Restricted Boltzmann Machines

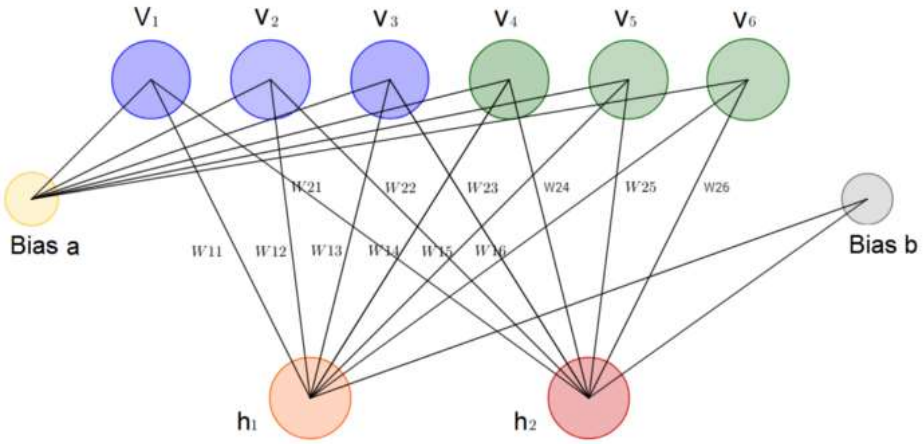
0. Introduction

Restricted Boltzmann Machines (RBMs) are neural networks that belong to so called *Energy Based Models*. This type of neural networks may be not that familiar to the reader of this article as e.g. feedforward or convolution neural networks. Yet this kind of neural networks gained big popularity in recent years in the context of the Netflix Prize where RBMs achieved state of the art performance in collaborative filtering and have beaten most of the competition.

1. Restricted Boltzmann Machines

1.1 Architecture

In my opinion RBMs have one of the easiest architectures of all neural networks. As it can be seen in Fig.1. a RBM consists out of one input/visible layer (v1,...,v6), one hidden layer (h1, h2) and corresponding biases vectors Bias *a* and Bias *b*. The absence of an output layer is apparent. But as it can be seen later an output layer wont be needed since the predictions are made differently as in regular feedforward neural networks.



1.2 An Energy-Based-Model

Energy is a term that may not be associated with deep learning in the first place. Rather is energy a quantitative property of physics. E.g. gravitational energy describes the potential energy a body with mass has in relation to another massive object due to gravity. Yet some deep learning architectures use the idea of energy as a metric for measurement of the models quality.

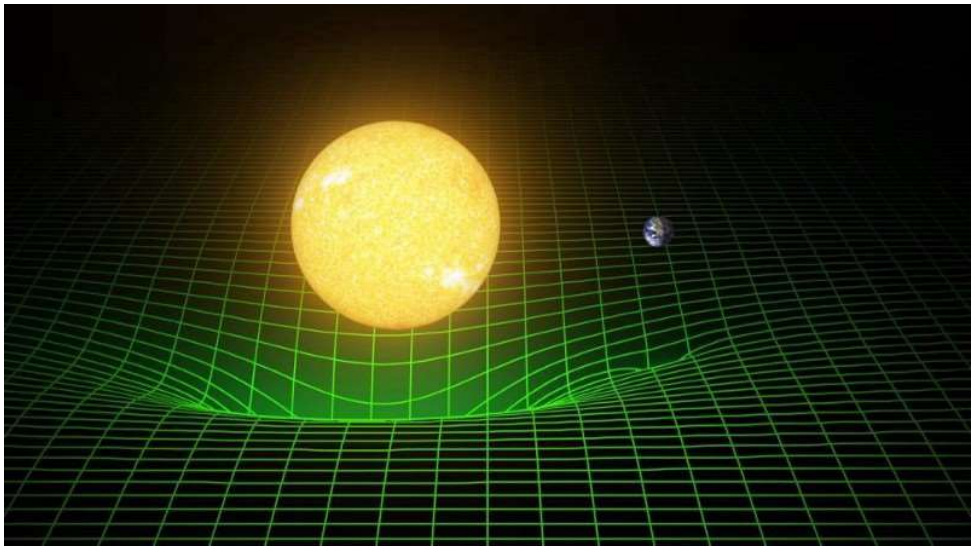


Fig. 2. Gravitational energy of two body masses.

One purpose of deep learning models is to encode dependencies between variables. The capturing of dependencies happen through associating of a scalar energy to each configuration of the variables, which serves as a measure of compatibility. A high energy means a bad compatibility. An energy based model model tries always to minimize a predefined energy function. The energy function for the RBMs is defined as:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

Eq. 1. Energy function of a Restricted Boltzmann Machine

As it can be noticed the value of the energy function depends on the configurations of visible/input states, hidden states, weights and biases. The training of RBM consists in finding of parameters for given input values so that the energy reaches a minimum.

1.3 A probabilistic Model

Restricted Boltzmann Machines are probabilistic. As opposed to assigning discrete values the model assigns probabilities. At each point in time the RBM is in a certain state. The state refers to the values of neurons in the visible and hidden layers **v** and **h**. The probability that a certain state of **v** and **h** can be observed is given by the following joint distribution:

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})}$$

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$$

Eq. 2. Joint Distribution for **v** and **h**.

Here **Z** is called the 'partition function' that is the summation over all possible pairs of visible and hidden vectors.

This is the point where Restricted Boltzmann Machines meets Physics for the second time. The joint distribution is known in Physics as the Boltzmann Distribution which gives the probability that a particle can be observed in the state with the energy *E*. As in Physics we assign a probability to observe a state of **v** and **h,** that depends on the overall energy of the model. Unfortunately it is very difficult to calculate the joint probability due to the huge number of possible combination of **v** and **h** in the partition function **Z**. Much easier is the calculation of the conditional probabilities of state **h** given the state **v** and conditional probabilities of state **v** given the state **h**:

$$p(\mathbf{h}|\mathbf{v}) = \prod_i p(h_i|\mathbf{v})$$

$$p(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i|\mathbf{h})$$

Eq. 3. Conditional probabilities for **h** and **v**.

It should be noticed beforehand (*before demonstrating this fact on practical example*) that each neuron in a RBM can only exist in

a binary state of 0 or 1. The most interesting factor is the probability that a hidden or visible layer neuron is in the state 1—hence activated. Given an input vector **v** the probability for a single hidden neuron *j* being activated is:

$$p(h_j = 1|\mathbf{v}) = \frac{1}{1 + e^{(-(b_j + W_j v_i))}} = \sigma(b_j + \sum_i v_i w_{ij})$$

Eq. 4. Conditional probability for one hidden neuron, given **v**.

Here is **σ** the Sigmoid function. This equation is derived by applying the Bayes Rule to Eq.3 and a lot of expanding which will be not covered here.

Analogous the probability that a binary state of a visible neuron *i* is set to 1 is:

$$p(v_i = 1|\mathbf{h}) = \frac{1}{1 + e^{(-(a_i + W_i h_j))}} = \sigma(a_i + \sum_j h_j w_{ij})$$

Eq. 5. Conditional probability for one visible neuron, given **h**.

2. Collaborative Filtering with Restricted Boltzmann Machines

2. 1 Recognizing Latent Factors in The Data

Lets assume some people were asked to rate a set of movies on a scale of 1–5 stars. In classical factor analysis each movie could be explained in terms of a set of latent factors. For example, movies like *Harry Potter* and *Fast and the Furious* might have strong associations with a latent factors of *fantasy* and *action*. On the other hand users who like *Toy Story* and *Wall-E* might have strong associations with latent *Pixar* factor. RBMs are used to analyse and find out these underlying factors. After some epochs of the training phase the neural network has seen all ratings in the training date set of each user multiply times. At this time the model should have learned the underlying hidden factors based on users preferences and corresponding collaborative movie tastes of all users.

The analysis of hidden factors is performed in a binary way. Instead of giving the model user ratings that are continues (e.g. 1–5 stars), the user simply tell if they liked (rating 1) a specific movie or not (rating 0). The binary rating values represent the inputs for the input/visible layer. Given the inputs the RMB then tries to discover latent factors in the data that can explain the movie choices. Each hidden neuron represents one of the latent factors. Given a large dataset consisting out of thousands of movies it is quite certain that a user watched and rated only a small amount of those. It is necessary to give yet unrated

movies also a value, e.g. -1.0 so that the network can identify the unrated movies during training time and ignore the weights associated with them.

Lets consider the following example where a user likes *Lord of the Rings* and *Harry Potter* but does not like *The Matrix*, *Fight Club* and *Titanic*. The Hobbit has not been seen yet so it gets a -1 rating. Given these inputs the Boltzmann Machine may identify three hidden factors *Drama*, *Fantasy* and *Science Fiction* which correspond to the movie genres.

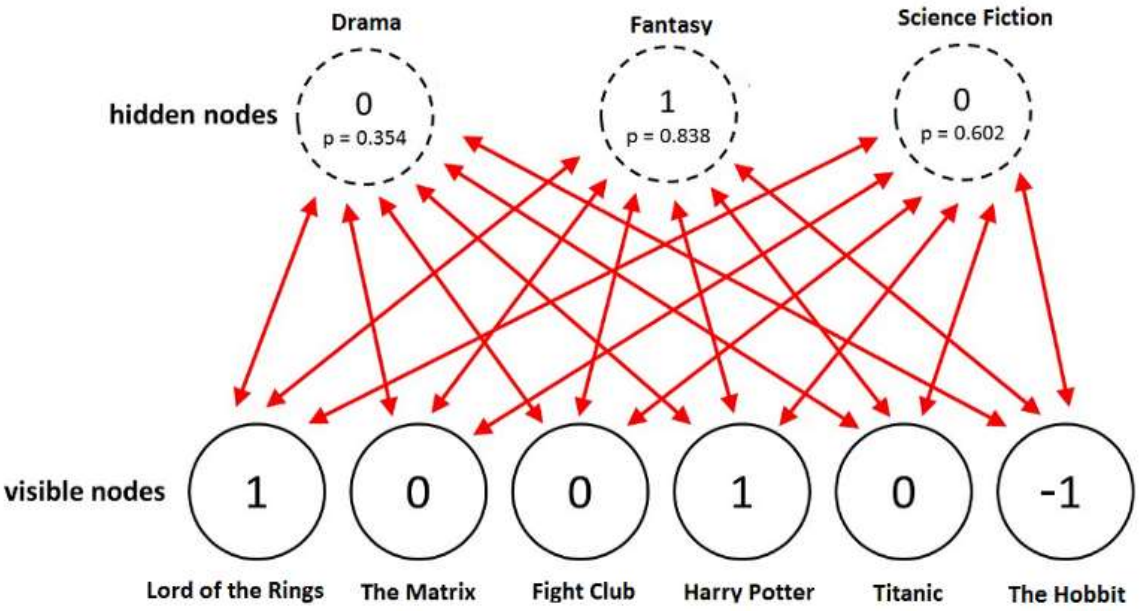


Fig. 3. Identification of latent factors.

Given the movies the RMB assigns a probability **p(h|v)** (Eq. 4) for each hidden neuron. The final binary values of the neurons are obtained by sampling from Bernoulli distribution using the probability **p**.

In this example only the hidden neuron that represents the genre *Fantasy* becomes activate. Given the movie ratings the Restricted Boltzmann Machine recognized correctly that the user likes *Fantasy* the most.

2.2 Using Latent Factors for Prediction

After the training phase the goal is to predict a binary rating for the movies that had not been seen yet. Given the training data of a specific user the network is able to identify the latent factors based on this users preference. Since the latent factors are represented by the hidden neurons we can use **p(v|h)** (Eq. 5) and sample from Bernoulli distribution to find out which of the visible neurons now become active.

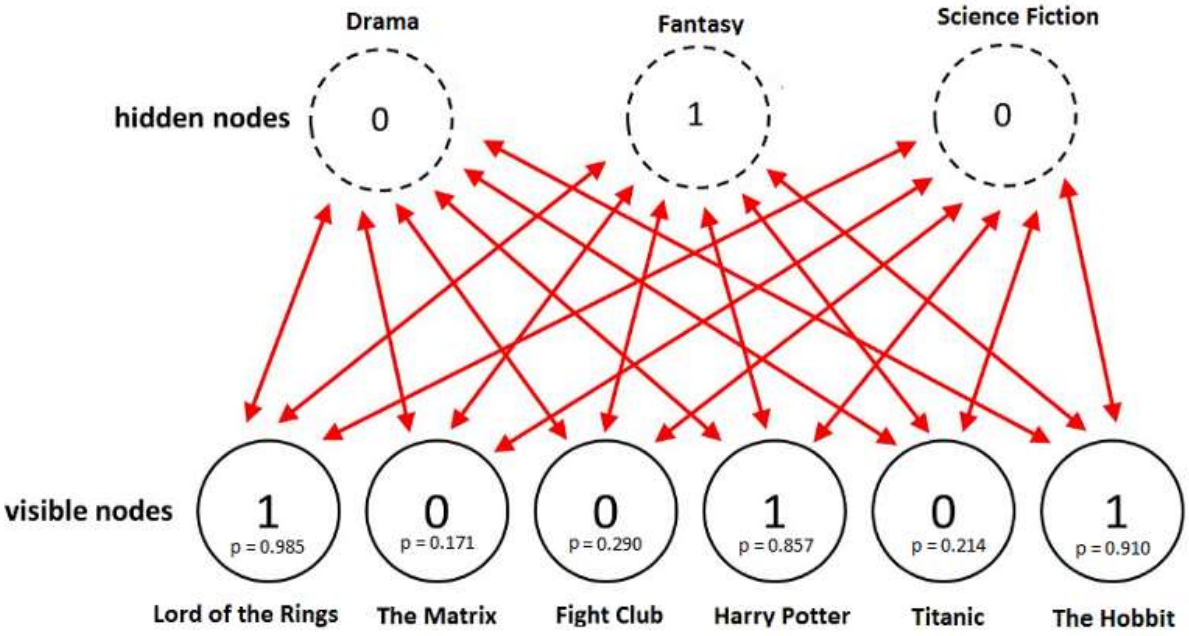


Fig. 4. Using hidden neurons for the inference.

Fig. 4 shows the new ratings after using the hidden neuron values for the inference. The network did identified *Fantasy* as the preferred movie genre and rated *The Hobbit* as a movie the user would like.

In summary the process from training to the prediction phase goes as follows:

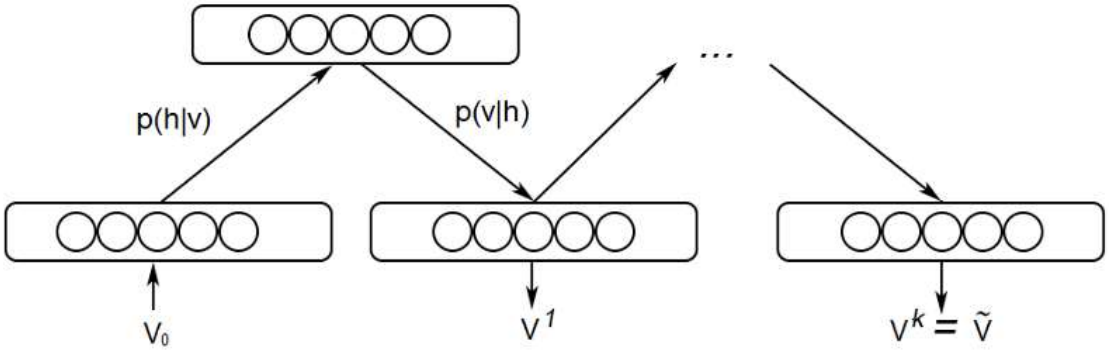
1. Train the network on the data of all users
2. During inference time take the training data of a specific user
3. Use this data to obtain the activations of hidden neurons
4. Use the hidden neuron values to get the activations of input neurons
5. The new values of input neurons show the rating the user would give yet unseen movies

3. Training

The training of the Restricted Boltzmann Machine differs from the training of a regular neural networks via stochastic gradient descent. The deviation of the training procedure for a RBM wont be covered here. Instead I will give an short overview of the two main training steps and refer the reader of this article to check out the original paper on Restricted Boltzmann Machines.

3.1 Gibbs Sampling

The first part of the training is called *Gibbs Sampling*. Given an input vector **v**we are using **p(h|v)** (Eq.4) for prediction of the hidden values **h.** Knowing the hidden values we use **p(v|h)** (Eq.5) for prediction of new input values **v**. This process is repeated *k* times. After *k* iterations we obtain an other input vector **v_k** which was recreated from original input values **v_o**.



3.2 Contrastive Divergence

The update of the weight matrix happens during the *Contrastive Divergence*step. Vectors **v_o** and **v_k** are used to calculate the activation probabilities for hidden values **h_o** and **h_k** (Eq.4). The difference between the outer productsof those probabilities with input vectors **v_o** and **v_k** results in the update matrix:

$$\triangle W = \mathbf{v}_0 \otimes p(\mathbf{h}_0|\mathbf{v}_0) - \mathbf{v}_k \otimes p(\mathbf{h}_k|\mathbf{v}_k)$$

Eq. 6. Update matrix.

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

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

Eq. 7. Update rule for the weights.