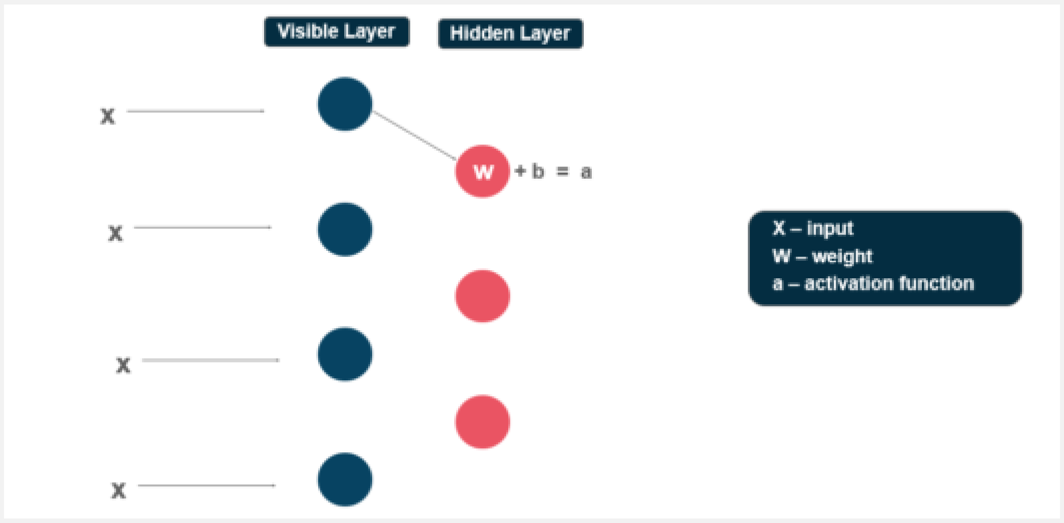
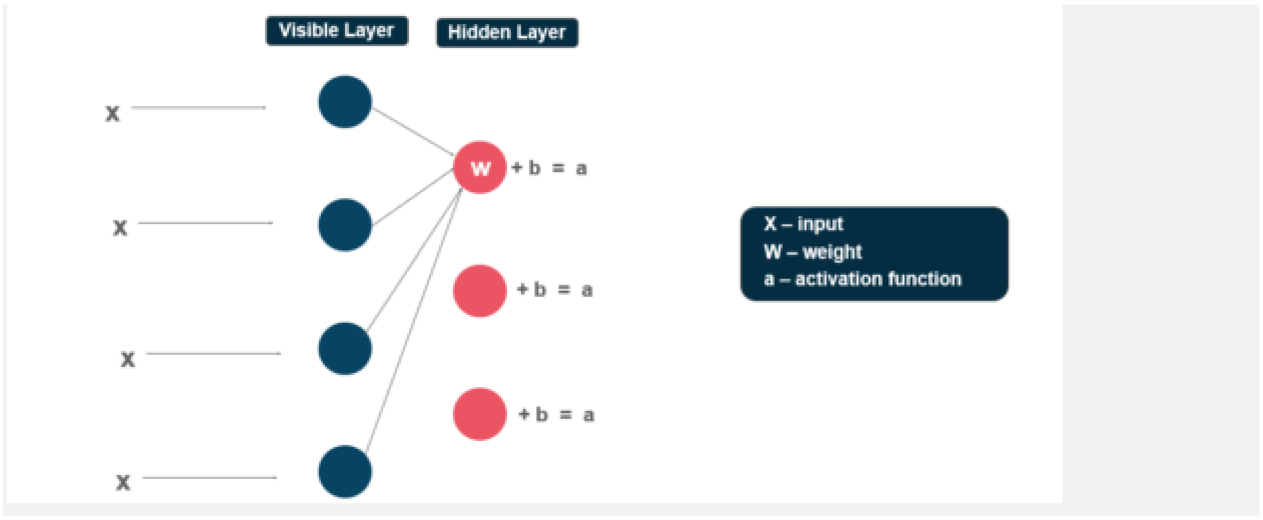
Working of Restricted Boltzmann Machine

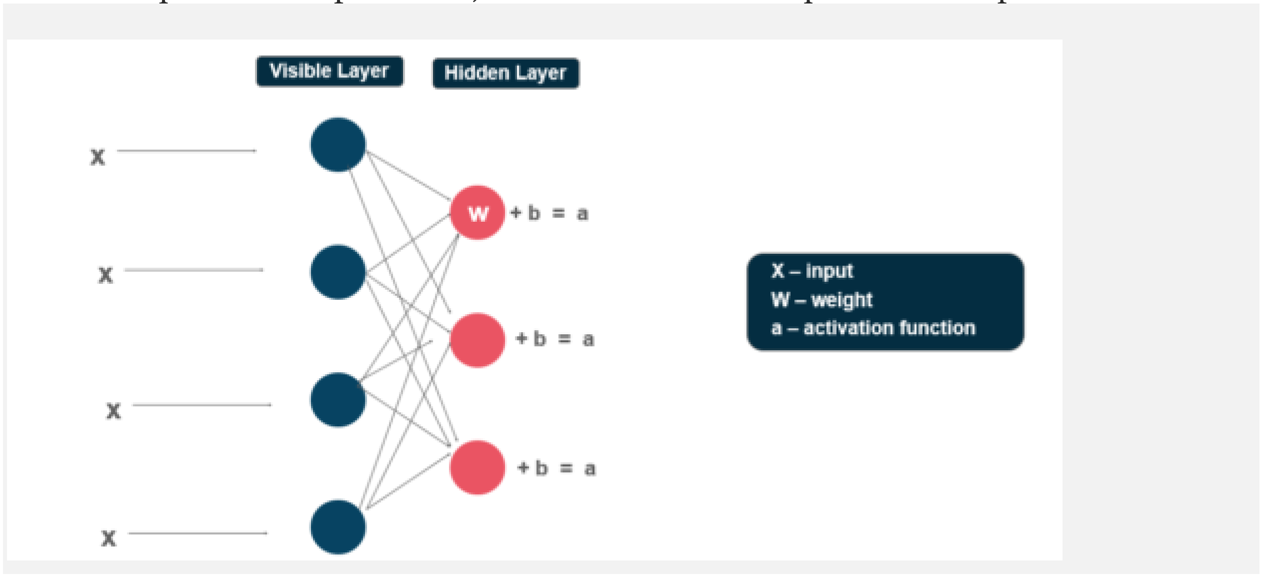
Each visible node takes a low-level feature from an item in the dataset to be learned. At node 1 of the hidden layer, **x is multiplied** by a ***weight*** and added to a ***bias***. The result of those two operations is fed into an ***activation function***, which produces the node's output, or the strength of the signal passing through it, given input x.



Next, let's look at how several inputs would combine at one hidden node. Each **x is multiplied** by a separate weight, the products are summed, added to a bias, and again the result is passed through an activation function to produce the node's output.



At each hidden node, each input **x is multiplied** by its respective weight w. That is, a single input x would have three weights here, making 12 weights altogether (4 input nodes x 3 hidden nodes). The weights between the two layers will always form a matrix where the rows are equal to the input nodes, and the columns are equal to the output nodes.



Each hidden node receives the four inputs multiplied by their respective weights. **The sum** of those products is again **added to a bias** (which forces at least some activations to happen), and the result is passed through the activation algorithm producing one output for each hidden node.

Now that you have an idea about how Restricted Boltzmann Machine works, let’s continue our Restricted Boltzmann Machine Tutorial and have a look at the steps involved in the training of RBM**.**

Training of Restricted Boltzmann Machine

The training of the Restricted Boltzmann Machine differs from the training of regular **neural networks** via stochastic gradient descent.

The Two main Training steps are:

Gibbs Sampling

The first part of the training is called *Gibbs Sampling*. Given an input vector **v** we use **p(h|v)**for prediction of the hidden values **h.** Knowing the hidden values we use **p(v|h)** :

$$p(v_i = 1 \mid \mathrm{h}) = \frac{1}{1+e(-(\mathrm{a}_i + w_i h_i))} = \tilde{\mathrm{O}}\ (\mathrm{a}_i + \sum \mathrm{h}_j \mathrm{w}_{ij})$$

for prediction of new input values **v**. This process is repeated *k* times. After *k* iterations, we obtain another input vector **v_k** which was recreated from original input values **v_o**.

$$p(h_i = 1 \mid \mathrm{v}) = \frac{1}{1+e(-(\mathrm{b}_j + W_j v_i))} = \sigma\,(\mathrm{b}_j + \sum_i v_i \mathrm{w}_{ij})$$

Contrastive Divergence step

The update of the weight matrix happens during the *Contrastive Divergence* step. Vectors **v_0** and **v_k** are used to calculate the activation probabilities for hidden values **h_0** and **h_k :**

$$p(v_i = 1 \mid \mathrm{h}) = \frac{1}{1+e(-(\mathrm{a}_i + w_i h_j))} = \tilde{\mathrm{O}}\ (\mathrm{a}_i + \sum \mathrm{h}_j \mathrm{w}_{ij})$$

The difference between the outer products of those probabilities with input vectors **v_0** and **v_k** results in the updated matrix :

$$\Delta W = \mathrm{v_O}) \otimes P(\mathrm{h_O} \mid \mathrm{v_O}) - \mathrm{v}_k \otimes P(\mathrm{h}_k \mid \mathrm{v}_k) - \mathrm{v}_k)$$

Using the update matrix the new weights can be calculated with gradient **ascent,** given by:

$$W_{new} = W_{old} + \triangle W$$