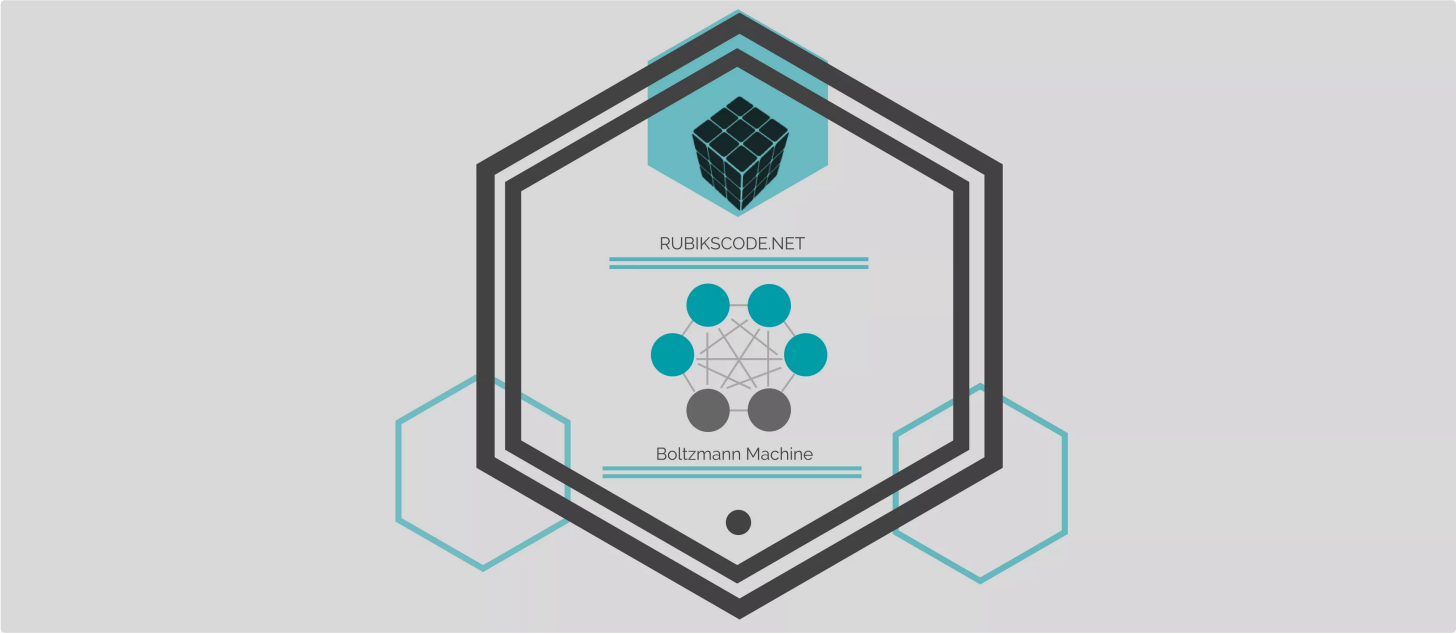
RUBIK'S CODE

Freedom.
Wisdom.
Excellence.

Introduction to Restricted Boltzmann Machines

OCTOBER 1, 2018 — 2 COMMENTS



So far in our artificial neural network journey, we have explored typical statistical models. In general, the entire point of statistical modeling and machine learning is to detect dependencies and connections between input variables. Standard Neural networks are exceptional at this job. Not only are they capable of detecting these dependencies, but also based on these dependencies neural networks have the ability to answer questions about the values of unknown variables given the values of known variables.

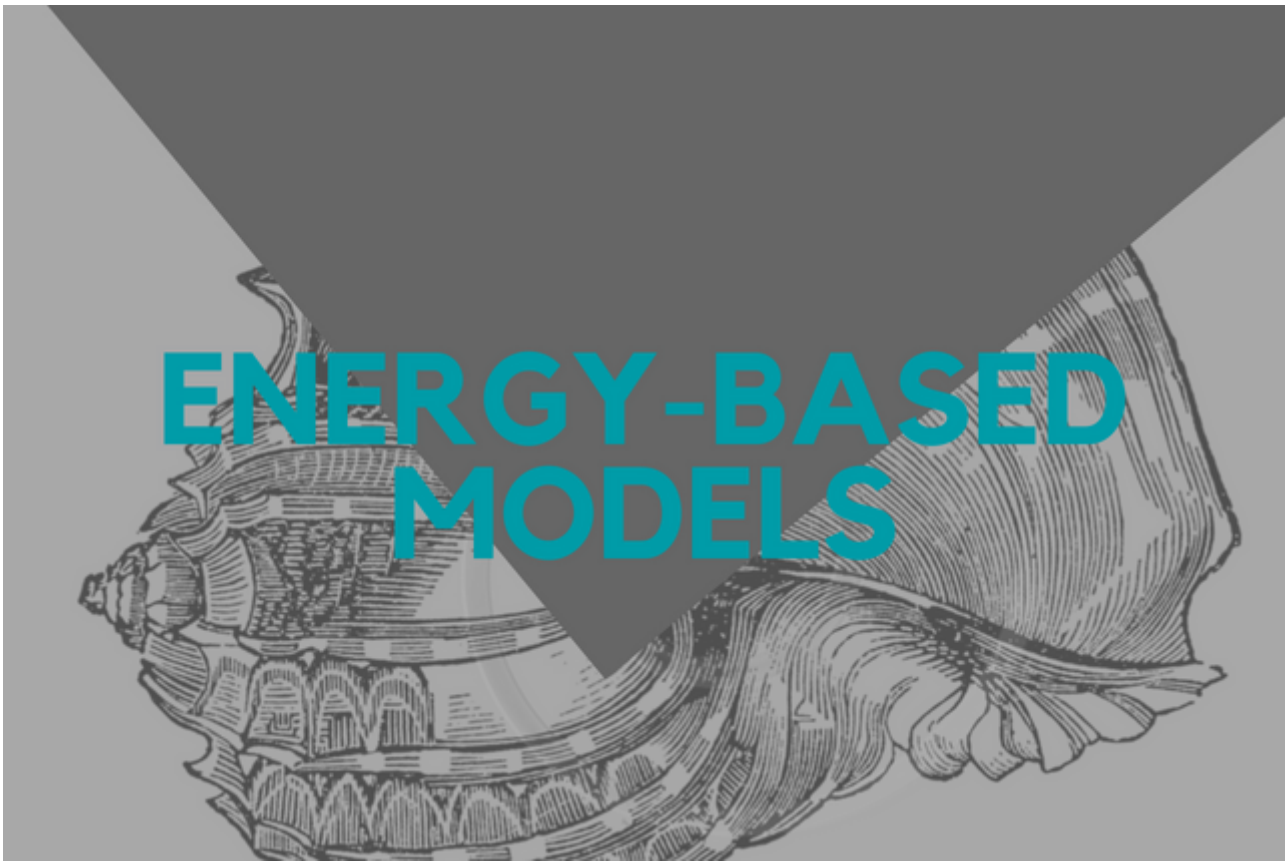
However, these systems are also closed-blacked-boxed. After you have trained your neural network, you can pass information into it and get a result. There is little to no information as to why a neural network made a certain decision and the reasoning behind it. That is why

architectures of neural networks that we are going to examine in this article, (Boltzmann machine), have a different approach. They also provide a user with means with which they can verify the validity of the networks' decisions. In a way, they are more transparent.

If you have worked with standard, feed-forward neural networks, you have probably been in a situation to face their biggest issue – the vanishing gradient. This is a problem that occurs during training of the network, when the gradient becomes vanishingly small. A small gradient prevents the weights from changing their value. The vanishing gradient problem spun a branch of researches that tried to come up with a new model that wouldn't have this problem. Eventually, we got the Deep Belief Networks, which have Restricted Boltzmann Machine as their main building unit.

To understand the Restricted Boltzmann Machine, we need to understand the standard Boltzmann Machine first. They are Energy-Based Models, which sounds like a weird mashup of physics and deep learning, and in fact, it is. This physics-based concept was first introduced by John J. Hopfield in 1982, but was further developed by Geoffrey E. Hinton and Terrence J. Sejnowski in their papers "*Analyzing Cooperative Computation*" and *"Optimal Perceptual Inference".* They gained popularity through the Netflix Prize competition. So, our attack plan would be to first check out the idea behind Energy-Based Models, and then learn a bit about the Boltzmann Machine, which would finally lead us to the exploration of the Restricted Boltzmann Machines.

Energy-Based Models



The first time I heard of this concept I was very confused. “Energy is a term from physics”, my mind protested, “what does it have to do with deep learning and neural networks?”. Well, in physics, energy represents the capacity to do some sort of work. It also comes in many forms, meaning that energy can be potential, kinetic, thermal, electrical, chemical, nuclear and so on. There is a set of deep learning models called Energy-Based Models (EBMs), which utilize this concept. They determine dependencies between variables by associating a scalar value to the complete system – *Energy*. This scalar value actually represents a measure of the probability that the system will be in a certain state. In the Boltzmann Machine this is probability defined by the Boltzmann Distribution:

$$p_i = \frac{e^{-\varepsilon_i/kT}}{\sum_{j=1}^{M} e^{-\varepsilon_j/kT}}$$

From this example, you can conclude that probability and Energy are inversely proportional, i.e. if the Energy is higher, the probability is lower and vice-versa. Energy-based models have two important processes:

- Inference

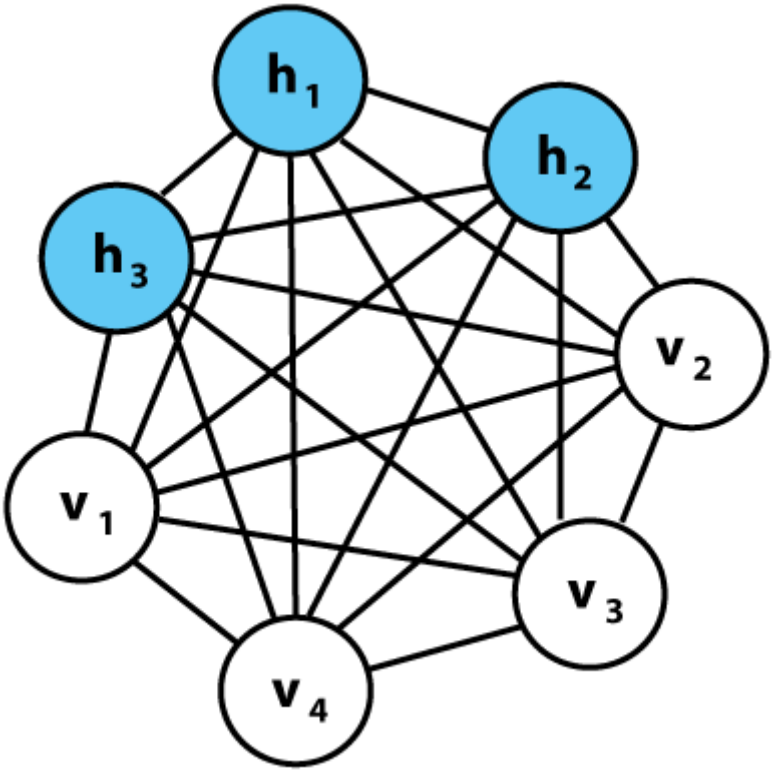
- Learning

Inference represents the process of making a prediction or a decision. In essence, it is done in two steps. First, the values of the observed variables are set to 1. The values for other variables are found so that Energy of the complete system is minimized. The goal is to increase the probability of the system by reducing the energy of the system. On the other hand, the learning process consists of finding an energy function that will associate low energies to correct values of the remaining variables and higher energies to incorrect values.

In these models, another value plays a big role. It is called loss functional and it is used to measure the quality of available energy function. This value is minimized during the learning process. Because we can have a wide range of energy functions and loss functionals at our disposal, the energy-based concept can create many statistical models. In a sense, this concept is a framework which, by using, we can create both probabilistic or non-probabilistic models. Another benefit of this approach is that there is no normalization, which gives these models the ability to avoid problems associated with the normalization constant.

Boltzmann Machine

The Boltzmann Machine is one type of Energy-Based Models. They consist of symmetrically connected neurons. These neurons have a binary state, i.e they can be either on or off. The decision regarding the state is made stochastically. Boltzmann machines utilize simple learning mechanism, using which are able to discover interesting features among values in a set of binary vectors. Usually, these networks are used to solve search problems and decision problems. In a sense, they are modeled in a way that they can learn about the system using input data and then we can use them as a sort of simulator of that system. Here is one example of Boltzmann machine architecture:



As you can see, there are seven neurons and they are all connected with one another. Of course, every connection is weighted. Apart from that, some neurons are visible and some are hidden. For example, in the image above we have three hidden neurons *(h1, h2, h3*) as well as four visible ones (*v1, v2, v3, v4).* This means that the user will be able to set input to the visible neurons, but not to hidden ones. The inputs' value doesn't need to have values for every neuron; the Boltzmann machine will generate it for us. This is another big difference from standard neural networks in a sense that inputs are also the outputs, but we will talk more about this during the learning procedure. Now let's explain the stochastic nature of this system a bit.

When neuron *i* is given the opportunity to change its binary state, it first calculates the total input on connections of all active neurons and adds its own bias to it,

$$z_i = b_i + \sum_j s_j w_{ij}$$

where *bi* represents the aforementioned mentioned bias, *sj* the current state of the neuron that is connected, and *wij* the weight on the connection between neurons *i* and *j.* This means that there is the probability that neuron *i* will be active, meaning it will have a state one is given by the formula:

$$prob(s_i = 1) = \frac{1}{1 + e^{-z_i}}$$

If neurons update their state in any given order, this means that the network will eventually reach a Boltzmann distribution, which states that probability of a state vector *v* is determined by the "energy" of that state vector relative to energies of all possible binary state vectors:

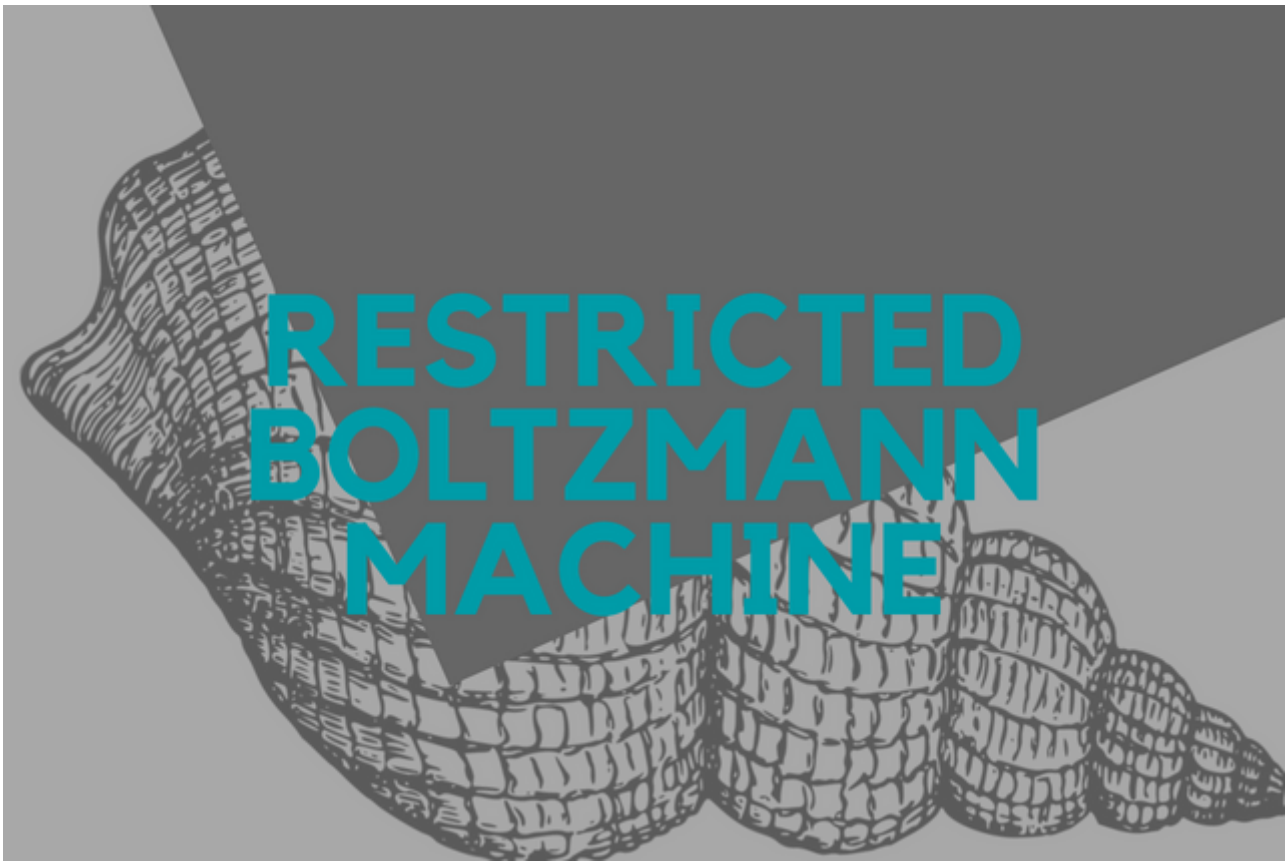
$$P(\mathbf{v}) = e^{-E(\mathbf{v})} / \sum_{\mathbf{u}} e^{-E(\mathbf{u})}$$

The energy of a state vector in the Boltzmann machine is defined as

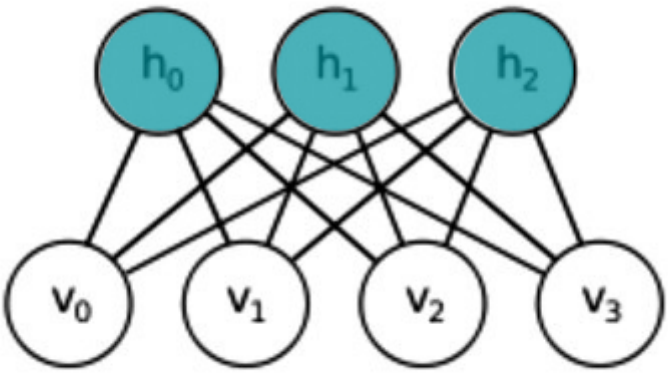
$$E(\mathbf{v}) = -\sum_i s_i^{\mathbf{v}} b_i - \sum_{i<j} s_i^{\mathbf{v}} s_j^{\mathbf{v}} w_{ij}$$

where *si* is the state assigned to neuron *i* by state vector *v*. From this equation we can see the dependency between the energy of the system and the weighted connections. In a nutshell, the goal of the learning process is to find a set of weights that will minimize the energy. However, the Boltzmann machine's architecture is resource-demanding. Since every neuron is connected to every other neuron, calculations can take a long time. That is why Restricted Boltzmann Machines (RBM) came into the picture.

Restricted Boltzmann Machines



The only difference in the architecture between RBMs and the standard Boltzmann machines is that visible and hidden neurons are not connected among each other, i.e. visible neurons are only connected to hidden neurons. That looks something like this:



This way the number of connections is reduced and a learning process of this kind of networks is demanding. As you can see, these kinds of networks have a very simple architecture, which is the main building block of deep belief neural networks. Energy function for RBM is defined by this formula,

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

where *ai* is the bias of the visible neuron *i* that is in state *vi, bj* is a bias of the hidden vector *j* that is in state *hj* and *wij* is the weight of the connection between neuron *i* and *j.* This function is just a specialization of the previous formula.

The probability of the whole system can be presented using the states of neurons in the hidden layer – *h* as well as the states of the neurons in the visible layer – *v*

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

where *Z summs all* possible pairs of visible and hidden vectors and it is called the *partition function*. This again, is just a specialization of the Boltzmann distribution to the RBM's architecture. This formula provides us with an opportunity to calculate the probability that any neuron is activated. Given an input vector *v,* the probability for a single hidden neuron *j* being activated is

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

where σ is the Sigmoid function. Analogous to this the probability that visible neuron *i* is activated can be calculated like this:

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

Learning Process

As you can imagine, the learning process of RBM greatly differs from the one that is in place with the feed-forward neural networks. It has two major processes:

- Gibbs sampling
- Contrastive Divergence

Gibbs sampling is a sub-process that itself consists of two parts. First, the input values are set to the visible layer, and then based on that, states of the neurons in the hidden layer are calculated. That is done using probability functions *p(h|v)* from the previous chapter. Once these values are available, the other function *p(v|h)* is used to predict new input values for the visible layer. This process can be repeated *k* times, but in practice, it is repeated once or twice for every input sample. In the end, we get the other input vector *vk* which was recreated from original input values *v0.*

Contrastive Divergence is a sub-process during which the weights are updated. Vectors *v0* and *vk* are used to calculate activation probabilities for the hidden layers *h0* and *hk,* again using the formula for *p(h|v)*. Finally, we can get the matrix which will define delta for which weight needs to be updated:

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

Conclusion

The Restricted Boltzmann machines are one alternative concept to standard networks that open a door to another interesting chapter in deep learning – the deep belief networks. Their simple yet powerful concept has already proved to be a great tool. It is used in many recommendation systems, Netflix movie recommendations being just one example. In the articles to follow, we are going to implement these types of networks and use them in a real-world problem.

Thank you for reading!

This article is a part of Artificial Neural Networks Series, which you can check out **here**.

Read more posts from the author at **Rubik's Code**.
