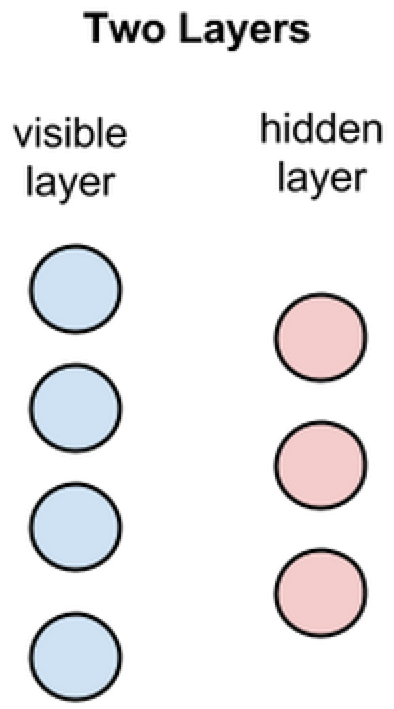
Restricted Boltzmann Machine

To reduce dimensionality, classify and predict the future, a Restricted Boltzmann machine was developed by Geoffrey Hinton. It may also be used for feature learning and topic modelling.

Deep belief networks are constructed from RBMs, which are two-layered shallow neural nets. Layer one of the RBM is referred to as input and layer two as the hidden layer. In the machine learning community, most practitioners prefer generative adversarial networks or variational autoencoders to RBMs, which are nevertheless utilised on occasion. For all their historical significance, RBMs are a little like the Model T of neural networks: intriguing to look at, but outperformed by more modern models in every way.



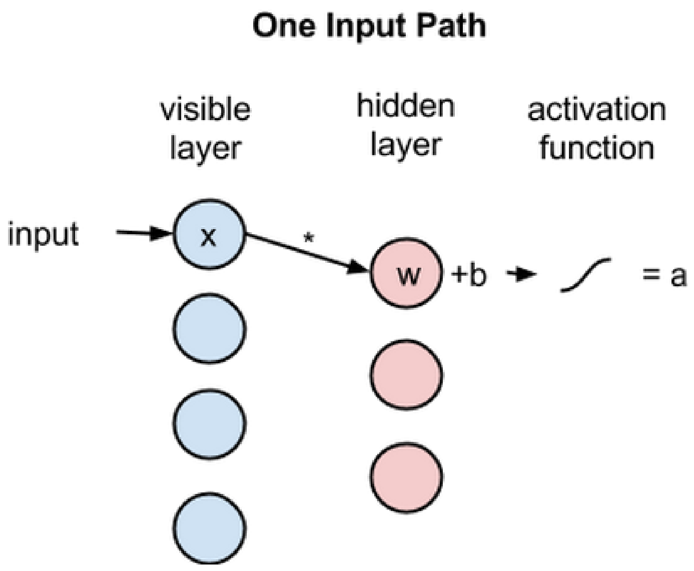
In the graph above, each circle is a node, which is a neuron-like unit where calculations take place. The nodes are interconnected across layers, however no two nodes in the same layer are linked together.

To put it another way, in a constrained Boltzmann machine, there is no intra-layer communication. To begin processing input, each node makes a stochastic decision about whether or not to transmit it.

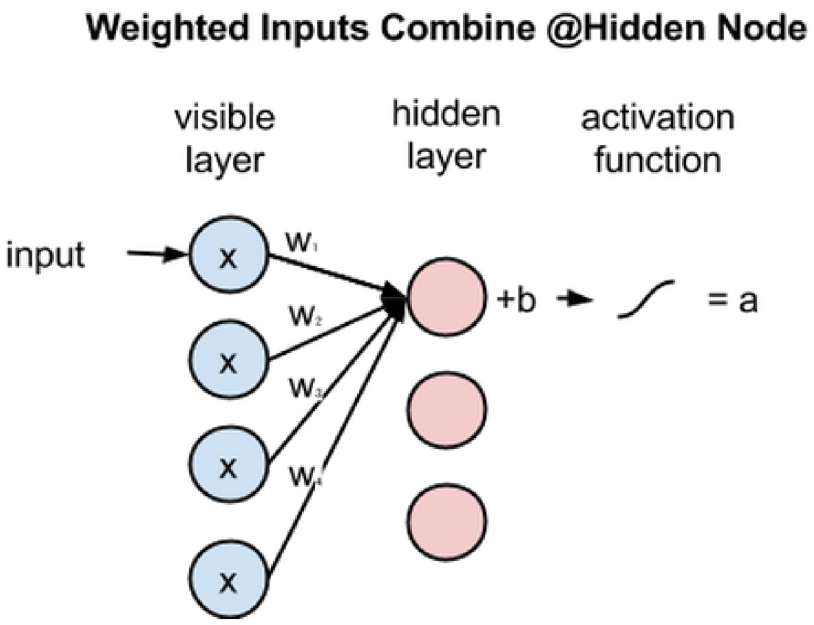
In the dataset, each visible node takes a low-level feature from an item. For example, each visible node would receive one pixel-value for each pixel in one image from a dataset of grayscale photos.

x will be our starting point, and we'll be tracking it across the two-layer network from here on out There is a weight applied to x at node 1 of the hidden layer, and that weight is then added to a bias. A node's output or signal strength is produced by an activation function fed with the results of these two actions, given input x.

activation f((weight w * input x) + bias b) = output a



Next, consider how a hidden node would aggregate numerous inputs. In order to get the node's output, each x is multiplied by its own weight and the resulting products are summed, added to a bias, and then run through an activation function again.



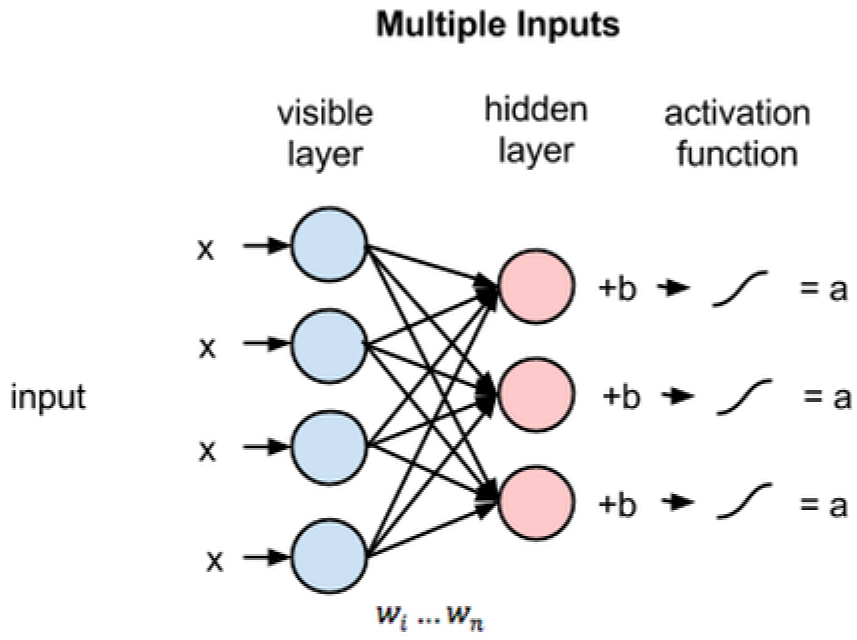
Next, consider how a hidden node would aggregate numerous inputs. In order to get the node's output, each x is multiplied by its own weight and the resulting products are summed, added to a bias, and then run through an activation function again.

An RBM can be described as a symmetrical bipartite network because inputs from all visible nodes are routed to all hidden nodes.

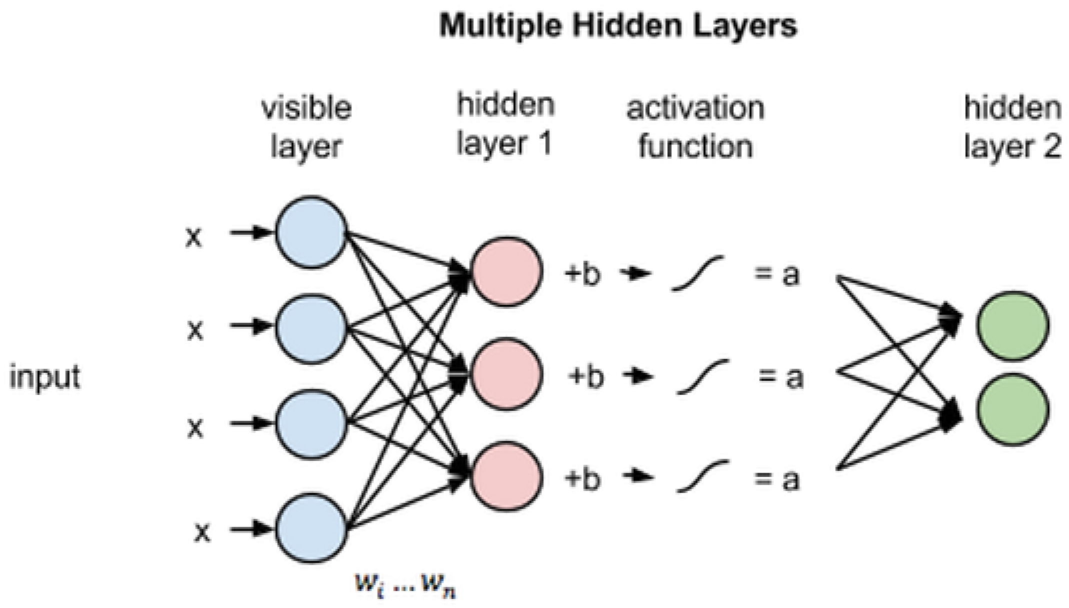
Symmetrical refers to the fact that each node, whether visible or not, is linked to every other node. In mathematics, the graph is referred to as a bipartite graph since it includes two levels or sections.

Each input x is multiplied by its corresponding weight w at each buried node. In other words, a single input x would be given three different weights, totaling a total of 12 different weights (4 input nodes x 3 hidden nodes). Weighing the input nodes and output nodes together will always result in a matrix with rows and columns equal to the input and output nodes.

The four inputs are multiplied by their respective weights and sent to each hidden node. For each hidden node, one output is generated by adding the total of its products to a bias.



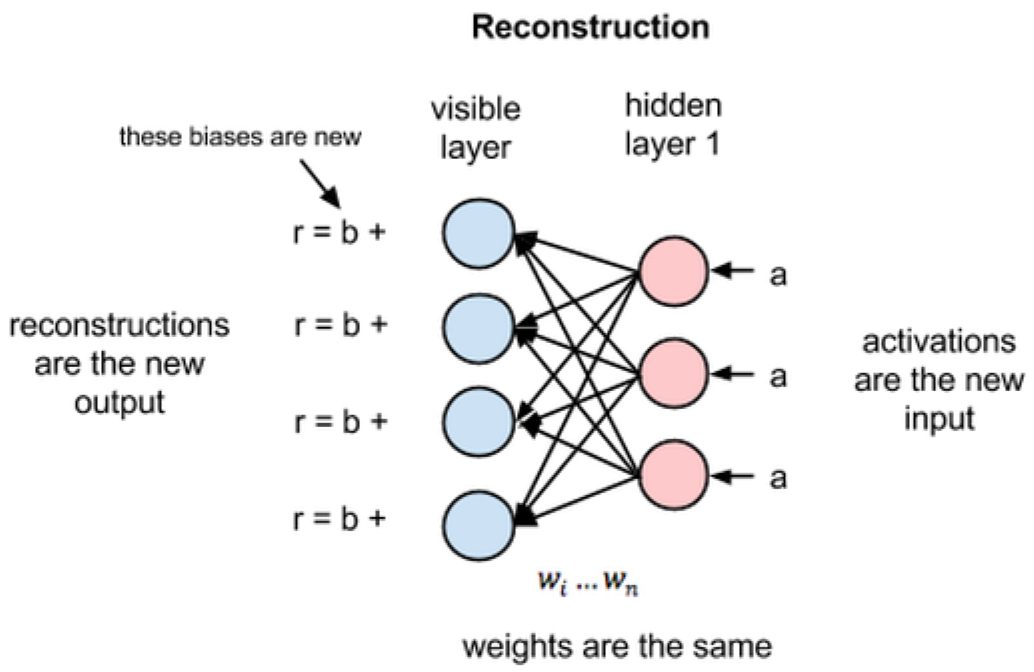
A deeper neural network would incorporate these two layers, with the outputs of hidden layer 1 being transmitted as inputs to hidden layer 2, and so on until they reach the final classification layer.



Reconstructions

To begin, we'll look at how restricted Boltzmann machines can learn to reconstruct data on their own in an unsupervised manner, making several forward and backward passes between the visible layer and hidden layer no. 1 without involving a deeper network. Unsupervised means there are no ground truth labels in the test set.

The activations of hidden layer no. 1 are used as input in a backward pass during reconstruction. As with the forward pass, they are multiplied by the same weights, one for each internode edge. All of these products are added to a visible-layer bias at each visible node and the result is a reconstruction, a close approximation to the original data that was entered. An illustration of this is shown in the diagram below:



The RBM's weights are randomly initialised, therefore the reconstructions often differ significantly from the original input. Errors in reconstruction are calculated by backpropagating against the RBM weights until a minimum amount of error has been achieved. This method is called iterative learning.

p(a|x; w) is the probability of output given a weighted x in an RBM's forward pass, as you can see from the diagram.

As a result of its backward pass, an RBM attempts to estimate the likelihood of inputs (x) given activations (a) that have the same weighting coefficients as those employed on its forward pass. This is known as reconstructing the original data. P(x|a; w) can be used to express the second phase.

Inputs x and activations a will lead to a joint probability distribution, which we'll refer to as the p (x, a).

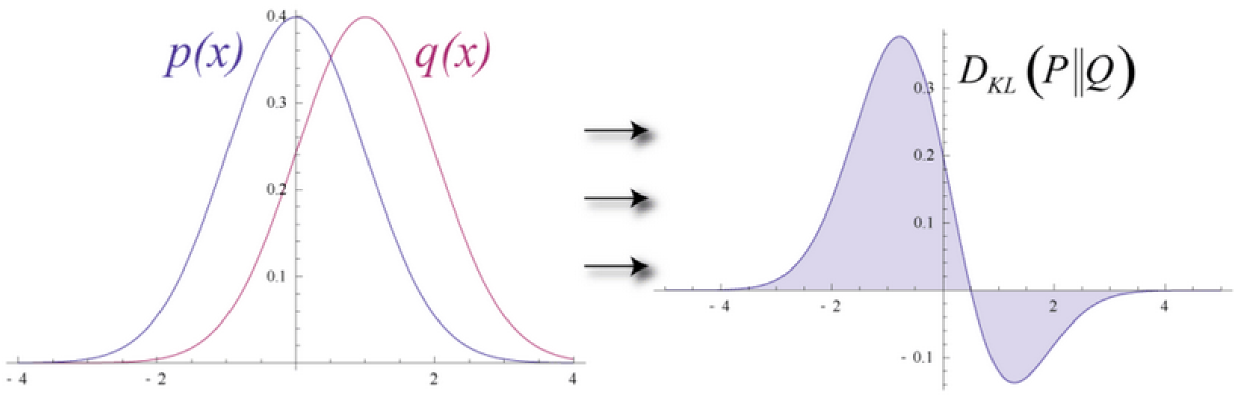
While regression estimates a continuous value based on numerous inputs, reconstruction makes educated predictions about which discrete label should be applied to a specific input example. Reconstruction also differs from categorization.

If you want to reconstruct something, you have to make educated predictions about the original input's probability distribution. To contrast this from the so-called discriminative learning, which maps inputs to labels and effectively draws lines between groups of data points, we use a process termed generative learning.

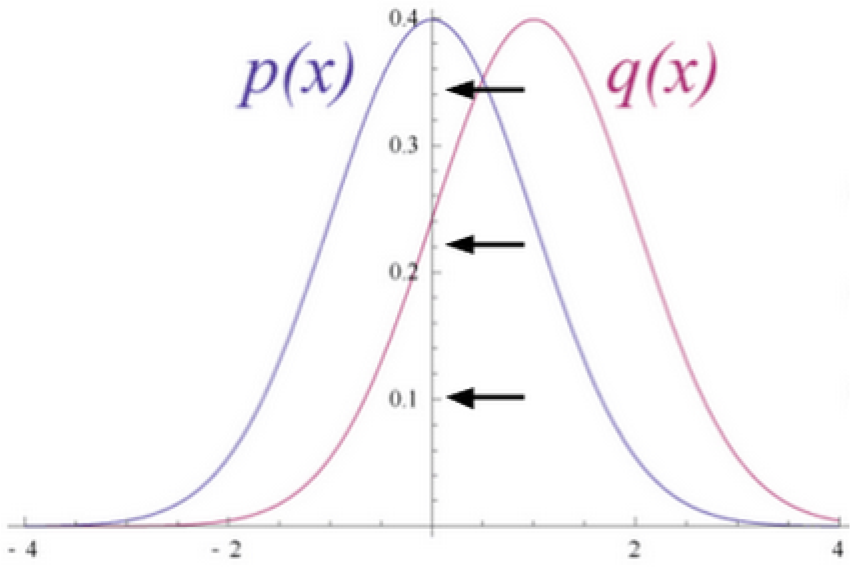
What if all you have is different-shaped normal curve input data, and all you have is partial-overlap reconstructions?

RBMs use Kullback-Leibler Divergence to gauge how far its estimated probability distribution differs from the input's true distribution.

If the hidden layer activations are multiplied by the shared weights, they should provide a close approximation to the original input, which is measured by KL-Divergence. An RBM's optimization algorithm aims to minimise the diverging areas beneath the two curves. Here we see the original input's p probability distribution next to the reconstructed distribution q, and the integration of their differences on the right.



An RBM learns to approximate the original data by iteratively modifying the weights based on the error they cause. The activations in the first hidden layer encode the structure of the input, and as a result, the weights begin to reflect that structure over time, as well. The process of learning resembles the gradual convergence of two probability distributions.



RBM output numbers can be interpreted as percentages. When the reconstruction shows a number other than 0, the RBM has learned the input.

Among all shallow feedforward networks, RBMs are not the most stable or consistent. A dense-layer autoencoder is preferable in many cases. In fact, techniques like variational autoencoders and GANs are becoming increasingly popular in the sector.