07/11/22

DEEP LEARNING TECHNIQUES

UNIT 5 - ASSIGNMENT QUESTIONS

KRUTHI.M
RA2011047010044
B.Tech in AI-A

Unit 5 - Assignment Questions

1) Attention Mechanisms:

(i) What are attention models in Neural Networks? Explain the concepts:

(i) Attention over images.

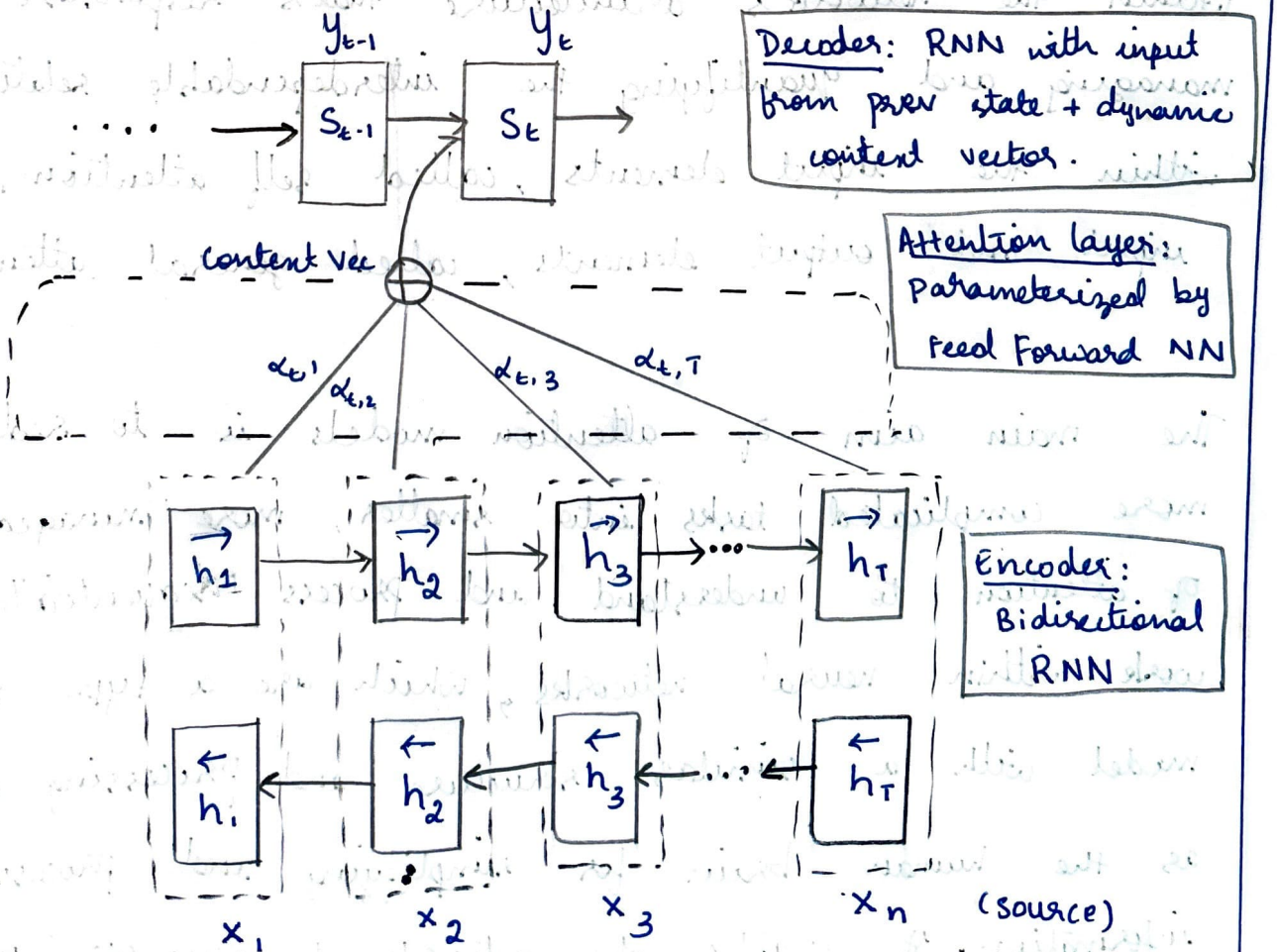
(ii) Hierarchical attention.

Soln:

Attention models, also called attention mechanisms, are deep learning techniques used to provide an additional focus on a specific component. In deep learning, attention relates to focus on something in particular and note its specific importance. The model typically focuses on one component within the network's architecture that's responsible for managing and quantifying the interdependable relationships within the input elements, called self attention, or between input and output elements, called general attention.

The main aim of attention models is to reduce larger, more complicated tasks into smaller, more manageable areas of attention to understand and process sequentially. The model work within neural networks, which are a type of network model with a similar structure and processing methods as the human brain for simplifying and processing information. It enables the network to focus on particular aspects.

An attention mechanism that can help a neural network memorize long sequences of information or data can be considered as the attention mechanism that broadly it is used in the case of Neural machine translation (NMT). The encoder compresses the sequential input as processes in the form of a content vector. We can introduce an attention mechanism to create a shortcut between the entire input and the content vector where the weights of the shortcut connections can be changeable for every output. Because of the connection between input and content vectors, the content vector can have access to the entire input, the problem of forgetting long sequences can be resolved to an extent.



Attention over image:

The encoder-decoder image captioning system would encode the image, using a pre-trained CNN. Then it would decode this hidden state by using a LSTM and generate caption.

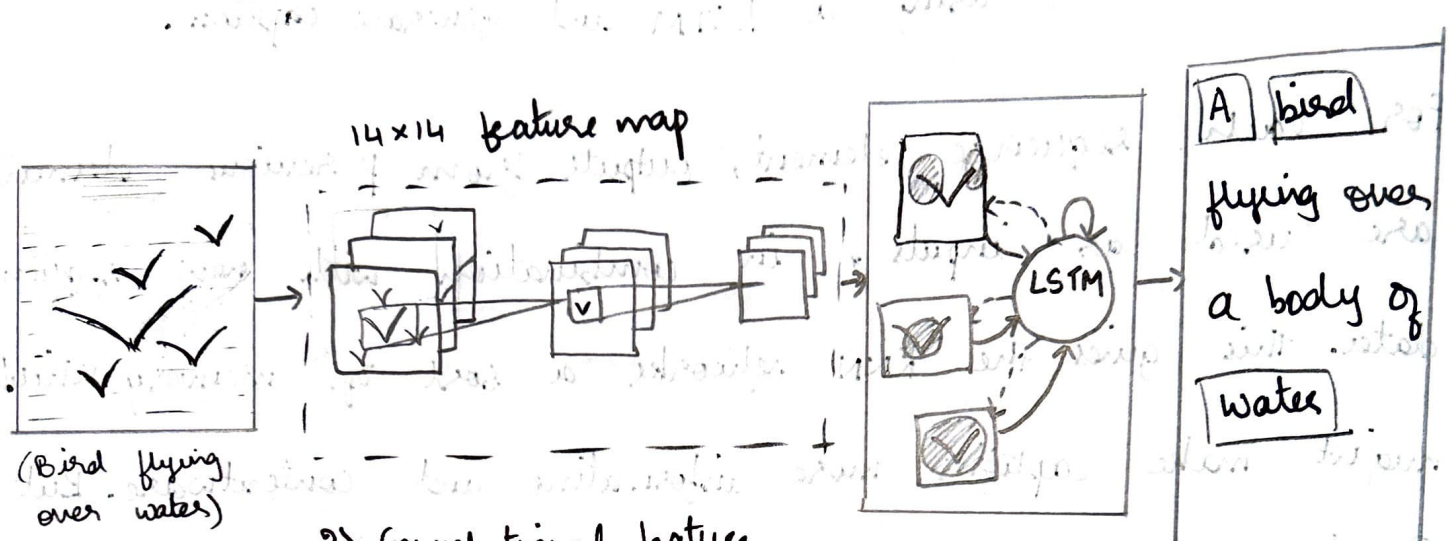
For each sequence element, outputs from previous elements are used as inputs, in combination with new sequence data. This gives the RNN networks a sort of memory which might make captions more informative and contextware. But RNN's tend to be computationally expensive to train and evaluate. Attention model can help address this problem by selecting the most relevant elements from an input image.

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

Local attention finds an alignment position and then calculates the attention weights in the left and right windows where its position is located and finally weights the context vector.

In the 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

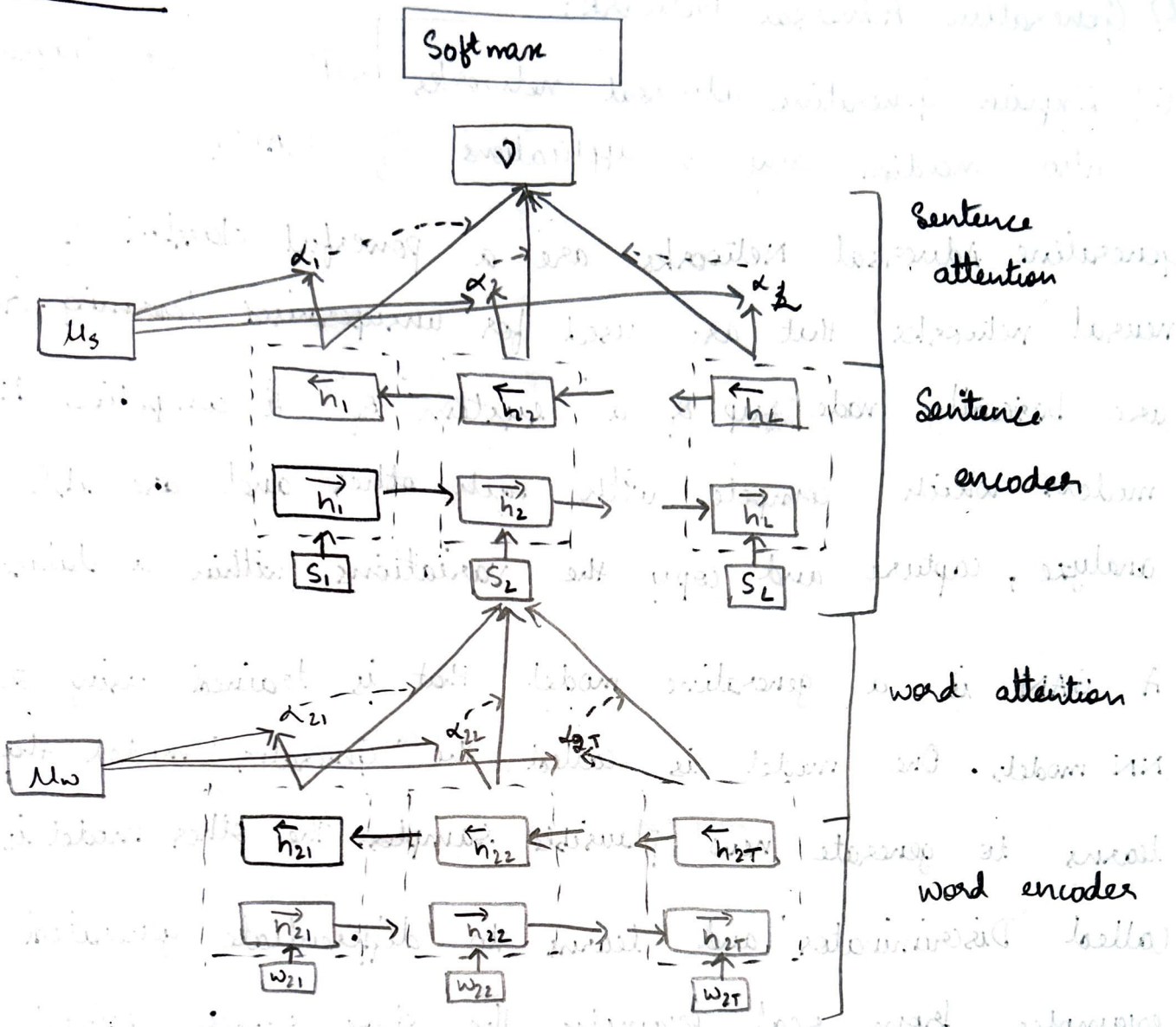
the current decoding according to a prediction function and then navigate through the content window, considering only the words within the window.



Hierarchical attention:

It uses stacked RNN on word level followed by attention model to extract such words that are important to the meaning of the sentence and aggregate the representation of those informative words to form a sentence vector. Then the same procedure applied to the derived sentence vectors which then generate a vector who conceives the meaning of the given document and that vector can be passed further to text classification.

Architecture :



First the network considers the hierarchical structure of docs by constructing a doc representation by building representations of sentences and then aggregating those into a doc 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 representation is built in the same way, however, it only requires the sentence vector of each sentence of the document as input.

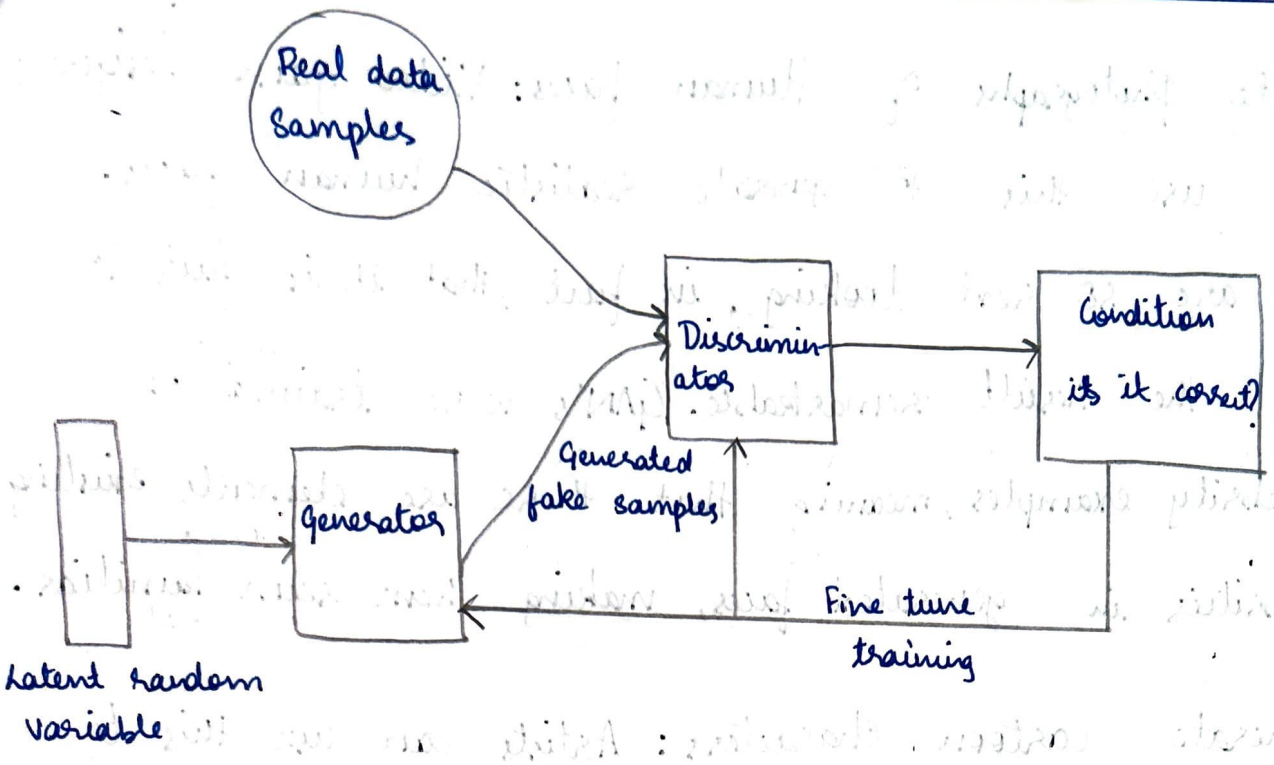
2) Generative Adversal Network:

(i) Explain Generative adversal networks with a neat diagram? Also mention any 3 applications of GAN?

Generative Adversal Networks are a powerful classful of neural networks that are used for unsupervised learning. GAN's are basically made up of a system of 2 competing NN models which compete with each other and are able to analyze, capture and copy the variations within a dataset.

A GAN is a generative model that is trained using 2 NN models. One model is called the 'Generator' model that learns to generate new plausible samples. The other model is called 'Discriminator' and learns to differentiate generated examples from real examples. The steps repeate several times and in this, the generator & discriminator get better and better in their respective jobs after each repetition.

Generative model captures the distribution of data and is trained in such a manner that it tries to maximize the probability ~~that the example~~ of the Discriminator in making a mistake. The Discriminator, on the other hand, is based on a model that estimates the probability that the sample that it got is received from training.



* When training begins, the generator produces obviously fake data, and the discriminator quickly learns to tell that its fake.
* As training progresses, the generator gets closer to producing output that can fool the discriminator.
* Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real and its accuracy decreases.
* The generator output is connected directly to the discriminator input.

Applications Of GAN's:

1) **Generate Examples for Image datasets**: Generating examples is very handy in medicine or material science, where there's very little data to work with. GAN's were used to generate new plausible examples for the MNIST dataset.

2) Generate photographs of Human faces: Video game designers can use this to generate realistic human faces. They are so real looking, in fact, that it is fair to call the result remarkable. GAN's were trained on celebrity examples, meaning that these are elements existing celebrities in generated faces, making them seem familiar.

3) Generate Cartoon characters: Artists can use this to create a new character design, or scenes in a cartoon, or even in a video game. GAN's are usually used to generate anime characters, such as pokemon characters, etc.

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

The GAN training algorithm involves training both the discriminator and generator model in parallel.

Algorithm: Minibatch SGD -training of GAN's sets. k=1.

for number of training iterations do

for k steps do

- Sample minibatch of m noise examples $\{z^{(1)}, \cdots, z^{(m)}\}$ from noise prior $P_g(z)$.

- Sample minibatch of m batch examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
- Update the discriminator by ascending its SGD.

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

end for

- Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Update the generator by descending its SGD.

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

end for

The gradient-based updates can use any standard gradient based learning rule. We used momentum turn in our exp.

Discriminator loss:

$$L^{(D)} = \max \left[\log(D(x)) + \log \left(1 - D(G(z))\right) \right]$$

Generator loss:

$$L^{(G)} = \min \left[\log(D(x)) + \log \left(1 - D(G(z))\right) \right]$$

In the first section in the algorithm, real data & fake data are inserted into the discriminator with correct labels and training takes place. Gradients are propagated keeping generator fixed.

Also, we update the discriminator by ascending its SGD because for discriminator we want to maximize the loss function.

On the other hand, we update the generator by keeping discriminator fixed and passing fake data with fake labels in order to fool the discriminator. Here, we update the generator by descending its SGD because because for the generator we want to minimize the loss function given.

3) Restricted Boltzman Machines:

(i) What do you mean by Restricted Boltzman machines?

The Boltzman Machine is a generative unsupervised model that relies on the learning of a probability distribution from a unique dataset and the use of the distribution to draw conclusions about unexplored data. It has one or more hidden layers in addition to the input layer, also known as the visible layer or the hidden layer.

In RBM's, the restricted term means that we are not permitted to connect two types of layers that are of the same type to one another.

Restricted Boltzman machines are stochastic two layered neural networks which belong to a category of energy based models that can detect inherent patterns automatically in the data by reconstructing input. They have two layers visible and hidden. Visible layer has input nodes and the hidden layer is formed by nodes which extract feature information from the data and the output at the hidden layer is a weighted sum of input layers. We only take care of input nodes and don't worry about hidden nodes.

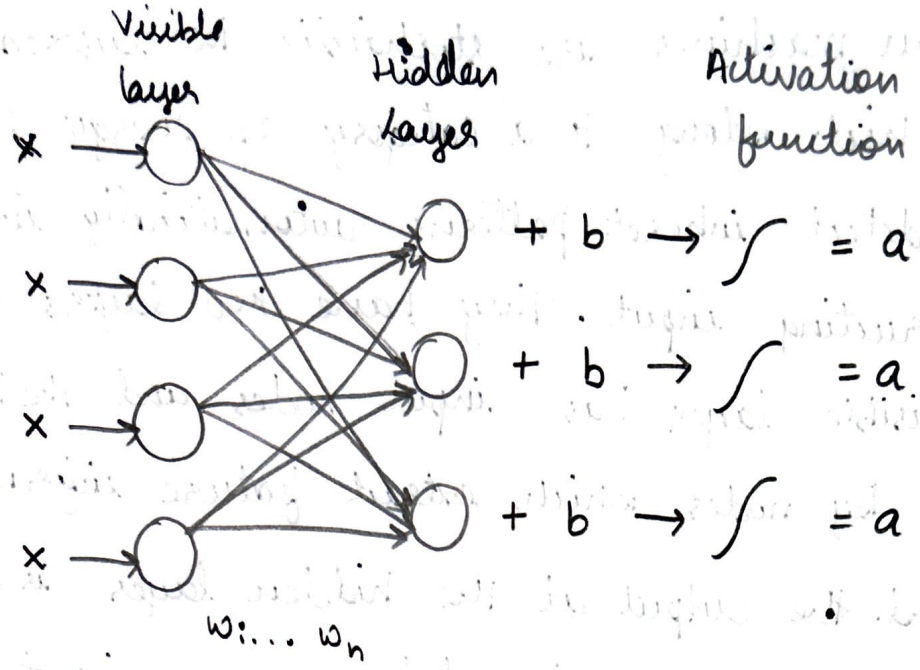
Once the input is provided RBM's automatically capture all the patterns, parameters and correlation among the data.

→ RBM's use recurrent and symmetric structure.

→ RBM's in their learning process try to associate high probability with low energy states & vice-versa.

(ii) How RBM's are used as Stochastic Neural Network ?

RBM is a stochastic neural network which means that each neuron will have some random behaviour when activated. There are 2 other layers of bias units in an RBM. The hidden bias RBM produces the activation on the forward pass and the visible bias helps RBM to reconstruct the input during a backward pass. The reconstructed input is always different from the actual input as there is no connections among the visible units and no transferring info.

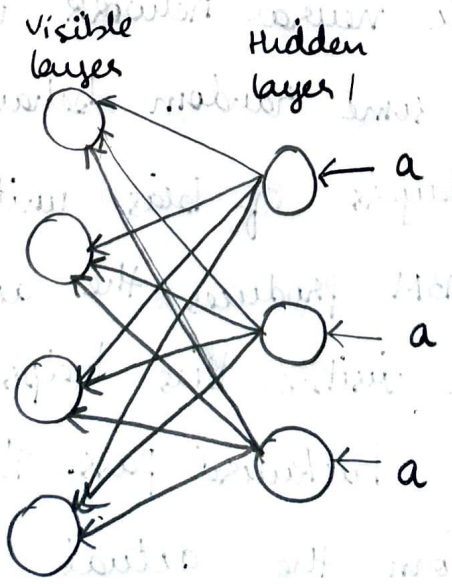


The above diagram represents first step in training an RBM with multiple inputs. The result is then passed through a sigmoid function and output determines if the hidden state gets activated or not. The first hidden node will receive the vector multiplication of the inputs multiplied by the first column of weights before the corresponding bias term is added to it.

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

h(1) & v(0)
↳ Vectors
a → hidden layer bias

Reconstruction:



r = b +

r = b +

r = b +

r = b +

reconstructions are the new input

Activations are the new input

weights are the same

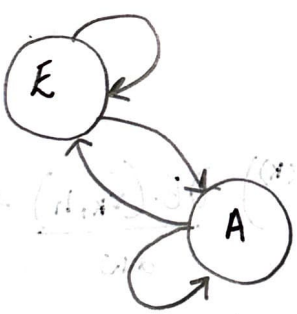
$$v^{(1)} = S(h^{(1)}W^T + b)$$

(iii) How to set up a Markov chain for RBM's?

We use the Markov chain Monte carlo algorithm for obtaining a sequence of observations which are approximated from a specified multivariate probability distribution i.e. the markov chain. The learning rule is

$$\Delta w_{ij} = \alpha \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon} \right)$$

Markov chain in simple words, the probability that $n+1^{th}$ steps will be x depends only on the n^{th} steps not to complete sequence of steps that came before n.



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

Gibbs sampling is a method where the values tend to converge towards distribution, also known to construct a markov chain. This is mainly used to sample multi-variable probability distribution. This method works iteratively for each variable in an instance. The Gibbs chain is initialized with a training sample $v^{(0)}$ of the training set and yields

the sample $v^{(k)}$ after k steps. Each step t consists of sampling $h^{(t)}$ from $p(h/v^{(t)})$ and sampling $v^{(t+1)}$ from $p(v/h^{(t)})$ subsequently (the value k=1 works quite well). The learning rule now becomes;

$$\Delta w_{ij} = \alpha(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}) \quad \cdots$$

The learning works well though it is only crudely approx the gradient of the log probability of the training data. The learning rule is much more closely approx the gradient of another objective function called the Contrastive Divergence which is the difference between two Kullback-Liebes divergences:

$$CD_k(w, v^{(0)}) = -\sum_h p(h/v^{(k)}) \frac{\partial E(v_k, h)}{\partial w} + \sum_h p(h/v_k) \frac{\partial E(v_k, h)}{\partial w}$$

Where the second term is obtained after each k steps of Gibbs sampling.