Deep Learning Part-2

Representing Joint Distributions: Suppose there are 3 random variables :- I takes 2 values, G takes 3 values, S takes 2 values. Now if we want to represent their joint distribution, we would require $2 \times 3 \times 2 - 1 = 11$ values, last one is 1 - (sum of 11). If we use chain rule for representing the joint representation :- $P(S,G,I) = P(S|G,I) \times P(G|I) \times P(I)$

$P(I)$ can be represented with 1 value :- $P(I_1)$ or $P(I_2)$

$P(G|I)$ can be represented with 4 values :- $P(G_1|I_1), P(G_2,I_1)$
$P(G_1|I_2), P(G_2,I_2)$

$P(G_3|I_1)$ and $P(G_3|I_2)$ can be calculated/derived

$P(S|G,I)$ can be represented with 6 values.

$P(S_1|$ 6 different combinations of I and G$)$

In total we still take $1+4+6 = 11$ values to represent the joint distribution. But if we make some assumptions on the conditional independence of S and G given I, then the number of parameters can be reduced. So $P(S|G,I) = P(S|I)$. Hence the joint distribution can

$P(G|S,I) = P(G|I)$

now be written as :- $P(S,G,I) = P(S|I) \cdot P(G|I) \times P(I)$

2 values $P(S_1|I_1)$ and $P(S_1|I_2)$ — 4 values — 1 value.

The number of parameters reduces to $2+4+1 = 7$ values.

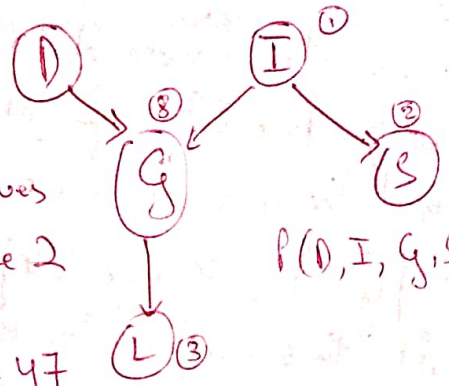
Assumptions like this can be made based on the application, the advantages of this representation are :- 1) more natural, variables that exhibit dependence on others is made explicit.

2) More compact, less parameters required if conditional independence is assumed.

3) More modular, these values can be reused if more variables are added later, reusing values is not an option in plain representation.

Bayesian Networks :- Nodes are random variables and directed edges specify the dependence relations between the random variables.

For example :-



The joint distribution can be specified as:-

$$P(D, I, G, S, L) = P(D) \cdot P(I) \cdot P(S|I) \cdot P(G|D, I) \cdot P(L|G)$$

If, G takes 3 values and all others take 2 values, the plain representation takes = 47 values

will the compact representation takes $1+1+2+8+3 = 15$ values.

The graph structure along with conditional probability distributions is called a Bayesian Network.

⇒ The independence assumptions are made by us as a modelling choice and the number of parameters and data required to train the model is a consequence of our modelling choice.

Reasoning Encoded in Bayesian Network :-

i) **Causal Reasoning**:- we try to predict downstream effects of various factors. For example asking question like what is the chance, the student gets a good LOR (L^1) given that they are intelligent (I^1). Intelligence (I) is a factor and how it affects L is the question that is being asked here. $P(L^1|I^1)$

$$P(L^1|I^1) = \frac{P(L^1, I^1)}{P(I^1)}$$

, we expect that $P(L^1|I^1) > P(L^1)$ since the chance of getting a good LOR increases given the student is intelligent.

ii) **Evidential Reasoning**:- We reason about causes by looking at their effects. Asking questions like what is the probability of the student being intelligent given that he got a poor Grade(G) $P(I^1|G^3)$, we expect $P(I^1|G^3) < P(I^1)$, same for difficulty $P(D^1|G^3) > P(D^1)$, the difficulty of course is most probably hard given the student scored poorly.

(increases)

(ii) Explaining Away :- We see how different causes of the same effect can interact. We already saw that $P(I'|G^3) < P(I')$ But what if we are given another cause of the grade, the difficulty. So probability of student being intelligent given he scored poorly and the course was difficult. $P(I'|G^3, D') > P(I'|G^3)$, is our expectation since now the course was also difficult, this compensates the drop in intelligence. So in a way, various causes explain the variation in other causes.

Independencies Encoded by a Bayesian Network :-

Given n random variables, we are interested in knowing if $X_i \perp X_j$ or $X_i \perp X_j | Z$ where $Z \in X_1, X_2 \cdots, X_n / X_i, X_j$

Rule 1 :- A node is not independent of its parents. (By construction)

Rule 2 :- A node is not independent of its parents even when we are given the values of other variables. For example knowing the value of I does not mean that L and G become independent - L still depends on G even after knowing I.

These were some independencies between nodes and their parents. Now what about non-parents. L can still be affected by S which is a non-parent, since S affects I and I affects G which then affects L. So $L \not\perp S$, but given the value of G, $L \perp S$ holds. So a node is independent of non-descendents given the parents. So $L \perp S | G$, moreover $L \perp I | G$ also holds. This is because, the effect S or I have on L is through G, but we already know G, so knowing S or I does not matter for L.

Independencies of a node and its descendants is also encoded.

Rule 3 :- A node is independent to all non-descendents given the parent variables of the node.

Formally :- A Bayesian Network is a directed acyclic graph where nodes represent variables. Let P_{X_i} denote the parents of X_i

and $ND(X_i)$ denotes the non-descendants of X_i (does not include parents). Then for each variable X_i the graph encodes the following conditional independence assumptions:-

$$I_i(G) = \left((X_i \perp ND(X_i)) \mid P_{X_i} \right)$$

I-Maps :- Let P be a joint distribution over $X = X_1, X_2, \cdots, X_n$. $I(P)$ be the set of independence assumptions that hold in P. Each element of this set is of the form $X_i \perp X_j \mid Z$, $Z \subseteq X \mid X_i, X_j$. Let $I(G)$ be the set of independence assumptions associated with a graph G. We say that G is an I-map for P if $I(G) \subseteq I(P)$. Hence all the assumptions that G states hold in P. But P can have additional independencies.

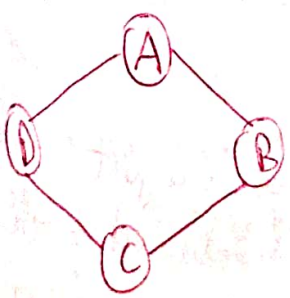
In practice, we do not know P and hence can't compute $I(P)$. So we just make assumptions about $I(P)$ and then construct a G such that $I(G) \subseteq I(P)$.

Theorem :- Let G be a BN over a set of random variables X and let P be a joint distribution over these variables. If G is an I-Map for P, then P factorizes according to G.

Theorem :- Let G be a BN over a set of random variables X and let P be a joint distribution over these variables. If P factorizes according to G, then G is an I-Map of P.

Perfect Map :- A graph G is a Perfect Map for a distribution P if the independence relations implied by the graph are exactly the same as those implied by the distribution.

Undirected Graphical Models :-

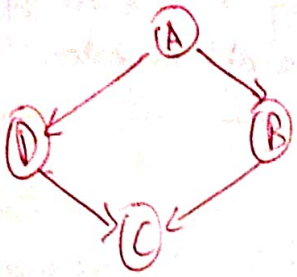


The conditional independencies in this problem are:-

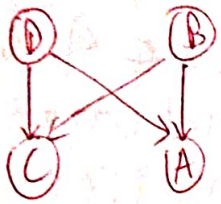
$A \perp C \mid \{B, D\}$ (Because A and C never interact)

$B \perp D \mid \{A, C\}$ (Because B and D never interact)

What if we try to model this using Bayesian Networks?



this captures $A \perp C \mid \{B, D\}$ but it does not capture $B \perp D \mid \{A, C\}$ because given C and B can affect D (Explaining Away)

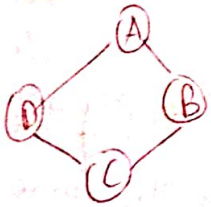


$A \perp C \mid \{B, D\}$ is encoded
But it also encodes
$B \perp D$ (unconditional)

> Turns out that no Bayesian Network can exactly capture independence relations completely (Perfect Map)

This is because a directed edge between two nodes implies some kind of direction in the interaction. But in our example A and B are equal partners. We instead want to capture the strength of the interaction between the nodes. These undirected network models are called Markov Networks.

We can't use Conditional Probability Distributions as they don't make sense in the undirected case since there is no natural conditioning. Hence the parameters in this case is the affinity strength between connected random variables. We can have factors $\phi_1(A,B)$, $\phi_2(B,C)$, $\phi_3(C,D)$, $\phi_4(A,D)$ which capture the affinity between the corresponding nodes. Suppose All variables take 2 values, then:-



$\phi_1(A,B)$			$\phi_2(B,C)$	...	$\phi_4(A,D)$	
a^0	b^0	30	b^0	c^0	a^0	d^0
a^0	b^1	5				
a^1	b^0	1				
a^1	b^1	10				

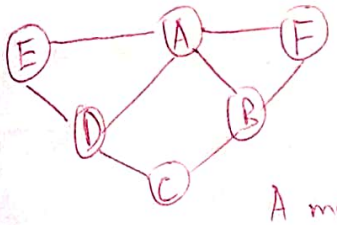
Notice that here parameters are defined over the edges, while in BN's the parameters were defined over the nodes.

These tables are not probability distributions. These are just weights which can be interpreted as the relative likelihood of an event. To convert these into probability distributions we can normalize the product :-

$$P(a,b,c,d) = \frac{1}{Z} \times \phi_1(a,b) \cdot \phi_2(b,c) \cdot \phi_3(c,d) \cdot \phi_4(d,a)$$

where $Z = \sum_{a,b,c,d} \phi_1(a,b) \cdot \phi_2(b,c) \cdot \phi_3(c,d) \cdot \phi_4(a,d)$

→ for all values of a, b, c, d. (combinations.)



We could still factorize pairwise or we could factorize based on maximal cliques.

A maximal cliques is a set of nodes in which adding any other will result in a non-clique set!

A clique is a set of nodes which are pairwise connected to each other.

So the above network can be factorized like this:-

$\phi_1(A,D,E) \cdot \phi_2(A,B,F) \cdot \phi_3(B,C) \cdot \phi_4(C,D)$

So a distribution factorizes over a Markov Network H if P can expressed as $P(X_1, \cdots, X_n) = \prod_{i=1}^{m} \phi_i(D_i)$ where D_i is a complete sub graph (clique in H)

A distribution is a Gibbs distribution parametrized by a set of factors $\Phi = \{\phi_1(D_1), \cdots, \phi_m(D_m)\}$ if it is defined as $P(X_1, X_2, \cdots, X_n) = \frac{1}{Z} \prod_{i=1}^{m} \phi_i(D_i)$

⇒ Let X, Y, Z be some distinct subsets of U (set of all Random variables), then a distribution P over these RVs would imply $X \perp Y \mid Z$ if and only if we can write $P(X) = \phi_1(X,Z)\ \phi_2(Y,Z)$

In our example of 4 variables. $P(a,b,c,d) = \frac{1}{Z} \phi_1(a,b) \cdot \phi_2(b,c) \cdot \phi_3(c,d) \cdot \phi_4($

can be written as :- $P(a,b,c,d) = \frac{1}{Z} \phi_5(b,\{a,c\}) \cdot \phi_6(d,\{a,c\}$

Hence $B \perp D \mid \{A,C\}$ holds. Similarly $A \perp C \mid \{B,D\}$ also hold

For a given Markov Network H we define Markov Blanket of a RV X to be the neighbor of X in H.

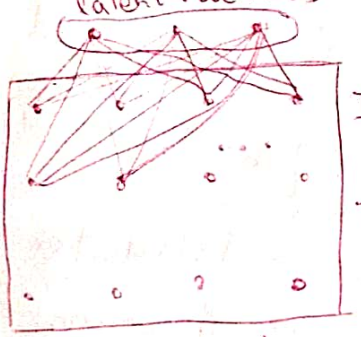
Analogous to the case of Bayesian Networks we can define the local independencies associated with H to be :-

$$X \perp (U - \{X\} - MB_H(X)) \mid MB_H(X)$$

X is independent of its non-neighbors given the neighbors of X in H. The neighborhood is defined by the Markov Blanket. For example in an image, the Markov Blanket can be the 8 pixels that surround it.

Latent Variables:- Now in the case of modelling an image as a Markov Network, each pixel can be treated as a random variable with blanket being the 8 surrounding pixels. So the joint distribution of images can be $P(V_1, V_2, \cdots V_{1024})$ where V_i is a pixel and can be factorized accordingly.

Instead of just modelling the pixels as random variables, we can introduce latent variables such that the value of pixels depend on the ~~visible~~ latent variables. Consider the task of generating a scenery. The latent variables can be:- is the scene sunny or not, does the scene have mountains or beaches, clouds or not etc. Based on these variables, the actual pixels can be generated. The Markov Network looks like this :—



pixels are dependent on the latent variables. The interactions between the pixels are captured through the latent variables.

This modelling choice helps in many tasks like abstraction. Suppose, we are able to learn the joint distribution $P(V,H)$. Using this distribution we can find

$$P(H|V) = \frac{P(V,H)}{\sum_V P(V,H)}$$

marginalize over V.

So given an image, we can find the most likely latent configuration of H that generated this image. The vector h calculated can be thought of as an abstract representation of the whole image. Similar images which are pixel wise still very different in the original space are expected to have very similar latent configuration.

⇒ In practice we don't have latent configurations given to us. This is like a Representation Learning Task and each element in the latent vector representation may not even be interpretable they just help us to learn a good abstraction of the data.

⇒ Another task these representations can help is generation. Using the joint distribution, we can find :- $P(V|H) = \frac{P(V,H)}{\sum_V P(V,H)}$

So, given a latent configuration, generate the image which most likely corresponds to this configuration.

Let the vector V of all visible variables be a binary vector, $V \in \{0,1\}^m$ and the vetor H also be a binary vector $H \in \{0,1\}^n$.

Notice that edges are only incident between each pair of hidden, -visible variables. We donot have edges between (hidden, hidden) and (visible - visible) variables. Hence the cliques in the graph only includes each pair of (hidden-visible) pair of variables. So the joint distribution can be factorized as :- $P(V,H) = \frac{1}{Z} \prod_{i=1}^{m} \prod_{j=1}^{n} \phi_{ij}(v_i, h_j)$

we can add additional factors such that

$$P(V,H) = \frac{1}{Z} \prod_{i=1}^{m} \prod_{j=1}^{n} \phi_{ij}(v_i,h_j) \prod_{i=1}^{m} \psi_i(v_i) \prod_{j=1}^{n} \varepsilon_j(h_j)$$

this is allowed as long as Z correctly normalizes to obtain a prob distribution. Z is calculated as follows :-

$$Z = \sum_V \sum_H \prod_{i=1}^{m} \prod_{j=1}^{n} \phi_{ij}(v_i,h_j) \prod_{i=1}^{m} \psi_i(v_i) \prod_{j=1}^{n} \varepsilon_j(h_j)$$

all 2^m possible configurations → all 2^n possible configurations. Hence this Z term includes 2^{m+n} terms in total.

Each ϕ_{ij} is a table which gives the values for each possible (4) configuration of v_i and h_j. Similarly ψ_i and ε_j are also tables giving values for each possible (2) configuration of v_i and h_j respectively.

Restricted Boltzmann Machines:- TO learn ϕ_{ij}s, ψ_is and ε_js, we introduce parameters. RBMs chooses the following parametric forms:- $\phi_{ij}(v_i, h_j) = e^{w_{ij} v_i h_j}$, $\psi_i(v_i) = e^{b_i v_i}$, $\varepsilon_j(h_j) = e^{c_j h_j}$

With this parametric form, the distribution can be expressed as follows:-

$$P(V,H) = \frac{1}{Z} \prod_{i=1}^{m} \prod_{j=1}^{n} e^{w_{ij} v_i h_j} \prod_{i=1}^{m} e^{b_i v_i} \prod_{j=1}^{n} e^{c_j h_j}$$

$$= \frac{1}{Z} e^{\left(\sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} v_i h_j + \sum_{i=1}^{m} b_i v_i + \sum_{j=1}^{n} c_j h_j \right)}$$

$$\boxed{P(V,H) = \frac{1}{Z} e^{-E(V,H)}} \quad \text{where} \quad \boxed{E(V,H) = -\sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} v_i h_j - \sum_{i=1}^{m} b_i v_i - \sum_{j=1}^{n} c_j h_j}$$

$E(V,H)$ is called the energy

High energy → low probability, and vice versa.

This is called RBM because, we donot allow edges between the visible vertices (restricted), and the form of joint distribution is similar to Boltzmann/Gibbs distribution.

RBMs as stochastic Neural Networks:- Let us try to model $P(H|V)$ and $P(V|H)$ given the form of joint distribution.

Let $V_{-\ell}$ be the state of all visible units except the ℓ^{th} unit.

$$\alpha_\ell(H) = -\sum_{j=1}^{n} w_{\ell j} h_j - b_\ell \quad \Big] \text{ part of } \ell^{th} \text{ visible unit in the summation of } E(V,H).$$

$$\beta(V_{-\ell}, H) = -\sum_{\substack{i=1 \\ i \neq \ell}}^{m} \sum_{j=1}^{n} w_{ij} v_i h_j - \sum_{\substack{i=1 \\ i \neq \ell}}^{m} b_i v_i - \sum_{j=1}^{n} c_j h_j$$

So $E(V,H) = v_\ell \alpha_\ell(H) + \beta(V_{-\ell}, H)$

We can hence calculate $P(V_\ell = 1 \mid H)$. Notice that since $V_{-\ell}$ is independent of V_ℓ given H. $P(V_\ell = 1 | H) = P(V_\ell = 1 | V_{-\ell}, H)$ doesn't matter what configuration $V_{-\ell}$ is.

Representing it in this way allows for using the joint distribution :- $P(V_\ell=1|H) = P(V_\ell=1|V_{-\ell},H) = \frac{P(V_\ell, V_{-\ell}, H)}{P(V_{-\ell},H)}$

$$\Rightarrow \frac{e^{-E(V_\ell=1, V_{-\ell}, H)}}{e^{-E(V_\ell=0, V_{-\ell}, H)} + e^{-E(V_\ell=1, V_{-\ell}, H)}} \Rightarrow \frac{e^{-\alpha_\ell(H) - \beta(V_{-\ell},H)}}{e^{-\alpha_\ell(H)\beta(V_{-\ell},H)} + e^{-\beta(V_{-\ell},H)}}$$

$$\Rightarrow \frac{e^{-\beta(V_{-\ell},H)} \cdot e^{-\alpha_\ell(H)}}{e^{-\beta(V_{-\ell},H)} \cdot e^{-\alpha_\ell(H)} + e^{-\beta(V_{-\ell},H)}} \Rightarrow \frac{e^{-\alpha_\ell(H)}}{e^{-\alpha_\ell(H)} + 1} = \frac{1}{e^{\alpha_\ell(H)} + 1}$$

$$\Rightarrow \boxed{P(V_\ell=1|H) = \sigma(-\alpha_\ell(H)) = \sigma\left(\sum_{j=1}^{n} w_{\ell j} h_j + b_\ell\right)}$$

Similarly it can be shown that $\boxed{P(h_p=1|V) = \sigma\left(\sum_{i=1}^{m} w_{ip} V_i + c_p\right)}$

Hence, the extra factors added to the form of $P(H,V)$ serve as biases in the formulation of RBMs. So RBM's can be interpreted as stochastic neural network where the nodes and edges correspond to neurons and synaptic connections, respectively. The conditional probability of a single variable being 1 can be interpreted as the firing of a neuron with sigmoid activation.

Having $P(v_\ell=1|H)$, the probability $P(V|H)$ is straightforward.

$$P(V|H) = \prod_{i=1}^{m} P(V_i = v_i|H)$$ → since V_i are conditionally independent independent.

$$\begin{cases} \sigma(-\alpha_\ell(H)) & \text{if } v_i = 1 \\ 1 - \sigma(-\alpha_\ell(H)) & \text{if } v_i = 0 \end{cases}$$

Since our task is unsupervised, our objective function should only depend on V's → the images. Eventually we want to maximize the likelihood of our data. So the task is now to find parameters w_{ij}, b_i, c_j that maximize the probability of the images. So we want to maximize $P(V|\Theta)$ → the set of params

So maximize $\prod_{i=1}^{m} P(V_i = v_i | \theta)$. Lets consider the loss for one single example. $P(V_i = v_i)$ is same as $P(V_i = v_i | \theta)$ (this just means based on the current params.)

So $P(V) = \prod_{i=1}^{m} P(V_i = v_i)$

Also we know that $P(V,H) = \frac{1}{Z} e^{-E(V,H)}$. So $P(V)$ can also be calculated $\Rightarrow \frac{1}{Z} \sum_{H} e^{-E(V,H)}$

$\hookrightarrow$ marginalized over H. all possible 2^n configuration.

So $P(V) = \frac{1}{Z} \sum_{H} e^{-E(V,H)} = \frac{\sum_{H} e^{-E(V,H)}}{\sum_{V,H} e^{-E(V,H)}}$ $\rightarrow$ Z, marginalized over 2^{n+m} configuration of V and H.

So $\ln P(V) = \ln \sum_{H} e^{-E(V,H)} - \ln \sum_{V,H} e^{-E(V,H)}$

(we maximize this instead)

$$\frac{\partial \ln P(V)}{\partial \theta} \rightarrow \text{any param} = \frac{-1}{\sum_{H} e^{-E(V,H)}} \times \sum_{H} e^{-E(V,H)} \cdot \frac{\partial E(V,H)}{\partial \theta} + \frac{1}{\sum_{V,H} e^{-E(V,H)}} \times \sum_{V,H} e^{-E(H,V)} \cdot \frac{\partial E(H,V)}{\partial \theta}$$

$$= -\sum_{H} \frac{e^{-E(V,H)}}{\sum_{H} e^{-E(V,H)}} \frac{\partial E(V,H)}{\partial \theta} + \sum_{H,V} \frac{e^{-E(H,V)}}{\sum_{V,H} e^{-E(H,V)}} \cdot \frac{\partial E(H,V)}{\partial \theta}$$

after dividing both numerator and denom by Z, observe that $\frac{P(V,H)}{P(V)} = P(H|V)$

$P(V,H)$

Hence $\boxed{\frac{\partial \ln P(V)}{\partial \theta} = -\sum_{H} P(H|V) \cdot \frac{\partial E(V,H)}{\partial \theta} + \sum_{H,V} P(V,H) \cdot \frac{\partial E(H,V)}{\partial \theta}}$

if $\theta = w_{ij}$ then

$$\frac{\partial L(\theta)}{\partial w_{ij}} = + \sum_{H} p(H|V) h_j v_i \quad - \sum_{H,V} p(V,H) h_j v_i$$

$$= + \underset{p(H|V)}{E}[h_j v_i] \quad - \underset{p(H,V)}{E}[h_j v_i]$$

Both these expectations have an exponential number of terms. And notice that this is gradient wrt 1 param of a single observation. Hence the learning is intractable in its current form. So we need sampling methods to estimate these expectations. But sampling is not straightforward as only a few samples are likely from the huge space of samples. So sampling has to be on the basis of the joint distribution but we don't have the distribution itself.

Solution is to draw samples from an easier distribution as long as sampling from the other distribution is same as if they were drawn from P(original). We do this in Gibbs Sampling.

Goal 1: Given a random variable $X \in \mathbb{R}^n$ we are interested in drawing samples from the joint distribution $P(X)$

Goal 2: Given a function $f(X)$ defined over the random variable X, we are interested in computing the expectation $E_{P(X)}[f(X)]$ (useful for gradient calc in RBMs)

Suppose we have a chain of random variables $X_1, X_2, \ldots X_k$ each $X_i \in \mathbb{R}^n$, basically each Random variable X_i is in itself a vector. i corresponds to a time step. For our discussion, let $X_i \in \{0,1\}^n$. One way of looking is to see that X_i transitions to X_{i+1} when the state changes from i to $i+1$. X_i can take one of 2^n values in each state. One might be interested in finding the most likely value X_i will take given all the values of previous states $X_{i-1}, X_{i-2}, \ldots X_1$.

So we want :- $P(X_i = x_i | X_1 = x_1, X_2 = x_2, \ldots X_{i-1} = x_{i-1})$

But let the chain exhibit the Markov Property:-

$$P(X_i = x_i | X_1 = x_1, X_2 = x_2, \ldots X_{i-1} = x_{i-1}) = P(X_i = x_i | X_{i-1} = x_{i-1})$$

given the previous state X_{i-1}, X_i is independent of all preceding states. $X_1 \to X_2 \to \cdots \to X_k$ $\left[X_i \perp\!\!\!\perp X_1^{i-2} \mid X_{i-1}\right]$.

This chain of random variables is called a Markov Chain. We are dealing with a discrete time and discrete space Markov Chain.

i is discrete; X_i takes discrete values.

Let $2^n = l$. Now to specify $P(X_i = x_i \mid X_{i-1} = x_{i-1})$ we need to specify l^2 values, for each configuration of X_{i-1} and X_i. We can represent this as a matrix $T \in l \times l$ where T_{ab} gives the probability of transitioning from $X_{i-1} = a$ to $X_i = b$, where 'a' and 'b' encode two of the possible 2^n configurations respectively. T is called the transition matrix. T may be different for each transition, but we make the assumption that our Markov Chain is time homogeneous. So for all transitions, T remains the same. $\boxed{P(X_i = b \mid X_{i-1} = a) = T_{ab}} \quad \forall a, b, \forall i$

Let the distribution be specified by μ^k at k^{th} time step. μ^k is a l dimensional vector. $\mu^k_a = P(X_k = a)$. The a^{th} element of μ^k gives the probability of X_k being 'a'. Again, X_k is an n-dimensional vector but 'a' is an encoding of one of such possible vector.

an integer

Let the starting distribution be μ^0. How to calculate $P(X_1 = b)$ from μ^0?

$$P(X_1 = b) = \sum_a P(X_0 = a, X_1 = b) \quad \text{marginalize over } X_0.$$

Probability of X_1 taking b irrespective of the starting point

$$= \sum_a P(X_0 = a) \cdot P(X_1 = b \mid X_0 = a)$$

$$= \sum_a \mu^0_a T_{ab} = \mu'_b \quad \text{since } P(X_1 = b) = \mu'_b$$

So μ' can be written as $\mu' = \mu^0 T = [\mu^0_1 \ \mu^0_2 \cdots \mu'_l] \begin{bmatrix} T_{11} & & \\ T_{21} & & \\ \vdots & & \\ T_{l1} & & \end{bmatrix}$

Similarly $\mu^2 = \mu^1 T^2$

So in general $\boxed{\mu^k = \mu^0 T^k}$ → this gives us a way to get the distribution at time k. To get the random variables at k, we can perform sampling from μ^k.

This is still computationally expensive as the dimensions of the vectors involved are exponential in terms of the number of variables.

Also notice that if at some time step t, u^t reaches a distribution π such that $u^t T = u^t = \pi$. Then for all subsequent time steps $u^j = \pi \quad \forall j \geq t$

π is called the stationary distribution of the Markov Chain

$X_t, X_{t+1}, X_{t+2}, \cdots$ will all follow the same distribution π.

So if we run a Markov Chain for a large number of time steps then after a point we start getting samples x_t, x_{t+1}, x_{t+2}, which are essentially being drawn from the stationary distribution. But as mentioned earlier, drawing from this distribution is computationally expensive and still intractable.

But suppose if we are given a Markov Chain such that it is easy to draw samples from it and the stationary distribution of this chain is $P(X)$, Then we could empirically estimate $E_{P(x)}[f(x)]$ as $\frac{1}{n} \sum_{i=t}^{t+n} f(X_i)$

→ the time step after we get the stationary distribution.

Theorem:—

If $X_0, X_1, \cdots X_t$ is an irreducible time homogeneous discrete Markov Chain with stationary distribution π, then

$\frac{1}{t} \sum_{i=1}^{t} f(X_i)$ converges almost surely $\to E_\pi[f(X)]$ where,

$t \to \infty$

$X \in \mathcal{X} \to \{0,1\}^n$ in our case

$X \sim \pi$

for any function $f: \mathcal{X} \to \mathbb{R}$

If further the Markov Chain is aperiodic then

$$\begin{bmatrix} \cdots T_{11} \\ T_{21} \\ T_{11} \end{bmatrix} \quad P(X_t = x_t \mid X_0 = x_0) \to \pi(X) \quad \text{as } t \to \infty$$

$\forall x, x_0 \in \mathcal{X}$

Basically means that doesn't matter the starting state, after large number of steps, the samples calculated are drawn from $\pi(X)$

Now our task is to find such a Markov Chain whose $\pi(X)$ is same as $P(X)$, moreover it is irreducible and aperiodic.

For ease of notation, instead of $X = V_1, V_2, \ldots V_m, H_1, H_2, \ldots H_n$ we will use $X = X_1, X_2, \ldots X_{n+m}$ (Basically renaming them all to X).

We will now set up a Markov Chains for RBMs.

Here X_i does not mean random vector at time step i, here it means the i^{th} element of the random vector, basically a scalar 0 or 1.

We set up the chain in this way:- i) We start, at a random where each element is set to 0 or 1 uniformly at random.

ii) Sample a value $i \in \{1, \text{to } n+m\}$ (index) using a distribution, $q(i)$, can be uniform.

iii) Fix the value of all variables except X_i. Sample a new value for X_i using the following conditional distribution

$$P(X_i = y_i \mid X_{-i} = x_{-i})$$

↳ conditioned on remaining variables.

So a transition from state x to state y is only possible if x and y differs at most by 1 variable out of $n+m$ variables. All other transitions are not possible in this chain. Hence the transition matrix is extremely sparse and can be explained like this:-

$$T_{xy} = \begin{cases} q(i)\, P(X_i = y_i \mid X_{-i} = x_{-i}) & , \text{ if } \exists\, i \{1 \text{ to } n+m\} \\ & \text{such that if } v \neq i \quad x_v = y_v \\ 0, \text{ otherwise} & \end{cases}$$

Observe that in the case of RBMs, $P(X_i = y_i \mid X_{-i} = x_{-i})$ is easily calculated. If $i \leq m$ (visible variable is selected)

$$P(V_i = v_i \mid V_{-i}, H) = P(V_i = v_i \mid H) = \sigma\left(\sum_{j=1}^{n} w_{ij} h_j + b_i\right)$$

and if $i > m$, (hidden variable is selected)

$$P(H_j = h_j \mid V, H_{-i}) = P(H_i = h_i \mid V) = \sigma\left(\sum_{i=1}^{m} w_{ij} v_i + c_j\right)$$

Hence the sampling procedure at each step is as follows:—

=) Sample i from $q(i)$

=) Sample X_i from a Bernoulli Distribution, keeping all variables the same. Both these computations are easy. Hence it is easy to sample from the chain.

Theorem (Detailed Balanced Condition):— To show that a distribution π is a stationary distribution for a Markov Chain described by the transition probabilities T_{xy}, it is sufficient to show that $\forall\ x, y$ the following condition holds:—

$$\boxed{\pi(x)\, T_{xy} = \pi(y)\, T_{yx}}$$

. So in our case if we want to prove that $P(X)$ is actually the stationary distribution of this chain then it suffices to show that $P(x) T_{xy} = P(y) T_{yx}$

To prove this, we look at 3 cases, when x and y differ in more than variables, 1 variable, are same. i) differ in more than 2 variables:— In this case $T_{xy} = 0$ and $T_{yx} = 0$, hence $P(x) T_{xy} = P(y) T_{yx} = 0$ holds true

ii) when x any y are same so $P(x) = P(y)$ and $T_{xy} = T_{yx} = T_{xx}$

So $P(x) T_{xy} = P(x) T_{xx} = P(y) T_{yx}$, holds true.

iii) when x and y differ in 1 variable then, let that variable be the i^{th} variable.

So, $T_{xy} = q(i)\, P(X_i = y_i \mid X_{-i} = x_{-i})$

$$P(x) T_{xy} = P(x_i, x_{-i}) \cdot q(i) \cdot \frac{P(y_i, x_{-i})}{P(x_{-i})}$$

$$= P(y_i, x_{-i}) \cdot q(i) \cdot \frac{P(x_i, x_{-i})}{P(x_{-i})}$$

Since other variables are same ⇕

$$= P(y) \cdot q(i) \cdot P(X_i = x_i \mid X_{-i} = x_{-i})$$

$$P(x) T_{xy} = P(y) \cdot T_{yx} \quad [\because \text{L.H.S} = P(X_i = x_i \mid X_{-i} = y_{-i})]$$

Hence the detailed balance condition holds and $P(X)$ is indeed the Stationary Distribution of this chain.

Now we need to prove that this chain is irreducible and aperiodic.
A markov chain is irreducible if one can get from any state in the sample space to any other state in a finite number of transitions.
$\forall x, y \quad \exists k > 0$ with $P(X^{(k)} = y \mid X^{(0)} = x) > 0$.

Our Markov chain is irreducible as from any starting state, any other state only differs in finite number of variables, and those variables can be flipped in one by one with a non zero probability.

A chain is called aperiodic if $\forall x$ in the sample space, the greatest common divisor of the set :-
$\{k \mid P(X^{(k)} = x \mid X^{(0)} = x) > 0)$ is 1. This set is the set of all those time steps such that the initial value is obtained again. Having a GCF of 1 essentially means that these numbers are arbitrary and are not a multiple of any number, which would otherwise make the chain periodic. This has to satisfy for all x in the sample space. Our Markov chain is aperiodic since the initial value can be retained in each time step with a non-zero prob, and gcf of all natural numbers is 1.

We return to gradient computation of RBMs.

$$\frac{\partial L(\theta)}{\partial w_{ij}} = \underset{P(H|V)}{E}[h_j v_i] - \underset{P(H,V)}{E}[v_i h_j] = \sum_H p(h|v) h_j v_i - \sum_{V,H} p(v,h) h_j v_i$$

$$= \sum_H p(h|v) h_j v_i - \sum_V p(v) \sum_H p(H|v) h_j v_i$$

$$\sum_H p(h|v) h_j v_i = \sum_{H_j} \sum_{H_{-j}} p(h_j|v)\, p(h_{-j}|v)\, h_j v_i$$

H_j: 2 values; H_{-j}: 2^{n-1} values here.

$$= \sum_{H_j} p(h_j|v) h_j v_i \underbrace{\sum_{H_{-j}} p(h_{-j}|v)}_{\text{this just sum of a distribution. Hence equal to 1.}}$$

$$= p(h_j = 1|v) v_i + p(h_j = 0|v) v_i \times 0$$

$$= p(h_j = 1|v) v_i = \sigma\left(\sum_{i=1}^{m} w_{ij} v_i + c_j\right) v_i$$

Hence $\frac{\partial L(\theta)}{\partial w_{ij}} = \sigma\left(\sum_{i=1}^{m} w_{ij} v_i + c_j\right) v_i - \sum_V p(v)\, \sigma\left(\sum_{i=1}^{m} w_{ij} v_i + c_j\right) v_i$

we can write this expression as a general for the whole matrix W.

$$\boxed{\frac{\partial L(\theta)}{\partial W} = v_0 \sigma(v^T W + c^T) - \sum_v p(v)\, v_0 \sigma(v^T W + c^T)}$$

where v is the vector $[v_1, v_2, \cdots, v_m]$ and W is an $m \times n$ matrix.

$\sigma(v^T W + c^T)$ is an element wise sigmoid of the vector inside

$$v_0 \sigma(v^T W + c^T) = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix}_{m\times 1} \cdot \sigma\left([v_1\ v_2 \cdots v_m] \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}_{m\times n} + [c_1\ c_2 \cdots c_n] \right)$$

Element wise Softmax operation

$$= \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix} \cdot \sigma\left(\left[\sum_{i=1}^{m} w_{i1} v_i + c_1,\ \sum_{i=1}^{m} w_{i2} v_i + c_2,\ \cdots,\ \sum_{i=1}^{m} w_{in} v_i + c_n \right] \right)$$

$$= \begin{bmatrix} v_1 \cdot \sigma[z_1, z_2, \cdots, z_n] \\ v_2 \cdot \sigma[z_1, z_2, \cdots, z_n] \\ \vdots \\ v_m \cdot \sigma[z_1, z_2, \cdots, z_n] \end{bmatrix}_{m\times n} \Rightarrow \underset{P(H|V)}{E}[h_j v_i]$$

Now let us derive gradients with respect to other variables :—

$$\frac{\partial L(\theta)}{\partial b_i} = \underset{P(H|V)}{E}[v_i] - \underset{P(H,V)}{E}[v_i] = \sum_H p(h|v)\, v_i - \sum_{V,H} p(h,v)\, v_i$$

$$= v_i \sum_H p(h|v) - \sum_V v_i\, p(v) \sum_H p(h|v)$$

$$\boxed{\frac{\partial L(\theta)}{\partial b_i} = v_i - \sum_V v_i\, p(v), \quad \frac{\partial L(\theta)}{\partial b} = v - \sum_v p(v)\, v}$$

Similarly $\frac{\partial L(\theta)}{\partial c_j} = \underset{P(H|V)}{E}[h_j] - \underset{P(H,V)}{E}[h_j]$

$$\frac{\partial L(\theta)}{\partial c_j} = P(h_j = 1|v) - \sum_v p(v)\, P(h_j = 1|v)$$

$$\boxed{\frac{\partial L(\theta)}{\partial c_j} = \sigma\left(\sum_{i=1}^{m} w_{ij} v_i + c_j \right) - \sum_v p(v)\, \sigma\left(\sum_{i=1}^{m} w_{ij} v_i + c_j \right)}$$

$$\nabla_c L(\theta) = \sigma(W^T v^* + c) - \mathop{E}_{P(v)}[\sigma(W^T v + c)]$$

Notice that all the gradient computations still involve this Expectation term, which is intractable. So we will set up a markov chain and sample to estimate these expectations.

RBM training with Block Gibbs Sampling :-

for all v_d (a sample) $\in D$ (set of samples).

Randomly initialize $v^{(0)}$

for $t = 0, 1, \dots k, k+1, \dots, k+r$:

  for $j = 1, 2, \dots, n$

    Sample $h_j^{(t)} \sim P(h_j | v^{(t)})$ $\Rightarrow$ given by $P(h_j = 1 | v)_m = \sigma\left(\sum_{i=1}^{} w_{ij} v_i + c_j\right)$

  for $i = 1, 2, \dots, m$

    Sample $v_i^{(t+1)} \sim P(v_i | h^{(t)})$

Notice that instead of going one by one from a sample to another, we make steps in blocks (first all hidden, then all visible) because we have ~~either~~ otherwise run the chain for large iterations. Hence each time step t actually consists of multiple time steps.

~~$W = W + \eta(\sigma(W$~~

$$W = W + \eta\left[v_d \sigma(v_d^T W + c^T) - \frac{1}{r}\sum_{p=k+1}^{k+r} v^{(p)} \sigma(v^{(p)T} W + c^T)\right]$$

notice that these terms are based on the actual sample v_d

these terms are based on the samples of the Markov Chain.

$$b = b + \eta\left[v_d - \frac{1}{r}\sum_{p=k+1}^{k+r} v^{(p)}\right]$$

$$c = c + \eta\left[\sigma(W^T v_d + c) - \frac{1}{r}\sum_{p=k+1}^{k+r} \sigma(W^T v^{(p)} + c)\right]$$

In practice, Gibbs Sampling can also be very inefficient because for every step of stochastic gradient descent, we need to run the Markov Chain for many steps.

In practice we use K-contrastive Divergence to train RBMs.

Contrastive Divergence uses the following idea:-

=> Instead of starting the Markov Chain at a random point, start from v_d where v_d is the current training instance.

=> Run Gibbs sampling for k steps, and denote the sample at the k^{th} step by $\tilde{v}$. Replace the expectation by a point estimate

$$E_{p(v,h)}[v_i h_j] \approx \sigma(W_j^T \cdot \tilde{v} + c_j) \cdot \tilde{v}_i$$

(W_j: j^{th} column of W)

Hence the gradients are as follows:-

$$W = W + \eta\left[v_d \cdot \sigma(v_d^T W + c^T) - \tilde{v} \cdot \sigma(\tilde{v}^T W + c^T)\right]$$

$$b = b + \eta\left[v_d - \tilde{v}\right]$$

$$c = c + \eta\left[\sigma(W^T v_d + c) - \sigma(W^T \tilde{v} + c)\right].$$

The higher the value of k, the less biased the estimate of the gradient will be.

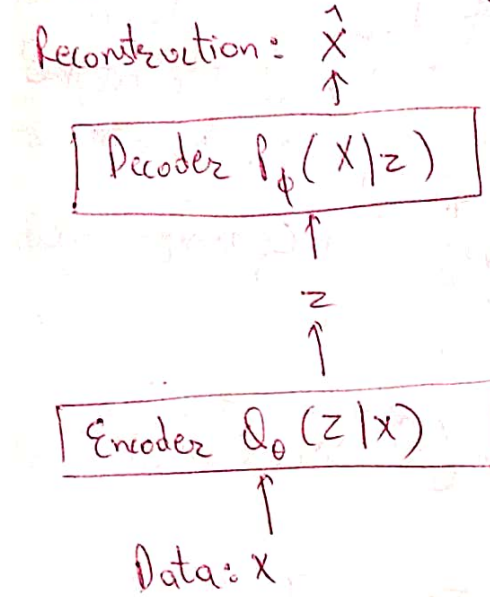
Variational Autoencoders:-

Before VAE's some info about autoencoders. An autoencoder contains an encoder which takes the input X and maps it to a hidden representation, then the decoder takes this hidden representation and tries to reconstruct the input from it as $\hat{X}$. The loss between $\hat{X}$ and X is used to train the params of the network (Unsupervised task). Can we use Autoencoders for generative purposes? Well given a hidden representation, we can produce an $\hat{X}$. But it is a deterministic procedure so variability will be less. Moreover, out of all the possible hidden representations, only a few are likely to be produced by our data. So deciding a good representation (hidden) for generation is still a question. In case of RBM's we have and approximate for the joint dist so we can find the likely hidden representations.

Variational auto encoders have similar/same structure as autoencoders, but they learn a distribution over the hidden variables.

visible variables will be referred as X, and hidden variables will be referred as Z.

The architechture and goals are as follows :-



Goal 1 :- Learn a Distribution over the latent variables $Q(z|X)$.

Goal 2 :- Learn a distribution over the visible variables $P(X|z)$

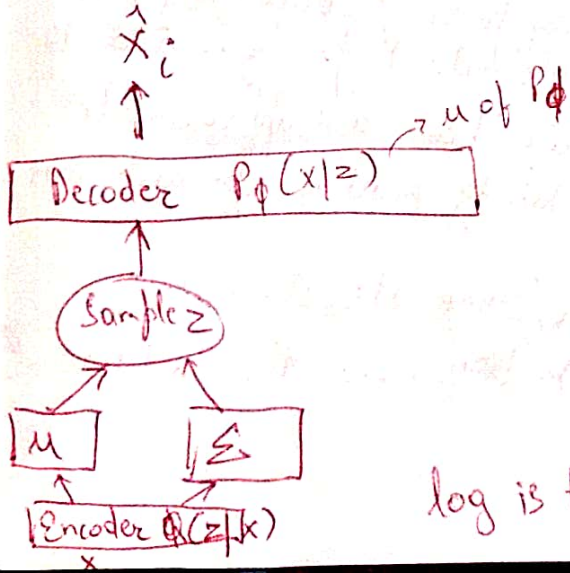
Notice that both the encoder and decoder learn distributions rather than the actual hidden / visible representations.

θ: the parameters of the encoder network.

ϕ: the parameters of the decoder network.

In the encoder part, we assume that the latent variables come from a standard normal distribution $N(0, I)$ and the job of the encoder is to predict the parameters of this distribution. The input of the encoder is the visible variables X, and it has to give the parameters (μ, Σ) of the distribution $Q_\theta(z|X) \sim N(\mu, \Sigma)$, and the reference distribution is $N(0, I)$

The job of the decoder is to predict a probability over X : $P(X|z)$ We assume that $P(X|z)$ is a Gaussian Distribution with unit variance. The decoder predicts the mean of the distribution, $P_\phi(X|z)$.



The loss function is as follows :-

we want to maximize the likelihood the probability of the data. Basically

$$P(X_i) = \int P(z)\, P(X_i|z)\, dz \quad \text{multivariate integral.}$$

$$= E_{z \sim Q_\theta(z|X_i)} [\log(P_\phi(X_i|z))]$$

$P(z)$ only.

log is taken for numerical stability.

But we want to also ensure that $Q_\theta(z|x_i)$ does not diverge much from $N(0,I)$. So a KL divergence term between these two distributions is also taken. So the total loss :—

$$L(x_i) = -\mathbb{E}_{z\sim Q_\theta(z|x_i)}[\log(P(x_i|z))] + KL(Q_\theta(z|x_i) \,||\, N(0,I))$$

We need to add this divergence term so that the network does not cheat. Otherwise it might just learn a unique representation for each datapoint and decode it as a mapping without learning anything. So we predict a distribution $Q_\theta(z|x_i)$ such that a sample from this distribution should be able to reconstruct the original datapoint with high probability. We assume that $P(z) \sim N(0,I)$ because any distribution in d dimension may be generated with a set of d normally distributed variables by using a sufficiently complex function. So the decoder learns this function to arrive at $P_\phi(x_i|z)$ using only normally distributed variables.

It can be shown that $KL[N(\mu(x), \Sigma(x)) \,||\, N(0,1)]$

$$= \frac{1}{2}\left(tr(\Sigma(x)) + \mu(x)^T[\mu(x) - k - \log\det(\Sigma(x))\right]$$

where $\Sigma(x)$ and $\mu(x)$ are the μ and Σ calculated by the decoder for $Q_\theta(z|x_i)$, it is expressed like this since it is a function of X. k is the dimensionality of the latent variables. This is easily computable. But $\mathbb{E}_{z\sim Q_\theta(z|x)}[\log P(x|z)]$ is still a bottleneck since it involves a multivariate integral. But in VAE's we approximate this expectation using a single z sampled from $N(\mu(x), \Sigma(x))$. Hence $\log P(x|z) = (-\frac{1}{2}\|x_i - \mu(z)\|^2$

$\mu(z)$ is the mean of the distribution $P_\phi(x|z)$ calculated by the decoder. It is of this form as $\mu(z)$ depends on z, moreover the expression can be derived from the PDF of Normal distribution

> we assume $P_\phi(x|z)$ is normal with unit variance.

Hence the effective objective function is:—

$$\underset{\theta,\phi}{\text{minimize}} \sum_{n=1}^{N}\left[\frac{1}{2}\left[tr(\Sigma(x)) + \mu(x)^T[\mu(x) - k - \log\det(\Sigma(x))\right] + \|x - \mu(z)\|^2\right]$$

Observe that $\mu(x)$ and $\Sigma(x)$ depend on θ, while $\mu(z)$ depends on ϕ.

But since the network involves sampling z. It is not end to end differentiable. To make it differentiable we sample ϵ in the input step itself. So we calculate z like this :-
Sample $\epsilon \sim N(0, I)$ in the input stage. Then

$$z = \mu(x) + A(x)\epsilon$$, where $A(x)$ is such that $\Sigma(x) = A(x)A(x)^T$

This is because z will now follow $N(\mu(x), AIA^T)$
$= N(\mu(x), \Sigma(x))$

Now the parameters θ and ϕ can be learnt using gradient descent and backpropogation.
In abstraction, we calculate z in the above way.
In Generation, we sample $z \sim N(0, I)$ and just feed it to the decoder, since model was trained to converge $Q_\theta(z|x) \sim N(0, I)$

Autoregressive Models :- these models do not contain any latent variables They also learn a joint distribution over X. For us, $X \in \{0,1\}^n$
AR models do not make any independence assumptions but use the default factorization of $p(x) :- p(x) = \prod_{i=1}^{n} p(x_i | x_{<i})$

$= p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_2, x_1) \cdots p(x_n|x_{<n})$

In AR models, these factors are parameterized using a neural network and these parameters are learnt to eventually learn the factors which are used to construct the joint distribution.
The network is set up such that, at the output layer, we predict the n factors/conditional probability distributions. The i^{th} neuron in the output layer gives $p(x_i = 1 | x_{<i})$. But the network should be constructed such that i^{th} output should only be connected to the previous $i-1$ inputs. Since it only depends on these $i-1$ variables.
In NADE (Neural Autoregressive Density Estimator), a hidden representation is calculated using only the $i-1$ inputs, then this hidden representation is used to calculate $p(x_i | x_{<i})$. So at the i^{th} step :-

$$h_i = \sigma(W_{<i} x_{<i} + b)$$ where $h_i \in \mathbb{R}^d$ and $W \in \mathbb{R}^{d \times n}$, $b \in \mathbb{R}^d$

So only first $i-1$ columns are used to calculate h_i. Then

$y_i = p(x_i | x_{<i})$ where $\boxed{y_i = \sigma(V_i h_i + c_i)}$ $\quad V_i \in \mathbb{R}^d$, $c_i \in \mathbb{R}$

Notice that for calculating h_1, no input is seen, hence we make h_1 as a learnable parameter of the network. So in total there are :- $d \times n$ (for W) $+ d$ (for b) $+ n \times d$ (for V_is) $+ n$ (for c_is)

this quantity is linear wrt n, instead of exponential. This is because we use weight tying. We use the same parameters W and b at each step. In practice, we mask the input appropriately to select the correct columns of W.

=) We can use the cross entropy loss for each node at the output as the loss for training. Since each output node predicts a distribution of a binary variable we can calculate its cross entropy loss and take the sum of all the losses coming from each node.

=) Notice that these models are not meant for abstraction by design. This is because we don't get a hidden representation for the whole input together from the models. We can of course devise a way to merge the individual hidden representation to calculate an abstraction

=) For generation, first the model gives $p(x_1)$, using this Bernoulli dist, we can sample x_1, and use the sampled value to calculate h_2 and then $y_2 (p(x_2|x_1))$. Then x_2 can be sampled and then x_1 and x_2 are used to calculate y_3. This process is repeated for all n vars.

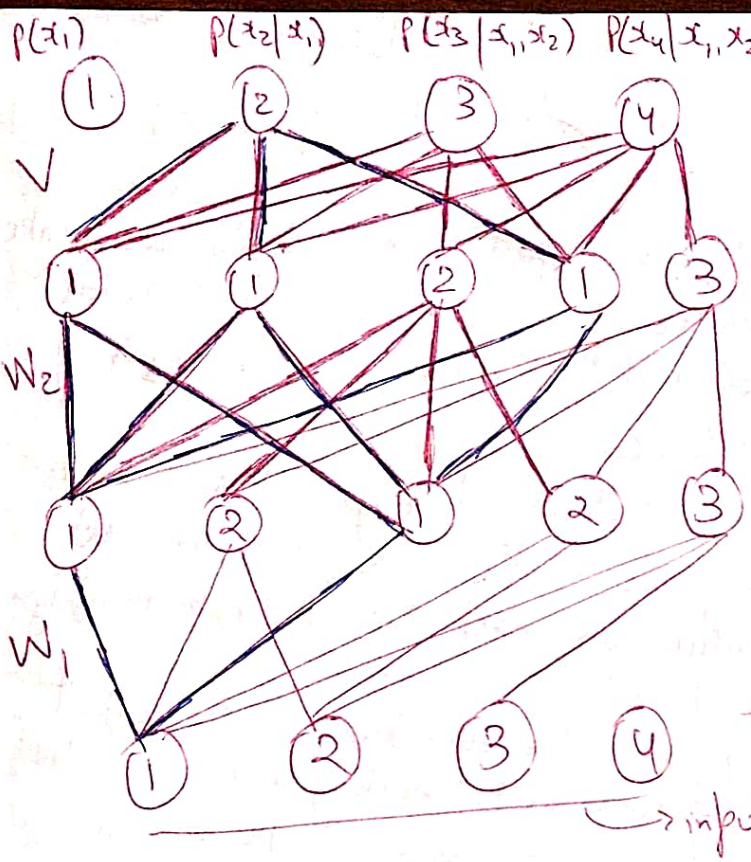
This process is sequential and slow. However it can be sped up by reusing calculations made in the previous step :-

$$W_{<i} x_{<i} = \underbrace{W_{<i-1} x_{<i-1}}_{\text{already calculated in previous step}} + W_{:i-1} x_{i-1}$$

$W_{:i-1}$ is the $i-1^{th}$ col. of W and x_{i-1} is a scalar.

Another autoregressive Model is MADE (Masked Autoencoder Density Estimator) which also considers the natural factorization. It uses the same architecture as an autoencoder, but introduces masks in the hidden layer computations which makes sure that the final i^{th} output depends on the previous $i-1$ inputs.

We start by assuming some ordering on the inputs and just number them from 1 to n.



the output only accepts input from nodes numbered strictly less that it self.

the hidden layer accepts input from nodes numbered less than or equal to itself.

Observe the blue weights. For output 2, the initial input is only ①, which is made sure by this network.

hidden.

=) Each node is randomly assigned a number between 1 to n-1

⇒ only for understanding, in practices, only masking is done.

Let masked $W_2 \in$

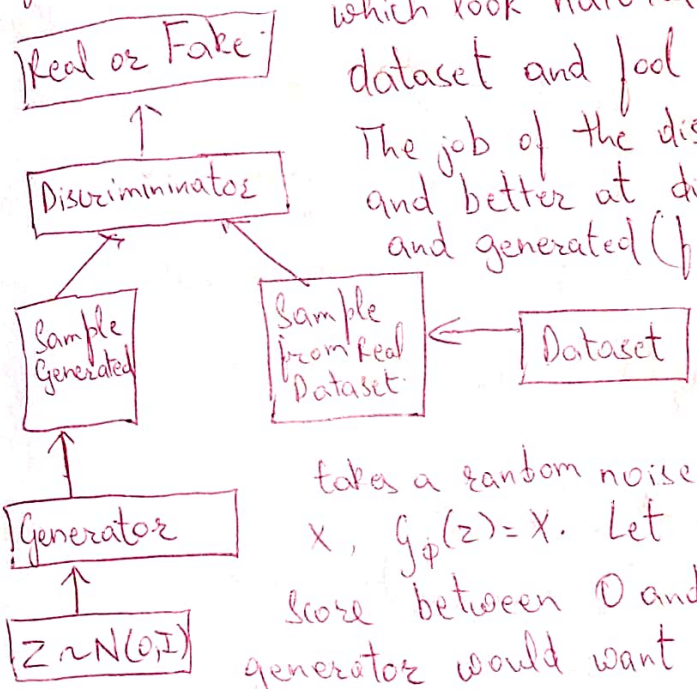
$$\begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} & \cdots & w_{15}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} & \cdots & w_{25}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{51}^{(2)} & w_{52}^{(2)} & \cdots & w_{55}^{(2)} \end{bmatrix}_{5\times 5} \odot \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_5 \end{bmatrix}$$

W_2. mask. ↑ input from first hidden layer

Since the first node of 2nd hidden layer depends only on 1st and 3rd node of 1st hidden layer, hence the mask contains [1 0 1 0 0] as the first row. The 3rd row of 2nd hidden layer depends only on first 4 nodes of 1st hidden layer. So the 3rd row of the mask is [1 1 1 1 0].

The loss function is same sum of cross entropy loss, and it follows the same autoregressive generation process as in NADE

Generative Adversarial Networks:- Instead of learning a joint distribution over the data random variables, GANs take a different approach to this problem where the idea is to sample from a simpler tractable distribution; say $z \sim N(0, I)$; and then learn a complex transformation from this to the training distribution. The training of the network is like a two player game.



The job of the generator is to produce images which look natural as if sampled from the real dataset and fool the discriminator.

The job of the discriminator is to get better and better at distinguishing between true images and generated (fake) images.

Let G_ϕ be the generator and D_θ be the discriminator. G_ϕ takes a random noise vector $z \sim N(0, I)$ and produces x, $G_\phi(z) = x$. Let the discriminator output a score between 0 and 1. For a given z, the generator would want to maximize $\log D_\theta(G_\phi(z))$ or minimize $\log(1 - D_\theta(G_\phi(z)))$. Since we want to minimize this loss overall z, we minimize the expectation of this loss. So for the generator

$$\min_\phi \; \mathbb{E}_{z \sim N(0,I)} \left[\log(1 - D_\theta(G_\phi(z))) \right]$$

The discriminator on the other hand has to assign a high score to real images and a low score to generated images. It should do for all possible real images and all possible fake images.

$$\max_\theta \; \mathbb{E}_{x \sim p_{data}} [\log D_\theta(x)] + \mathbb{E}_{z \sim N(0,I)} [\log(1 - D_\theta(G_\phi(z)))],$$ combining both the objectives of the generator and discriminator:-

$$\min_\phi \max_\theta \; \mathbb{E}_{x \sim p_{data}} [\log D_\theta(x)] + \mathbb{E}_{z \sim N(0,I)} [\log(1 - D_\theta(G_\phi(z)))]$$

So the overall training proceeds by alternating between these two steps:-

i) Gradient Ascent on Discriminator

ii) Gradient Ascent on Generator.

In practice, the above generator objective does not work well and we use a different objective:-

we minimize $-\log(D_\theta(G_\phi(z)))$.

GAN Training :-

for number of iterations.

for k steps

Sample m noise samples $\{z^{(1)}, z^{(2)}, \cdots, z^{(m)}\}$

Sample m examples $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$ from data.

update the discriminator by ascending its gradient:-

$$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} \left[\log D_\theta(x^{(i)}) + \log(1 - D_\theta(G_\phi(z^{(i)})))\right]$$

Sample m noise samples $\{z^{(1)}, z^{(2)}, \cdots, z^{(m)}\}$

~~Sample m examples $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$ from data.~~

update the generator by ascending its gradient:-

$$\nabla_\phi \frac{1}{m} \sum_{i=1}^{m} \log D_\theta(G_\phi(z^{(i)}))$$

Architecture :- For discriminator any CNN based classifier with 1 class at the output can be used (eg: VGG, ResNet).

For the Generator, we have to go from a random vector to an image, called transpose convolution.

Pooling layers are replaced by strided convolutions (discriminator) and fractional-strided convolutions (generator). ReLU is used in generator except at output where tanh is used as activation function. Use LeakyReLU in the discriminator for all layers.

It can be formally proved that when solve the above objective, the distribution learned by the Generator over the variables is same as the actual distribution.

Theorem:- The global minimum of the virtual training criterion $C(G) = \max_D V(G,D)$ is achieved if and only if $p_g = p_{data}$.

So even though GANs do not model $P(X)$, they implicitly learn $P(X)$ through an indirect modified objective.

Summary:-

	RBMs	VAEs	AR models	GANs
Abstraction	Yes	Yes	No	No
Generation	Yes	Yes	Yes	Yes
Compute $P(x)$	Intractable	Intractable	Tractable	No
Sampling	Gibbs Sampling	Fast	Slow	Fast
Type of GM	Undirected	Directed	Directed	Directed
Loss	KL Divergence	KL Divergence	KL Divergence	Jensen-Shannon Divergence
Assumptions	X independent given Z	X independent given Z	None	Not applicate since $P(x)$ is not computed.
Samples	Bad	OK	Good	Good (best)

Recent works combine these models eg Adversarial Autoencoders, Pixel GAN Autoencoders, etc.