DEEP LEARNING

UNIT5 ASSIGNMENT

—Done By
Jahnavi Darbhamulla
RA2011047010011

DEEP LEARNING UNIT 5 - ASSIGNMENT

1) What are Attention models in Neural Networks?

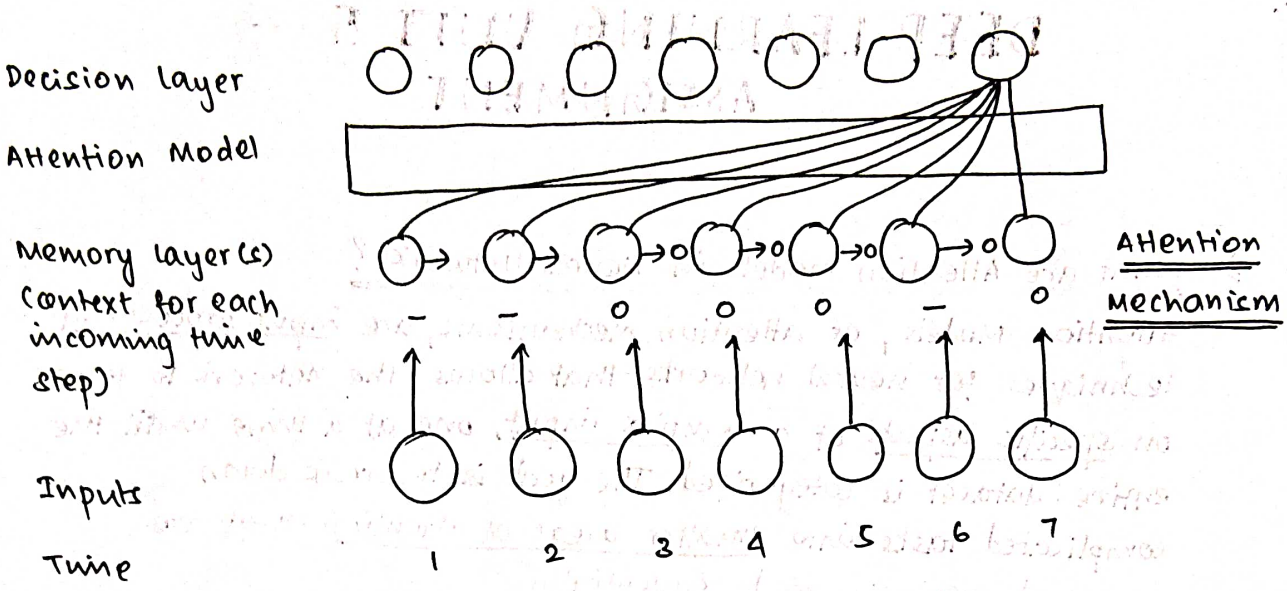
Attention Models, or attention mechanisms, are input processing techniques for neural networks that allows the network to focus on specific aspects of a complex input, one at a time until the entire dataset is categorised. The goal is to break down complicated tasks into smaller areas of attention that are processed eventually and sequentially.

How Attention Models work?

In broad strokes, attention is expressed as a function that maps a query and "a set" of key value pairs to an output. One in which the query, keys, values and final output are all vectors. The output is then calculated as a weighted sum of the values, with the weight assigned to each value expressed by a compatibility function of the query with the corresponding key value.

In practice, attention allows neural networks to approximate the visual attention mechanism humans use. Like people processing a new scene, the model studies a certain point of an image with itense, "high resolution" focus, while percieving the surrounding areas in "low resolution", then adjusts the focal point as the network begins to understand the scene.

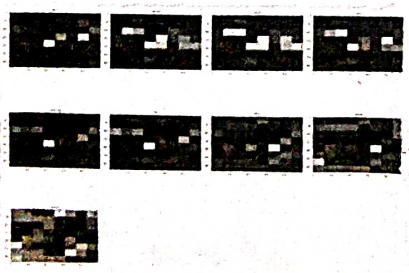
While this is a powerful technique for improving computer vision, the most work so far with attention mechanisms has focused on Neural Machine Translation. Here, the meaning of a sentence is mapped into a fixed length vector, which then generates a translation based on that vector as a whole.



(i) Attention over images

The Attention Mechanism is a complex ability that human beings possess. When people recieve information, they can consciously ignore some of the main information while ignoring other secondary information.

In the case of Image captioning Rather than compressing an entire image into a static representation, the Attention mechanism allows for salient features to dynamically come to the foretront as and when needed. This is especially important when there is a lot of clutter in the image.



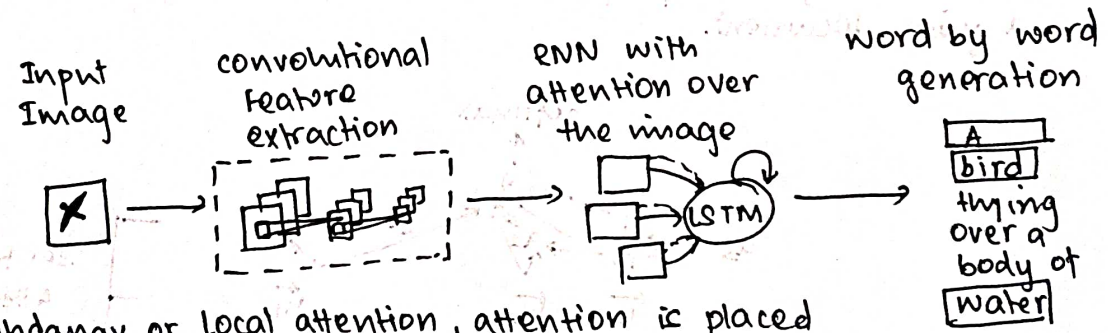
Our Aim would be to generate a caption like "two white dogs are running on the snow". To accomplish this we see how to implement a specific type of Attention mechanism called Bahdanaus Attention or local Attention.

The encoder-decoder image captioning system would encode the image, using a pre-trained convolutional Neural Network that would produce a hidden state. Then, it would decode this hidden state by using an LSTM and generate a caption.

For each sequence element, outputs from previous elements are used as inputs, in combination with new sequence data. This gives the RNN networks a sort of memory which might make captions more informative and contextware.

But RNNs tend to be computationally expensive to train and evaluate, so in practice, memory is limited to just a few elements. Attention models can help address this problem by selecting the most relevant elements from an input image.

With an Attention Mechanism, the image is first divided into n parts and we compute an image representation of each when the RNN is generating a new word, the attention mechanism is focusing on the relevant part of the image, so the decoder only uses specific parts of the image.



In Bahdanav or Local attention, attention is placed only on a few source positions. As Global focuses on all source side words for all target words it is computationally very expensive. To overcome this deficiency local attention chooses to focus on a small subset.
Local attention is to reduce the cost of the attention mechanism calculation. In calculation, the local attention is not to consider all the words on the source language side, but to predict the position of the source language end to be aligned at the current decoding according to a prediction function and navigate through the context window.

(ii) Hierarchial Attention

Contrary to most text classification implementations, a Hierarchial Attention network (HAN) also considers the hierarchial structure of documents and includes an attention mechanism that is able to find the most important words and sentences in a document while taking the context into consideration.

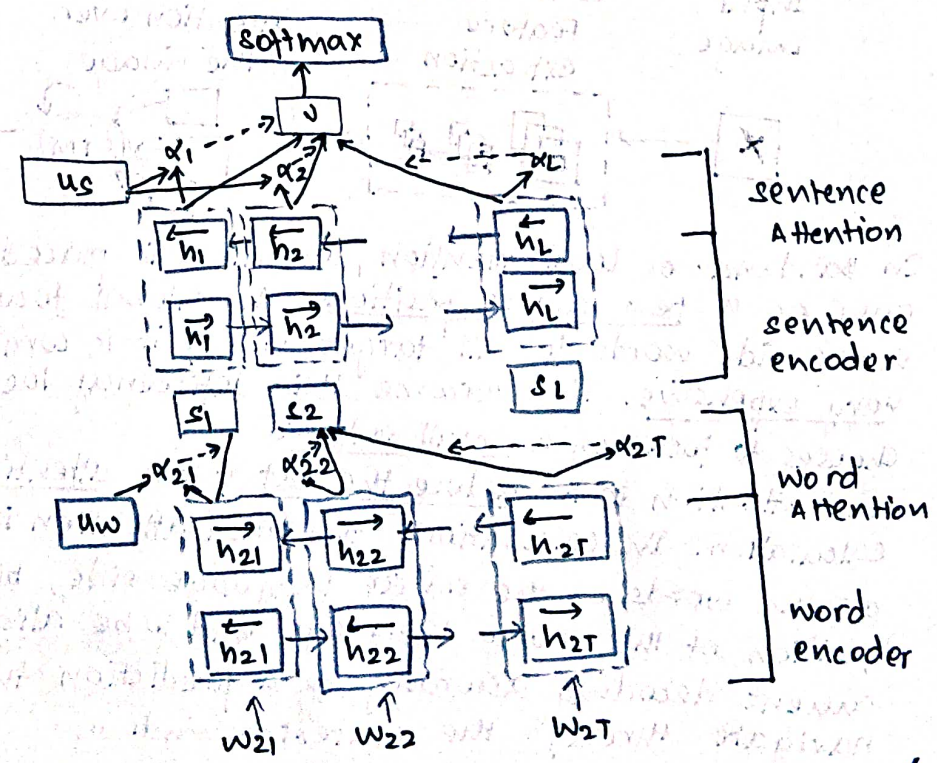
HAN finds a solution that previous works did not consider.

→ Not every word in a sentence and every word in a document are equally important to understand the main message of a document.

→ The changing meaning of a word depending on the context needs to be taken into consideration.
for example: The meaning of the word "pretty" can change depending on the way it is used.

"The Bouquet of flowers is pretty"]
"The food is pretty bad"] both have different meanings wrt to context.

In this way, HAN performs better in predicting the class of a given document.



first, the network considers the hierarchial structure of documents by constructing a document representation by building representations of sentences and then aggregating those into document representation.

sentence representations are built by first encoding the word of a sentence and then applying the attention mechanism on them resulting in a sentence vector.

Document representations is built in the same way, however it only recieves the sentence vector of each sentence of the document as an input.

The model consists of

→ The encoder which returns relevant contexts
→ The attention mechanism, which computes importance weights of these contexts as one vector.

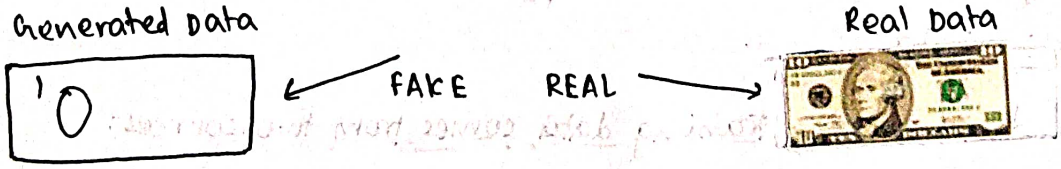
3) (1) Explain Generative Adversial Network with a neat diagram? Also, mention any three applications of GAN?

A generative adversial Network (GAN) has 2 parts :

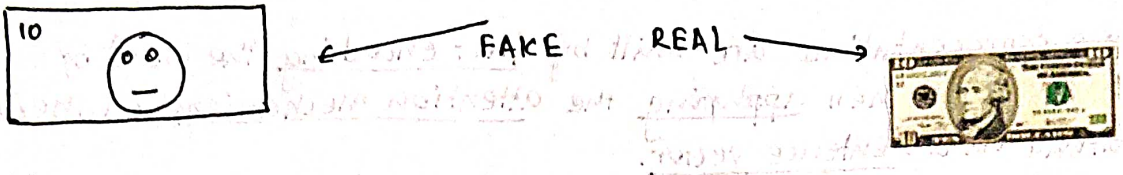
The generator learns to generate plausible data. The generated instances become negative training examples for the discriminator.

The discriminator learns to distinguish the generators' fake data from real data. The discriminator penalises the generator for producing implausible results.

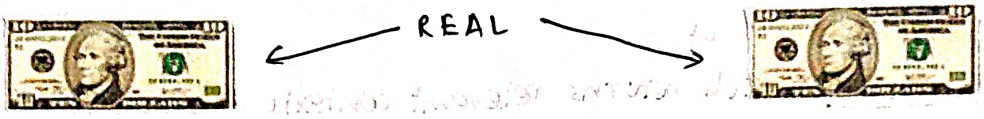
when training begins, the generator produces obviously fake data, and the discriminator quickly learns to tell its fake.



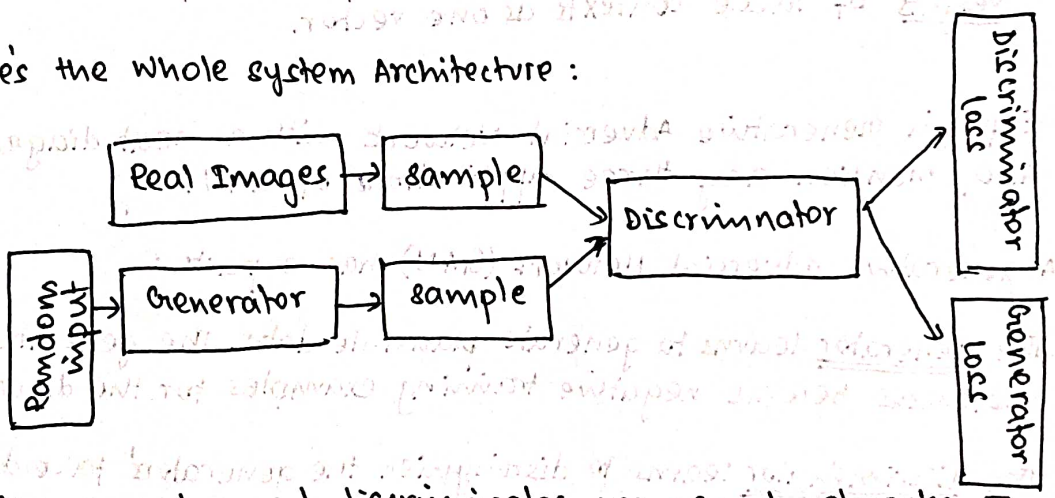
As training progresses, the generator gets closer to producing output that can fool the discriminator.



finaly if generator training goes well, the discriminator gets worse at telling the difference between real & fake. It starts to classify fake data as real, and its accuracy decreases.



Here's the whole system Architecture :



Both the generator and discriminator are neural networks. The generator output is connected directly to the discriminator input. Through backpropgation, the discriminator's classification provides a signal that the generator uses to update its weights.

Discriminator

The discriminators training data comes from two sources:

- Real data instances, such as real pictures of people. The discriminator uses these instances as positive examples during training.
- Fake Data instances, created by the generator. These are the negative examples.

Generator

The generator part of GAN learns to create fake data by incorporating feedback from the discriminator. It learns to, make the discriminator classify its output as real.

Generator training requires tighter integration between the generator and discriminator than discriminator training requires.

The portion of GAN that trains generator includes:

- → random input
- → generator network, which transforms the random input into a data instance.
- → discriminator network, which classifies the generated data
- → discriminator output
- → generator loss, which penalises the generator for failing to fool the discriminator.

Applications of GAN

(1) Bringing Monalisa Back to Life

Who imagined that one day we will be able to see the expressions of the Italian noblewoman Lisa Gherardini from the famous potrait of Mona Lisa painted by the Italian artist Leonardo da Vinci. Yes, this was made possible by Deepfake.

A team of researchers at Samsung AI, Moscow used a ML system that includes few-shot adversial learning to create expressions in the potrait. The system performs a lengthy meta-learning on a large dataset of videos and after was able to frame a few one-shot learning of neural talking head models of previously unseen people.

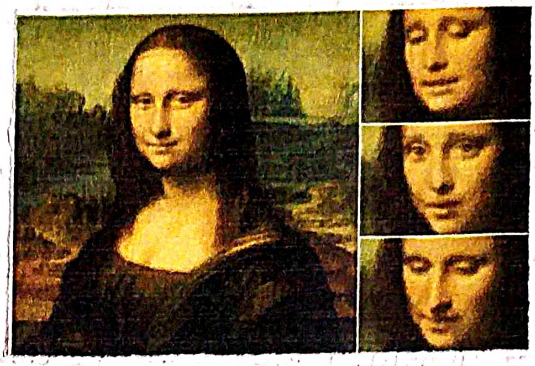
(2) Making Photos better

With the help of GAN, a team of researchers at the top social media Giant, facebook created an approach known as (ExGANs) that can produce photo realistic personalised in-painting results that are not only perpetual but also semantically plausible. The GAN masters to fill in the missing regions of a given image while discriminator n/w learns to judge the difference b/w both in-painted and real images.

(3) Face Aging

Grigory Antipov in their 2017 titled "Face Aging with conditional Generative Adversial Networks" use GANs to generate photographs of faces with different apparent Ages from younger to older This is also used for deaging photographs of faces.

(2) Image Enhancement (Making Images Better)



(1) Mona lisa with different expressions

(3) Face Aging

(ii) How do you train a GAN? Explain with proper algorithm and mathematical formulations.

Algorithm

for number of training iterations do
  for k steps do
    - Sample Minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$
    - Sample minibatch of m examples $\{x^{(1)}, \ldots x^{(m2}\}$ from data generating distribution $p_{data}(x)$.
    - Update the discriminator by ascending its stochastic gradient

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$$

  end for

- sample minibatch of m noise examples/samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Update the generator by descending its stochastic gradient.

$$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D(G(z^{(i)}))\right).$$

end for

The gradient based updates can use any standard gradient-descent based learning rule. We used momentum.

Algorithm Explanation

The outer loop of the Algorithm involves iterating over steps to train the Models in the architecture. One cycle through this loop is not an epoch, it is a single update compromised of a specific batch updates to the discriminator and generator Models.

An epoch is defined as one cycle through a training dataset, where the samples in a training dataset are used to update the model weights in mini-batch.

In case of GAN, the no. of training iterations must be defined based on the size of your training dataset and batch_size.

In case of a Dataset with 100 samples, A batchsize of 10, and 500 training epochs, we would first calculate the number of batches per epoch and use this to calculate training iterations.

Next, we can use that one iteration of training results in possibly multiple updates to the discriminator and one update to generator, where number of updates to discriminator is a hyperparameter set to 1.

we therefore summarise the training algorithm with python pseudocode as follows:

```
# gan training algorithm
def train_gan (dataset, n_epochs, n_batches):
    # calculate the number of batches per epoch
    batches_per_epoch = int (len(dataset) / n_batch)
    # calculate the number of training iterations
    n_steps = batches_per_epoch * n_epochs
    # gan training algorithm
    for i in range(n_steps):
    # update the discriminator model
    # ...
    # update the generator model
    # ...
```

(3) (i) What do you mean by Restricted Boltzmann Machines?

A restricted term refers to that we are not allowed to connect the same type layer to each other. In other words, the 2 neurons of the input layer or hidden layer can't connect to each other. Although the hidden layer and visible layer can be connected to each other.

As in this machine, there is no output layer so the question arises how we are going to identify, adjust the weights and how to measure if our prediction is accurate or not. This is where Restricted Boltzmann Machine comes to place.

The RBM Algorithm was proposed by Geoffery Hinton (2007), which learns probability distribution over its sample training data inputs. It has wide applications in different areas of supervised/unsupervised Machine learning algorithm such as feature engineering, dimensionality reduction, classification and topic modelling.

consider the Movie Rating recommender system, Movies like Avengers, Avatar and Interstellar have strong associations with the latest fantasy and science fiction Factor. Based on the user rating RBM will discover latent features that can explain the activation of movie choices.

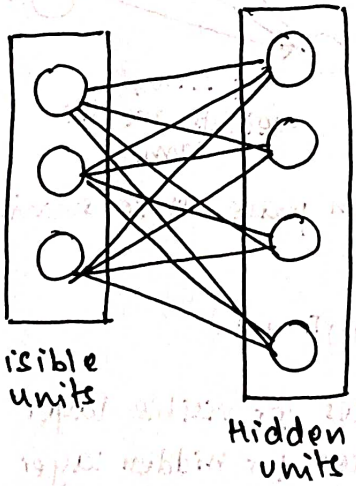
In short, RBM describes variability among correlated variables of input dataset in terms of potentially lower numbers of unobserved variables.

The energy function is given by,

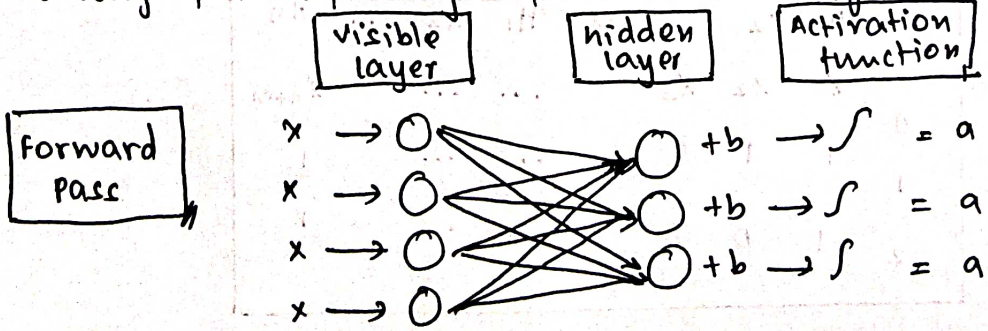
$$E(v,h) = -a^T v - b^T h - v^T W h$$

(ii) How RBMs are used as stochastic Neural Networks?

RBM is a stochastic Nevral Network which means that each neuron will have some random behaviour when activated. There are 2 other layers of bias units (hidden bias and visible bias) in an RBM. This is what makes RBMs different from auto encoders.



The reconstructed input is always different from the actual input as there are no connections among visible nodes and therefore no way of transferring information among themselves.



This is the first step in training an RBM with multiple inputs The inputs are multiplied by weights and then added to bias. The result is the passed through sigmoid function.

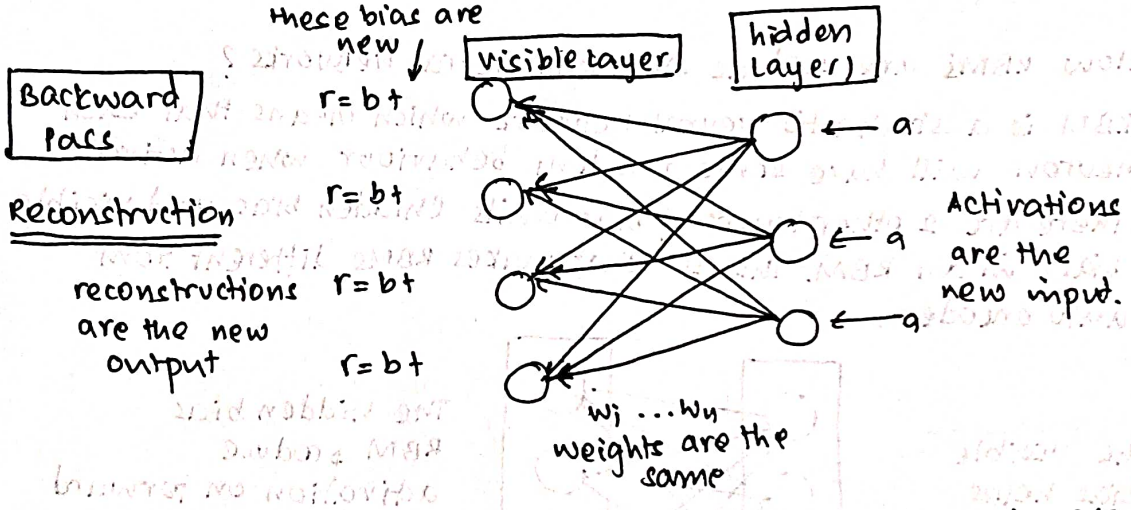
The sigmoid function is given by

$$s(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$$

so the equation that we get in this step would be

$$h^{(1)} = s(v^{(0)T} W + a)$$

$h(1)$ → column matrice for hidden layer
$v(0)$ → column matrice for visible layer



Reverse phase or Reconstruction phase. It is similar to first pass but in the opposite direction.

$$v^{(1)} = s(h^{(1)} W^T + a)$$

$v(1)$ → column matrices for visible layer
$h(1)$ → column matrices for hidden layer

(iii) How to setup a Markov Chain Model for RBM's?

	V_1	V_2	. .	V_m	H_1	H_2	...	H_n
	X_1	X_2	X_3		...	...		X_{n+m}
0	1	1	0		...	...		1
1								
2	1	0	1		...	...		1

- We begin by defining our Markov chain
- $x = \{v, H\} \in \{0,1\}^{n+m}$, At time step 0 we create a random vector $X \in \{0,1\}^{n+m}$
- A time-step 1, we transition to a new value of x

⇒ How do we do this transition?

	v_1	v_2	...	v_m	H_1	H_2	...	H_n
	x_1	x_2	x_3					x_{n+m}
0	1	1	0		...	...		1
1	1	0	0		...	...		1
2	1	0	1		...	...		1
3	1	0	1		...	...		1
4	1	0	1		...	...		0
⋮								

- We need to transition from a state $x = x \in \{0,1\}^{n+m}$ to $y \in \{0,1\}^{n+m}$
- sample a value $i \in \{1 \text{ to } n+m\}$ using a distribution $q(i)$
- fix the value of all variables except x_i
- sample a new value x_i using the following conditional distribution.

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

- Repeat the above process for many many time steps.

⇒ where is our transition matrix? and what does it contain?

we define T such that

$$P_{xy} = \begin{cases} q(i)\, P(y_i \mid x_{-i}) & \text{if } \exists i \in x \text{ so that } \forall u \in X \text{ with } v \neq i,\ x_v = y_v \\ 0, & \text{otherwise} \end{cases}$$

where $q(i)$ is probability that x_i is the random variable whose value transitions while the value of x_{-i} remains the same.

The second term $P(x_i = y_i \mid x_{-i})$ essentially tells us that given the value of the remaining random variable what is the probability of x_i taking on a certain value.

The Transition Matrix T is a sparse Matrix allowing only certain transitions.

The Transition Matrix T contains the probability of transitioning from any state x to any state ~~state~~ y.

⇒ How is it easy to create this chain?

At each step we are changing only one of the n+m random variables using the following probability.

$$P(x_i = y_i \mid x_{-i} = x_{-i}) = \frac{P(x)}{P(x_{-i})}$$

considering the cases,

$i <= m$ (we have decided to transition the value of one of the visible variables v_1 to v_m)

Then, $P(v_i = y_i \mid V_{-i}, H) = P(v_i = y_i \mid H) = \begin{cases} z & \text{if } y_i = 1 \\ 1-z & \text{if } y_i = 0. \end{cases}$

where $z = \sigma\left(\sum_{j=1}^{n} w_{ij} h_j + c_i\right)$

The above probability is easy to compute using sigmoid function.

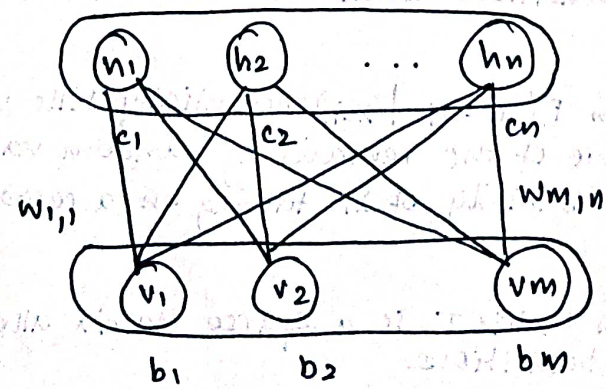
Once you compute the above probability, with probability z you will set the value of v_i to 1 and with probability 1-z you will set it to 0.

So essentially at every time step you sample a i from uniform distribution (q_i)

And then sample a value of $v_i \in \{0,1\}$ using the distribution Bernoulli (z).

Hence, it is easy to create this chain starting from any x_0.

(iv) How to train RBM's using Gibbs sampling?



$W \in \mathbb{R}^{m \times n}$

$V \in \{0,1\}^m$

$$E(v,H) = -\sum_i \sum_j w_{ij} v_i h_j - \sum_i b_i v_i - \sum_j c_j h_j.$$

$$\frac{\partial L(\theta)}{\partial w_{ij}} = -\sum_H p(H|v) \frac{\partial E(v,H)}{\partial w_{ij}} + \sum_{v,H} p(v,H) \frac{\partial E(v,H)}{\partial w_{ij}}$$

$$= \sum_H p(H|v) h_i v_j - \sum_{v,H} p(v,H) h_i v_j$$

$$= E_{p(H|v)}[v_i h_j] - E_{p(v,H)}[v_i h_j]$$

Algorithm: RBM Training with Block Gibbs sampling

Input: RBM $(v_1, .., v_m, H \ldots H_n)$, training batch D
Output: Learned parameters W, b, c

init W, b, c

for all $v \in D$ do
  Randomly initialise $v^{(0)}$
  for $t = 0, \ldots k, k+1, \ldots k+r$ do
    for $i = 1 \ldots n$ do
      sample $h_i^{(t)} \sim p(h_i | v^{(t)})$
    end
    for $j = 1 \ldots m$ do
      sample $v_j^{(t+1)} \sim p(v_j | h^{(t)})$
    end
  end
  $W \leftarrow W + \eta \nabla_W \alpha(\theta) [\sigma(W v_d + c) v_d^T - \frac{1}{r} \sum_{t=k+1}^{k+r} \sigma(W v^{(t)} + c) v^{(t)T}]$
  $b \leftarrow b + \eta \nabla_b \alpha(\theta) [v_d - 1/r \sum_{t=k+1}^{k+r} v^{(t)}]$
  $c \leftarrow c + \eta \nabla_c \alpha(\theta) [\sigma(W v_d + c) - 1/r \sum_{t=k+1}^{k+r} \sigma(W v^{(t)} + c)]$
end