E(v,h)}}{Z} \tag{19.2}$$

Partition function:

$$Z = \sum_{v,h} e^{-E(v,h)} = Z(a,b,W^{v|h}) \tag{19.3}$$

$$\begin{aligned} P(v|h) &= \frac{e^{b^T v + a^T h + v^T W^{v|h} h}}{\sum_v e^{b^T v + a^T h + v^T W^{v|h} h}} && (19.4)\\ &= \frac{e^{b^T v + v^T W^{v|h} h}}{\sum_v e^{b^T v + v^T W^{v|h} h}} && (19.5)\\ &= \prod_i \frac{e^{v_i (b_i + \sum_j W^{v|h}_{i,j} h_j)}}{\sum_{v_i=0,1} e^{v_i (b_i + \sum_j W^{v|h}_{i,j} h_j)}} && (19.6)\\ &= \prod_i P(v_i|h) && (19.7) \end{aligned}$$

$$P(v_i|h) = \frac{e^{v_i (b_i + \sum_j W^{v|h}_{i,j} h_j)}}{Z_i(h)} \tag{19.8}$$

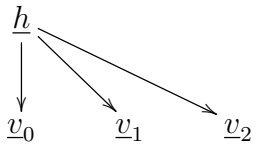


Figure 19.1: bnet for a Restricted Boltzmann Machine (rebo) with $numv = 3$

Eq.19.8 implies that a rebo can be represented by the bnet Fig.19.1.
Let

$$x_i = b_i + \sum_j W_{ij}^{v|h} h_j \ . \tag{19.9}$$

Then

$$\begin{aligned} P(v_i = 1|h) &= \frac{e^{x_i}}{1 + e^{x_i}} && (19.10) \\ &= \frac{1}{1 + e^{-x_i}} && (19.11) \\ &= \text{sig}(x_i) \ . && (19.12) \end{aligned}$$

One could also expand the node $\underline{h}$ in Fig.19.1 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 14) with $v_i, h_i \in \{0, 1\}$ and specific $P(h)$ and $P(v_i|h)$ node matrices.

Chapter 20

Simpson's Paradox

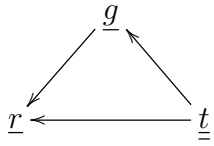


Figure 20.1: bnet for a simple case of Simpson's paradox.

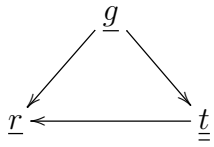


Figure 20.2: Equivalent to Fig.20.1

I wrote an article about this in 2020 for my blog "Quantum Bayesian Networks". See Ref.[13].

Chapter 21

Turbo Codes

This chapter is based on Ref.[14].

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.21.1, or by its abridged version Fig.21.2. 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).

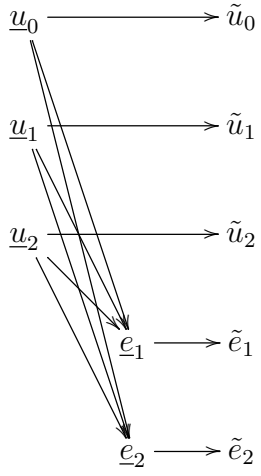


Figure 21.1: Turbo coding B net representing a message being encoded two different ways and then the original message and the 2 encodings pass through a memoryless channel.

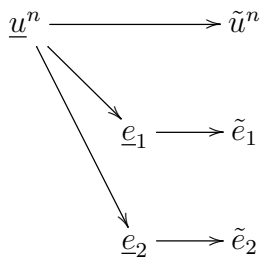


Figure 21.2: Abridged version of Fig.21.1.

Define

$$x = (u^n, e_1, e_2) \tag{21.1}$$

and

$$\tilde{x} = (\tilde{u}^n, \tilde{e}_1, \tilde{e}_2) \ . \tag{21.2}$$

Fig.21.1 implies that

$$P(x, \tilde{x}) = P(\tilde{u}^n|u^n) \left[\prod_{r=1,2} P(\tilde{e}_r|e_r) P(e_r|u^n) \right] P(u^n) \ . \tag{21.3}$$

Because the u^n are i.i.d.,

$$P(u^n) = \prod_i P(u_i) \ . \tag{21.4}$$

Because the channel is memoryless,

$$P(\tilde{u}^n|u^n) = \prod_i P(\tilde{u}_i|u_i) \ . \tag{21.5}$$

Because the encoding is deterministic, we must have for $r = 1, 2$

$$P(e_r|u^n) = \delta(e_r, e_r(u^n)) \ . \tag{21.6}$$

Define the belief functions

$$BEL_i = BEL_i(\underline{u}_i = a) = P(\underline{u}_i = a|\tilde{x}) \ . \tag{21.7}$$

The best estimate of u_j given all visible evidence $\tilde{x}$ is

$$\hat{u}_i = \text{argmax}_{u_i} BEL_i(u_i) \ . \tag{21.8}$$

Define the probability functions

$$\pi_i = \pi_i(u_i) = P(u_i) \ , \tag{21.9}$$

and the likelihood functions

$$\lambda_i = \lambda_i(u_i) = P(\tilde{u}_i|u_i) \ . \tag{21.10}$$

For $r = 1, 2$, define the Kernel functions

$$K_r = K_r(u^n) = P(\tilde{e}_r|e_r = e_r(u^n)) \ . \tag{21.11}$$

In this book, $\mathcal{N}(!a)$ denotes a normalization constant that does not depend on a. Define

$$\mathcal{N}_i = \mathcal{N}(!u_i) \ . \tag{21.12}$$

Claim 4

$$BEL_i = \mathcal{N}_i \lambda_i \pi_i \mathcal{T}_i^{K_1 K_2} [\prod_{j \neq i} \lambda_j \pi_j] \ , \tag{21.13}$$

where $\mathcal{T}_i^K(\cdot)$ with $K = K_1 K_2$ is an operator (transform) that acts on functions of u^n:

$$\mathcal{T}_i^K(\cdot) = \sum_{u^n} \delta(u_i, a) K(u^n)(\cdot) \ . \tag{21.14}$$

proof:

$$\begin{aligned}
P(\underline{u}_i = a|\tilde{x}) = & \\
&= \sum_x \delta(u_i, a) P(x|\tilde{x}) && (21.15)\\
&= \sum_x \delta(u_i, a) \frac{P(\tilde{x}|x)P(x)}{P(\tilde{x})} && (21.16)\\
&= \mathcal{N}(!a) \sum_x \delta(u_i, a) P(\tilde{x}|x) P(x) && (21.17)\\
&= \mathcal{N}(!a) \sum_x \delta(u_i, a) P(u^n) \left[\prod_{r=1,2} P(\tilde{e}_r|e_r) \delta(e_r, e_r(u^n)) \right] \prod_j P(\tilde{u}_j|u_j) && (21.18)\\
&= \mathcal{N}(!a) \lambda_i(a) \pi_i(a) R \ , && (21.19)
\end{aligned}$$

where

$$
\begin{aligned}
R &= \sum_{u^n} \delta(u_i, a) \left[\prod_{r=1,2} P(\tilde{e}_r | e_r(u^n)) \right] \prod_{j \neq i} P(\tilde{u}_j | u_j) P(u_j) && (21.20) \\
&= \sum_{u^n} \delta(u_i, a) \left[\prod_{r=1,2} K_r(u^n) \right] \prod_{j \neq i} \lambda_j(u_j) \pi_j(u_j) && (21.21) \\
&= \mathcal{T}_i^{K_1 K_2} [\prod_{j \neq i} \lambda_j(u_j) \pi_j(u_j)] \; . && (21.22)
\end{aligned}
$$

Hence

$$
BEL_i(a) = \mathcal{N}(!a) \lambda_i(a) \pi_i(a) \mathcal{T}_i^{K_1 K_2} [\prod_{j \neq i} \lambda_j(u_j) \pi_j(u_j)] \; . \tag{21.23}
$$

QED

Decoding Algorithm

The Turbo algorithm for decoding the encode message is as follows. For $m = 0$, let

$$
\pi_j^{(0)}(u_j) = \frac{1}{n_{\underline{u}_j}} \; . \tag{21.24}
$$

Then for $m = 1, 2, \ldots,$ let

$$
\pi_i^{(m)} = \mathcal{N}_i \mathcal{T}_i^{K_{m\%2}} [\prod_{j \neq i} \lambda_j \pi_j^{(m-1)}] \; , \tag{21.25}
$$

where $m\%2 = 1$ if m is odd and $m\%2 = 2$ if m is even. Furthermore, for $m > 0$, let

$$
\begin{aligned}
BEL_i^{(m)} &= \mathcal{N}_i \lambda_i \pi_i^{(m-1)} \pi_i^{(m)} && (21.26) \\
&= \mathcal{N}_i \lambda_i \pi_i^{(m-1)} \mathcal{T}_i^{K_{m\%2}} [\prod_{j \neq i} \lambda_j \pi_j^{(m-1)}] \; . && (21.27)
\end{aligned}
$$

As $m \to \infty$, $BEL_i^{(m)}$ given by Eq.(21.27) is expected to converge to the the exact BEL_i given by Eq.(21.13).

Turbo decoding can be represented by the bnets Figs.21.3 and 21.4.

The node transition matrices, printed in blue, for Fig.21.3, are given by:

$$
P(d_i^{(m)} = a \mid \tilde{u}^n, \tilde{e}_{m\%2}) = BEL_i^{(m)}(a) \; . \tag{21.28}
$$

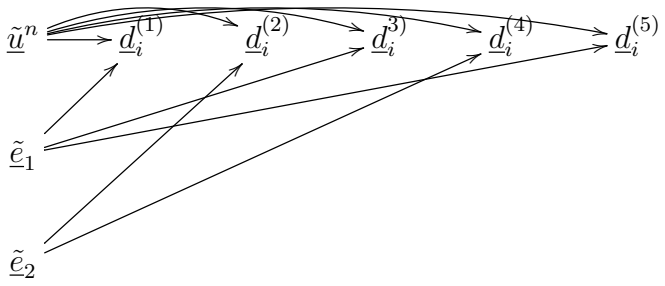


Figure 21.3: B net describing Turbo code generation of $BEL_i^{(m)}(a)$ for $m = 1, 2, \ldots$.

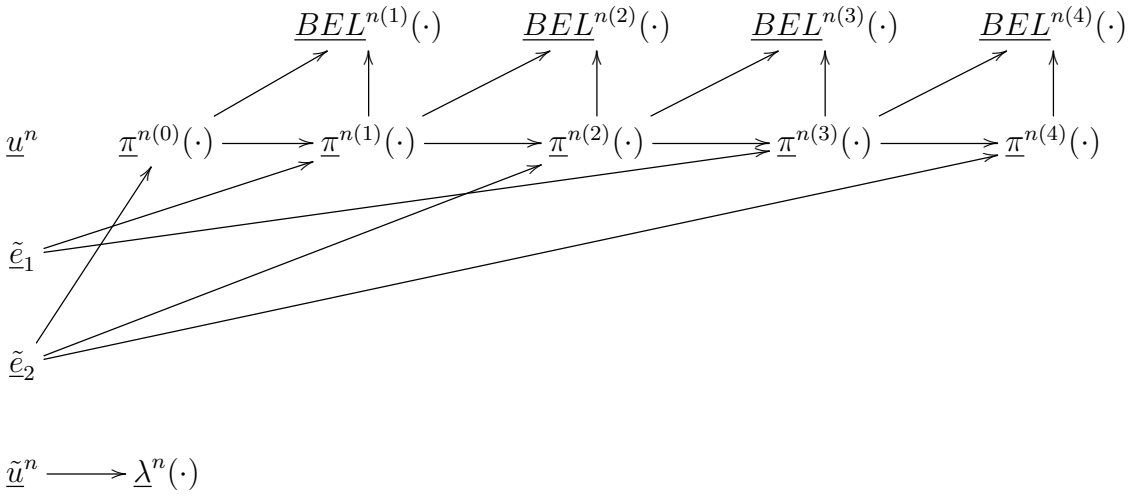


Figure 21.4: B net describing Turbo code generation of $BEL^{n(m)}(\cdot)$ and $\pi^{n(m)}(\cdot)$ for $m = 0, 1, 2 \ldots$. The following arrows were not drawn so as not to unduly clutter the diagram: Arrows pointing from node $\underline{\lambda}^n(\cdot)$ to nodes $\underline{\pi}^{n(m)}(\cdot)$ and $\underline{BEL}^{n(m)}(\cdot)$ for $m = 0, 1, 2, \ldots$.

The node transition matrices, printed in blue, for Fig.21.4, are given by:

$$P((\lambda^n)'(\cdot)|\tilde{u}^n) = \delta((\lambda^n)'(\cdot), \lambda^n(\cdot)) \tag{21.29}$$

$$P(\pi^{n(m)}(\cdot)|\lambda^n(\cdot), \pi^{n(m-1)}(\cdot), \tilde{e}_{m\%2}) = \prod_i \prod_{u_i} \delta(\pi_i^{(m)}(u_i), \mathcal{N}_i \mathcal{T}_i^{K_{m\%2}}[\prod_{j \neq i} \lambda_j \pi_j^{(m-1)}]) \tag{21.30}$$

$$P(BEl^{n(m)}(\cdot)|\lambda^n(\cdot), \pi^{n(m)}(\cdot), \pi^{n(m-1)}(\cdot)) = \prod_i \prod_{u_i} \delta(BEL_i(u_i), \mathcal{N}_i \lambda_i \pi_i^{(m-1)} \pi_i^{(m)}) \tag{21.31}$$

Message Passing Interpretation of Decoding Algorithm

Ref.[14] 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 **??**.

Bibliography

[1] 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/.

[2] Wikipedia. Hidden Markov model. https://en.wikipedia.org/wiki/Hidden_Markov_model.

[3] Gregory Nuel. Tutorial on exact belief propagation in Bayesian networks: from messages to algorithms. https://arxiv.org/abs/1201.4724.

[4] 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.

[5] Wikipedia. Kalman filter. https://en.wikipedia.org/wiki/Kalman_filter.

[6] 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.

[7] Wikipedia. Non-negative matrix factorization. https://en.wikipedia.org/wiki/Non-negative_matrix_factorization.

[8] Andrew Ng. Lecture at deeplearning.ai on recurrent neural networks. http://www.ar-tiste.com/ng-lec-rnn.pdf.

[9] Wikipedia. Long short term memory. https://en.wikipedia.org/wiki/Long_short-term_memory.

[10] Wikipedia. Gated recurrent unit. https://en.wikipedia.org/wiki/Gated_recurrent_unit.

[11] 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".

[12] Sergey Levine. Course CS 285 at UC Berkeley, Deep reinforcement learning. http://rail.eecs.berkeley.edu/deeprlcourse/.

[13] 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/.

[14] 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.