Bayesuvius,
a small visual dictionary of Bayesian Networks

Robert R. Tucci
www.ar-tiste.xyz

August 20, 2020

Figure 1: View of Mount Vesuvius from Pompeii

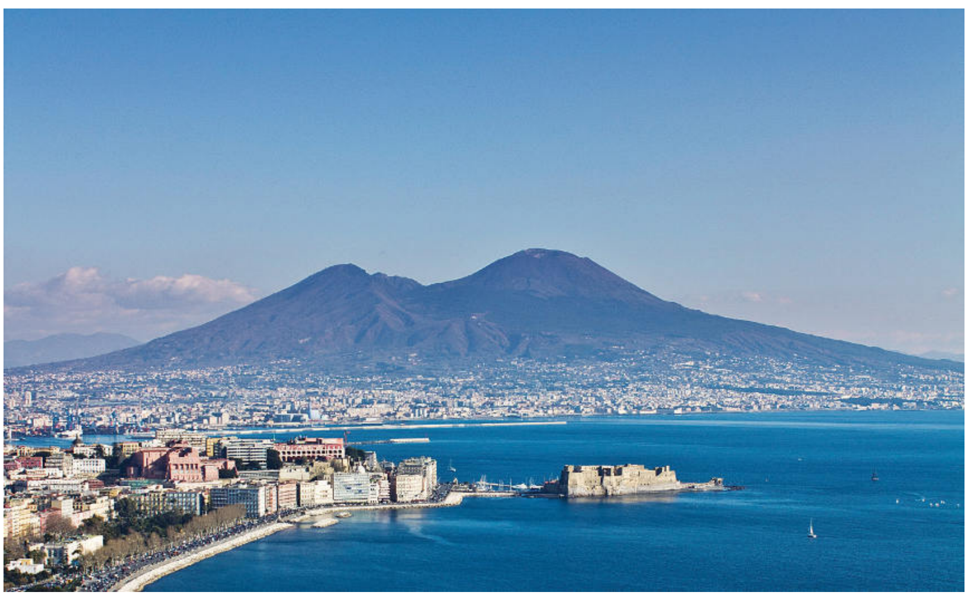

Figure 2: Mount Vesuvius and Bay of Naples

Contents

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.

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

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

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

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

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

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

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

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 $\int_0^\infty dx\ P(x) = 1$.

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

$$x \text{ or } y = x \vee y = x + y - xy \ , \tag{33}$$

$$x \text{ and } y = x \wedge y = xy \ , \tag{34}$$

and

$$\text{not } x = \overline{x} = 1 - x \ , \tag{35}$$

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

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

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

$$P(\overline{x}) = 1 - P(x) \ . \tag{36}$$

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

$$P(x \vee y) = P(x) + P(y) - P(x \wedge y) \ . \tag{37}$$

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 transition prob matrices

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

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

$$\underline{f}^4 \longleftarrow \underline{f}^3 \longleftarrow \underline{f}^2 \longleftarrow \underline{f}^1 \longleftarrow \underline{f}^0$$

(a) Composition

$$\underline{\tfrac{\partial f^4}{\partial x}} \longleftarrow \underline{\tfrac{\partial f^3}{\partial x}} \longleftarrow \underline{\tfrac{\partial f^2}{\partial x}} \longleftarrow \underline{\tfrac{\partial f^1}{\partial x}} \longleftarrow \underline{1}$$

(b) Forward-p

$$\underline{1} \longrightarrow \underline{\tfrac{\partial y}{\partial f^3}} \longrightarrow \underline{\tfrac{\partial y}{\partial f^2}} \longrightarrow \underline{\tfrac{\partial y}{\partial f^1}} \longrightarrow \underline{\tfrac{\partial y}{\partial f^0}}$$

(c) Back-p

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

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

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 transition prob matrices

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

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 transition prob matrices

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

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 $w_{i|j}$.

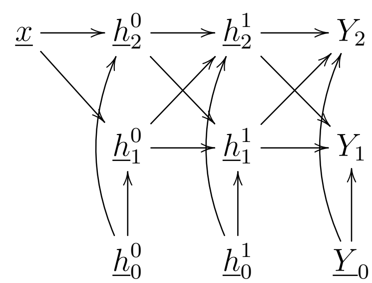

Figure 1.2: Nodes $\underline{h}_0^0, \underline{h}_0^1, \underline{Y}_0$ are all set to 1. They allow us to absorb b_i^λ into the first column of $w_{i|j}^\lambda$.

Below are, printed in blue, the transition prob matrices for the nodes of a NN bnet, as given in Chapter 22.

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

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

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 $w_{\cdot|\cdot}^{\lambda}$ by the augmented matrix $[b^{\lambda}_{\cdot}, w_{\cdot|\cdot}^{\lambda}]$ so that the new $w_{\cdot|\cdot}^{\lambda}$ satisfies

$$w_{i|0}^{\lambda} = b_i^{\lambda} \tag{1.19}$$

Let the nodes $\underline{h}_0^{\lambda}$ for all λ and $\underline{Y}_0$ be root nodes (so no arrows pointing into them). For each λ, draw arrows from $\underline{h}_0^{\lambda}$ 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 transition prob matrices, printed in blue, for the nodes of the NN bnet are as follows:

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

$$P(h_0^{\lambda}) = \delta(h_0^{\lambda}, 1) \ , \tag{1.20}$$

and

$$P(h_i^{\lambda} \mid h_{\cdot}^{\lambda-1}, h_0^{\lambda} = 1) = \delta\left(h_i^{\lambda}, \mathcal{A}_i^{\lambda}(\sum_j w_{i|j}^{\lambda} h_j^{\lambda-1})\right) \tag{1.21}$$

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

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

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:

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

$$\underline{\mathcal{A}^3} \longleftarrow \underline{\mathcal{B}^3} \longleftarrow \underline{\mathcal{A}^2} \longleftarrow \underline{\mathcal{B}^2} \longleftarrow \underline{\mathcal{A}^1} \longleftarrow \underline{\mathcal{B}^1} \longleftarrow \underline{\mathcal{A}^0} \longleftarrow \underline{\mathcal{B}^0} \longleftarrow \underline{x}$$

(a)

$$\underline{\frac{\partial\mathcal{A}^3}{\partial x}} \longleftarrow \underline{\frac{\partial\mathcal{B}^3}{\partial x}} \longleftarrow \underline{\frac{\partial\mathcal{A}^2}{\partial x}} \longleftarrow \underline{\frac{\partial\mathcal{B}^2}{\partial x}} \longleftarrow \underline{\frac{\partial\mathcal{A}^1}{\partial x}} \longleftarrow \underline{\frac{\partial\mathcal{B}^1}{\partial x}} \longleftarrow \underline{\frac{\partial\mathcal{A}^0}{\partial x}} \longleftarrow \underline{\frac{\partial\mathcal{B}^0}{\partial x}} \longleftarrow \underline{1}$$

(b)

$$\underline{1} \longrightarrow \underline{\frac{\partial Y}{\partial\mathcal{B}^3}} \longrightarrow \underline{\frac{\partial Y}{\partial\mathcal{A}^2}} \longrightarrow \underline{\frac{\partial Y}{\partial\mathcal{B}^2}} \longrightarrow \underline{\frac{\partial Y}{\partial\mathcal{A}^1}} \longrightarrow \underline{\frac{\partial Y}{\partial\mathcal{B}^1}} \longrightarrow \underline{\frac{\partial Y}{\partial\mathcal{A}^0}} \longrightarrow \underline{\frac{\partial Y}{\partial\mathcal{B}^0}} \longrightarrow \underline{\frac{\partial Y}{\partial x}}$$

(c)

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).

One has

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

Define $\mathcal{B}^\lambda_i$ thus

$$\mathcal{B}^\lambda_i(h^{\lambda-1}) = \sum_j w^\lambda_{i|j} h^{\lambda-1}_j \; . \tag{1.27}$$

Then

$$\begin{aligned}
\frac{\partial Y}{\partial w^\lambda_{i|j}} &= \frac{\partial Y}{\partial\mathcal{B}^\lambda_i}\frac{\partial\mathcal{B}^\lambda_i}{\partial w^\lambda_{i|j}} && (1.28)\\
&= \frac{\partial Y}{\partial\mathcal{B}^\lambda_i} h^{\lambda-1}_j && (1.29)
\end{aligned}$$

$$\begin{aligned}\frac{\partial\mathcal{E}}{\partial w^\lambda_{i|j}} &= \frac{\partial\mathcal{E}}{\partial\mathcal{B}^\lambda_j}\frac{\partial\mathcal{B}^\lambda_j}{\partial w^\lambda_{i|j}} && (1.30)\\ &= \frac{\partial\mathcal{E}}{\partial\mathcal{B}^\lambda_j}h^{\lambda-1}_j\,. && (1.31)\end{aligned}$$

This suggest that we can calculate the derivatives of the error $\mathcal{E}$ with respect to the weights $w^\lambda_{i|j}$ in two stages, using an intermediate quantity δ^λ_j:

$$\begin{cases}\delta^\lambda_j = \frac{\partial\mathcal{E}}{\partial\mathcal{B}^\lambda_j}\\ \frac{\partial\mathcal{E}}{\partial w^\lambda_{i|j}} = \delta^\lambda_j h^{\lambda-1}_j\end{cases} \tag{1.32}$$

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:

$$f^0_i = x_i \tag{1.33}$$

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

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

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

One has

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

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

From Eq.(1.27)

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

so

$$\frac{\partial\mathcal{B}^{\lambda+1}_i}{\partial\mathcal{A}^\lambda_j} = w^{\lambda+1}_{i|j}\,. \tag{1.41}$$

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:

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

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 transition prob matrix 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 18

Markov Chain Monte Carlo (MCMC)

Inverse of Cumulative Sampling

$$CUM_{\underline{x}}(x) = P(\underline{x} < x) = \int_{x' < x} dx'\ P_{\underline{x}}(x') \tag{18.1}$$

$$\begin{aligned} P(CUM_{\underline{x}}^{-1}(\underline{u}) < x) &= P(\underline{u} < CUM_{\underline{x}}(x)) && (18.2) \\ &= CUM_{\underline{x}}(x) && (18.3) \end{aligned}$$

$$\begin{tikzcd} \underline{u}^{(0)} \arrow[d] & \underline{u}^{(1)} \arrow[d] & \underline{u}^{(2)} \arrow[d] \\ \underline{\vec{x}}^{(0)} \arrow[r] & \underline{\vec{x}}^{(1)} \arrow[r] & \underline{\vec{x}}^{(2)} \end{tikzcd}$$

Figure 18.1: Inverse Cumulative sampling

$$P(u^{(t)}) = 1 \tag{18.4}$$

$$P(\vec{x}^{(t)} | \vec{x}^{(t-1)}, u^{(t)}) = \mathbb{1}(\quad \vec{x}^{(t)} = \vec{x}^{(t-1)} \oplus CUM_{\underline{x}}^{-1}(u^{(t)}) \quad) \tag{18.5}$$

Rejection Sampling

$$P_{\underline{x}}(x) < \beta P_{\underline{c}}(x) \tag{18.6}$$

$$P(u^{(t)} = u) = 1 \tag{18.7}$$

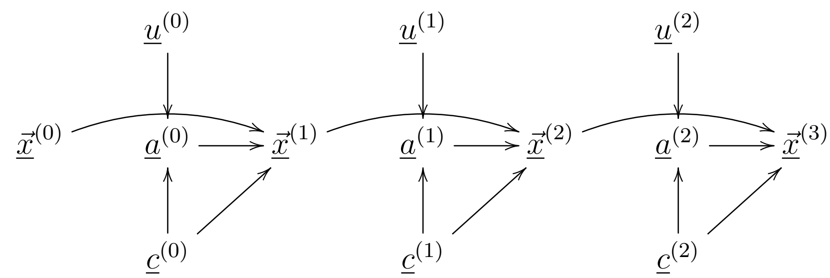

Figure 18.2: Rejection Sampling

$$P(\underline{c}^{(t)} = x) = P_{\underline{c}}(x) \tag{18.8}$$

$$P(\underline{a}^{(t)} = a | \underline{c}^{(t)} = x, \underline{u}^{(t)} = u) = \begin{cases} \mathbb{1}(a = 0) & \text{if } u\beta P_{\underline{c}}(x) \geq P_{\underline{x}}(x)) \\ \mathbb{1}(a = 1) & \text{if } u\beta P_{\underline{c}}(x) < P_{\underline{x}}(x)) \end{cases} \tag{18.9}$$

$$P(\vec{x}^{(t)} | \vec{x}^{(t-1)}, \underline{a}^{(t)} = a, \underline{c}^{(t)} = x) = \begin{cases} \mathbb{1}(\vec{x}^{(t)} = \vec{x}^{(t-1)}) & \text{if } a = 0 \\ \mathbb{1}(\vec{x}^{(t)} = \vec{x}^{(t-1)} \oplus x) & \text{if } a = 1 \end{cases} \tag{18.10}$$

Importance Sampling

Metropolis-Hastings Sampling

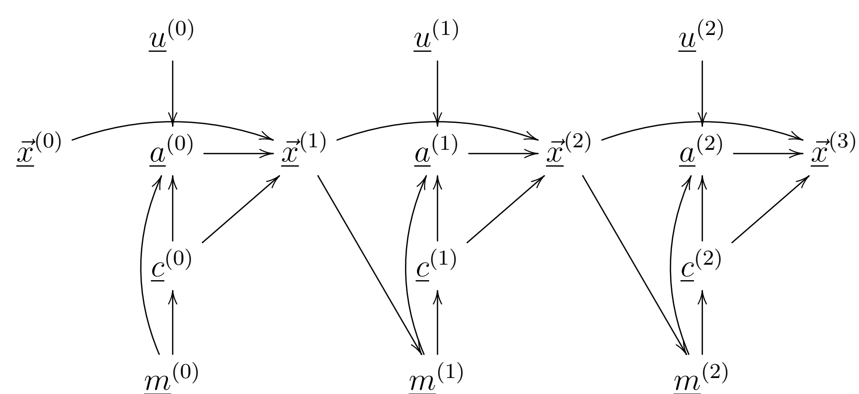

Figure 18.3: Metropolis-Hastings sampling

Gibbs Sampling

Chapter 22

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.

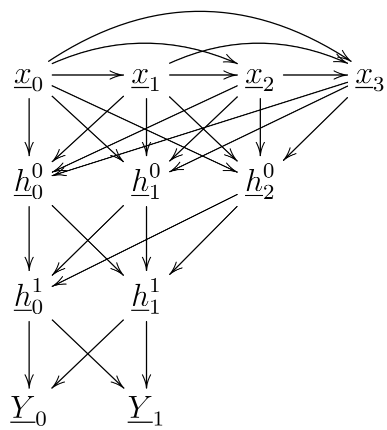

Figure 22.1: Neural Network (feed forward) with 4 layers: input layer $\underline{x}.$, 2 hidden layers $\underline{h}^0.$, $\underline{h}^1.$ and output layer $\underline{Y}$.

Consider Fig.22.1.

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

$\underline{h}_i^\lambda \in \mathbb{R}$ 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.22.1 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 transition probability matrices, printed in blue, for the nodes of the bnet Fig.22.1:

$$P(x_i \mid x_{i-1}, x_{i-1}, \ldots, x_0) = \text{given} \tag{22.1}$$

$$P(h^\lambda_i \mid h^{\lambda-1}_.) = \delta\left(h^\lambda_i, \mathcal{A}^\lambda_i(\sum_j w^\lambda_{i|j} h^{\lambda-1}_j + b^\lambda_i)\right) , \tag{22.2}$$

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

$$P(Y_i \mid h^{\Lambda-2}_.) = \delta\left(Y_i, \mathcal{A}^{\Lambda-1}_i(\sum_j w^{\Lambda-1}_{i|j} h^{\Lambda-2}_j + b^{\Lambda-1}_i)\right) . \tag{22.3}$$

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

Activation functions must be nonlinear.

- **Step function (Perceptron)**

$$\mathcal{A}(x) = \mathbb{1}(x > 0) \tag{22.4}$$

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

- **Sigmoid function**

$$\mathcal{A}(x) = \frac{1}{1+e^{-x}} = \text{sig}(x) \tag{22.5}$$

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

$$\begin{aligned} \text{sig}(x) + \text{sig}(-x) &= \frac{1}{1+e^{-x}} + \frac{1}{1+e^{x}} && (22.6) \\ &= \frac{2+e^x+e^{-x}}{2+e^x+e^{-x}} && (22.7) \\ &= 1 && (22.8) \end{aligned}$$

- **Hyperbolic tangent**

$$\mathcal{A}(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{22.9}$$

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

Odd function:

$$\tanh(-x) = -\tanh(x) \tag{22.10}$$

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

- **ReLU (Rectified Linear Unit)**

$$\mathcal{A}(x) = x\mathbb{1}(x > 0) = \max(0, x) \ . \tag{22.11}$$

Compare this to the step function.

- **Swish**

$$\mathcal{A}(x) = x \ \text{sig}(x) \tag{22.12}$$

- **Softmax**

$$\mathcal{A}(x_i|x.) = \frac{e^{x_i}}{\sum_i e^{x_i}} \tag{22.13}$$

It's called softmax because if we approximate the exponentials, both in the numerator and denominator of Eq.(22.13), by the largest one, we get

$$\mathcal{A}(x_i|x.) \approx \mathbb{1}(x_i = \max_k x_k) \ . \tag{22.14}$$

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,

$$\begin{aligned} \mathcal{A}(x_0|x.) &= \frac{e^{x_0}}{e^{x_0} + e^{x_1}} & (22.15) \\ &= \text{sig}(x_0 - x_1) \ , & (22.16) \end{aligned}$$

$$\mathcal{A}(x_1|x.) = \text{sig}(x_1 - x_0) \ . \tag{22.17}$$

Weight optimization via supervised training and gradient descent

The bnet of Fig.22.1 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.22.2.

In Fig.22.2, the nodes in Fig.22.1 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.22.2. 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

$$W^\lambda_{i|j} = [w^\lambda_{i|j}, b^\lambda_i] \; . \tag{22.18}$$

These are the transition probability matrices, printed in blue, for the nodes of the bnet Fig.22.2:

$$P(x.[s]) = \text{given} \; . \tag{22.19}$$

$$P(y.[s] \mid x.[s]) = \text{given} \; . \tag{22.20}$$

$$P(h^\lambda_i[s] \mid h^{\lambda-1}_.[s]) = \delta\left(h^\lambda_i[s], \mathcal{A}^\lambda_i(\sum_j w^\lambda_{i|j} h^{\lambda-1}_j[s] + b^\lambda_i)\right) \tag{22.21}$$

$$P(Y_i[s] \mid h^{\Lambda-2}_.[s]) = \delta\left(Y_i[s], \mathcal{A}^{\Lambda-1}_i(\sum_j w^{\Lambda-1}_{i|j} h^{\Lambda-2}_j[s] + b^{\Lambda-1}_i)\right) \tag{22.22}$$

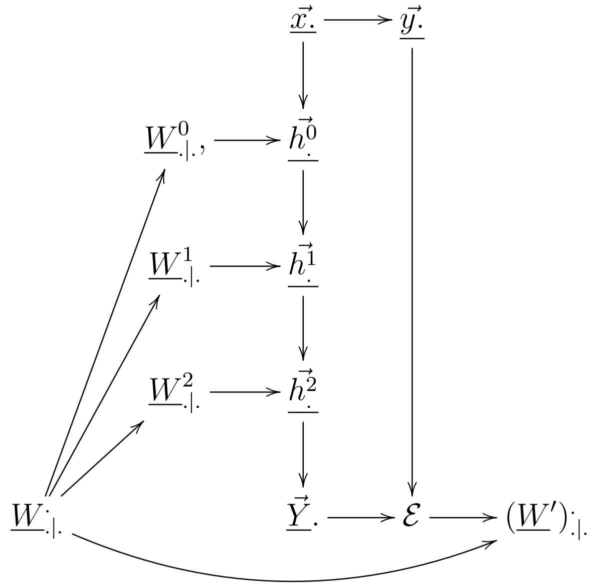

Figure 22.2: bnet for finding optimum weights of the bnet Fig.22.1 via supervised training and gradient descent.

$$P(W^{\cdot}_{.|.}) = \text{given} \tag{22.23}$$

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

$$P(W^{\lambda}_{.|.}|W^{\cdot}_{.|.}) = \delta(W^{\lambda}_{.|.}, (W^{\cdot}_{.|.})^{\lambda}) \tag{22.24}$$

$$P(\mathcal{E}|\vec{y}., \vec{Y}.) = \frac{1}{nsam(\vec{x})} \sum_s \sum_i d(y_i[s], Y_i[s]) \ , \tag{22.25}$$

where

$$d(y, Y) = |y - Y|^2 \ . \tag{22.26}$$

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

$$d(y, Y) = XE(y \to Y) = -y \ln Y - (1 - y) \ln(1 - Y) \ . \tag{22.27}$$

$$P((W')^{\lambda}_{i|j}|\mathcal{E}, W^{\cdot}_{.|.}) = \delta((W')^{\lambda}_{i|j}, W^{\lambda}_{i|j} - \eta \partial_{W^{\lambda}_{i|j}} \mathcal{E}) \tag{22.28}$$

$\eta > 0$ 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:

$$P(h_i^\lambda[s] \mid h_.^{\lambda-1}[s]) = \delta(h_i^\lambda[s], H_i^\lambda[s]) \ , \tag{22.29}$$

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.[14]. 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

  $$P(r_i^\lambda = r) = [p]^r[1-p]^{1-r} \text{ (Bernouilli dist.)} \ . \tag{22.30}$$

  Now one has

  $$P(h_i^\lambda[s] \mid h_.^{\lambda-1}[s], r_i^\lambda) = \delta(h_i^\lambda[s], H_i^\lambda[s]) \ , \tag{22.31}$$

  where

  $$H_i^\lambda[s] = \mathcal{A}_i^\lambda(r_i^\lambda \sum_j w_{i|j}^\lambda h_j^{\lambda-1}[s] + b_i^\lambda) \ . \tag{22.32}$$

  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.22.2, not in the classification bnet Fig.22.1. We prefer to remove the $\underline{r}_i^\lambda$ stochasticity from classification and for Fig.22.1 to act as an average over sampling space of Fig.22.2. Therefore, if weights $w_{i|j}^\lambda$ are obtained for a dropout layer λ in Fig.22.2, then that layer is used in Fig.22.1 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

$$H_i^\lambda[s] = \sum_{j=0}^{nf-1} h_{j+i\sigma}^{\lambda-1}[s]\mathcal{F}(j) \ . \tag{22.33}$$

For the indices not to go out of bounds in Eq.(22.33), we must have

$$nh(\lambda - 1) - 1 = nf - 1 + (nh(\lambda) - 1)\sigma \tag{22.34}$$

so

$$nh(\lambda) = \frac{1}{\sigma}[nh(\lambda - 1) - nf] + 1 \ . \tag{22.35}$$

- 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
  $$H_i^\lambda[s] = \frac{1}{size(Pool(i))} \sum_{j \in Pool(i)} h_j^{\lambda-1}[s] \tag{22.36}$$

  - MaxPool
  $$H_i^\lambda[s] = \max_{j \in Pool(i)} h_j^{\lambda-1}[s] \tag{22.37}$$

Autoencoder NN

If the sequence

$$nx, nh(0), nh(1), \ldots, nh(\Lambda - 2), ny \tag{22.38}$$

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**.

Bibliography

[1] Wikipedia. Boolean algebra. https://en.wikipedia.org/wiki/Boolean_algebra.

[2] ReliaSoft. System analysis reference. http://reliawiki.org/index.php/System_Analysis_Reference.

[3] 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/.

[4] 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/.

[5] Wikipedia. Binary decision diagram. https://en.wikipedia.org/wiki/Binary_decision_diagram.

[6] Wikipedia. Expectation maximization. https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm.

[7] Wikipedia. k-means clustering. https://en.wikipedia.org/wiki/K-means_clustering.

[8] 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.

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

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

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

[12] Wikipedia. Markov blanket. https://en.wikipedia.org/wiki/Markov_blanket.

[13] Judea Pearl. *Probabilistic Inference in Intelligent Systems*. Morgan Kaufmann, 1988.

[14] 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.

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

[16] theinvestorsbook.com. Pert analysis. https://theinvestorsbook.com/pert-analysis.html.

[17] Wikipedia. Program evaluation and review technique. https://en.wikipedia.org/wiki/Program_evaluation_and_review_technique.

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

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

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

[21] 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".

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

[23] 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/.

[24] 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.