Chapter 14 : Part 3

**Deep Learning**

**BM: Contrastive Divergence &**

**DBN, DBM**

Contrastive Divergence & Advanced topics

**14.3.1 Contrastive Divergence:** Diagrammatically

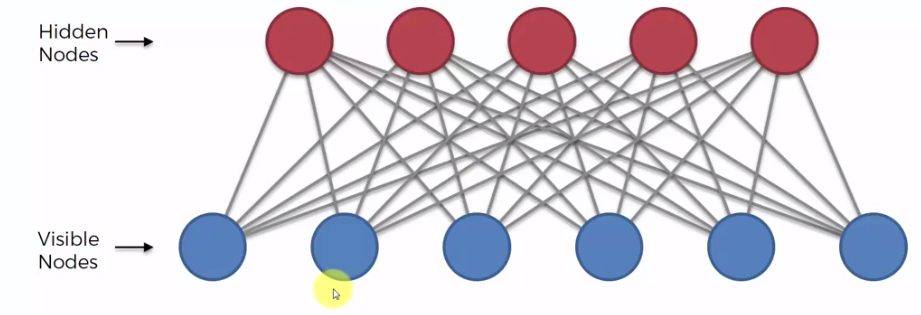
In this part we'll discuss about Contrastive Divergence. This is the algorithm that actually allows RBM to learn.

* Here we've got a diagrammatic representation of our RBM. We've got 2 input Nodes (Blue), and we've got 3 hidden nodes in Red.

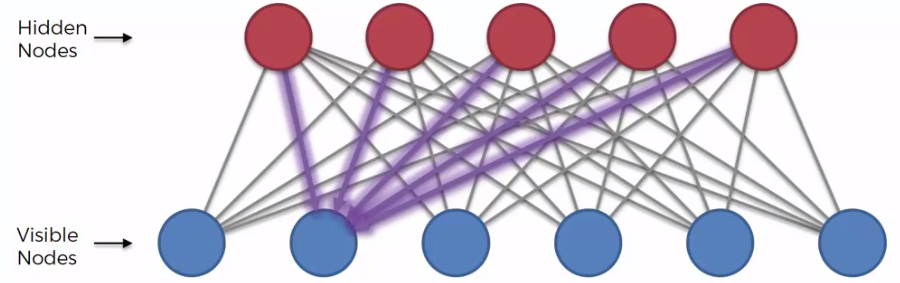
Here we focus on a specific part of the learning process. In previous section of this chapter

|  |  |  |
| --- | --- | --- |
| * We discussed exactly how we feed in different *values* in *RBM*, and how it looks at them and looks for *features* and then *assign* certain *nodes* to those *features*. * But we didn't discuss how RBM adjust those weights to connect the nodes. * Previously in the other NN, we had the *gradient descent* process, which allowed us to *back* *propagate* the *error* through the *NN*, and *adjust* the *weights* to *minimize* that *error*. * Previously those *NNs* are *directed* network. But in *RMB*, we don't have an *undirected* network. * This is where the Contrastive Divergence comes in. It is used to adjust weights in *undirected* network. | |  |
| * We're going to look at it in two ways. * *Diagrammatically* like this and * Later through an *Energy graph*. | * Here we've got 2 input nodes. At the very start the RBM calculates the *hidden* *nodes* using some *randomly* *assigned* *weights*. Then those hidden nodes are going to use the exact same weights to reconstruct the input nodes. * It important to understand that the ***reconstructed*** ***inputs*** are not going to equal to the ***original*** ***inputs***, even though the ***weights*** are the ***same***. | |

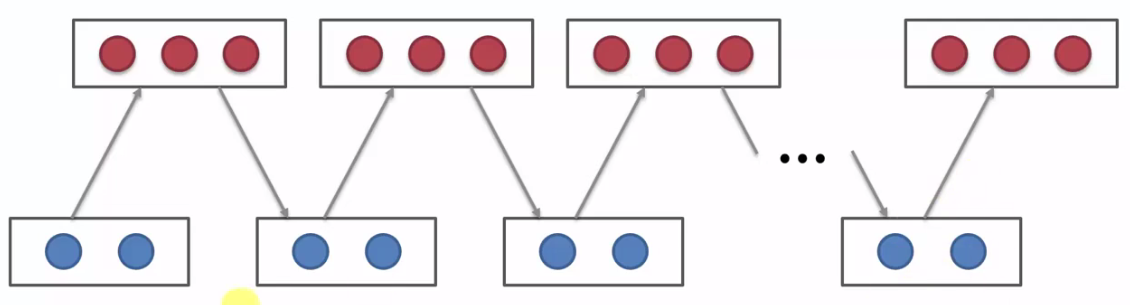
* Once we've *reconstructed the visible-nodes*, they're not identical to the original visible nodes. Even though we're using the same weights.
* The reason for that is because these nodes are not initially interconnected. There's no specific connection, and there's no formula or equation that's connecting them at very beginning.



* Lets say second input-node (visible-node) get reconstructed. It gets reconstructed using *all* the *hidden nodes* (all five hidden nodes). But those hidden nodes first created from our original input-nodes (all six *input-nodes/visible-node* contributed to create each *hidden* *node*).
* At the very-beginning in the RBM, these initial input-values will initiate some values in your hidden nodes. Then once we run it backwards, these hidden nodes will reconstruct, all of input-nodes including this second input-node.



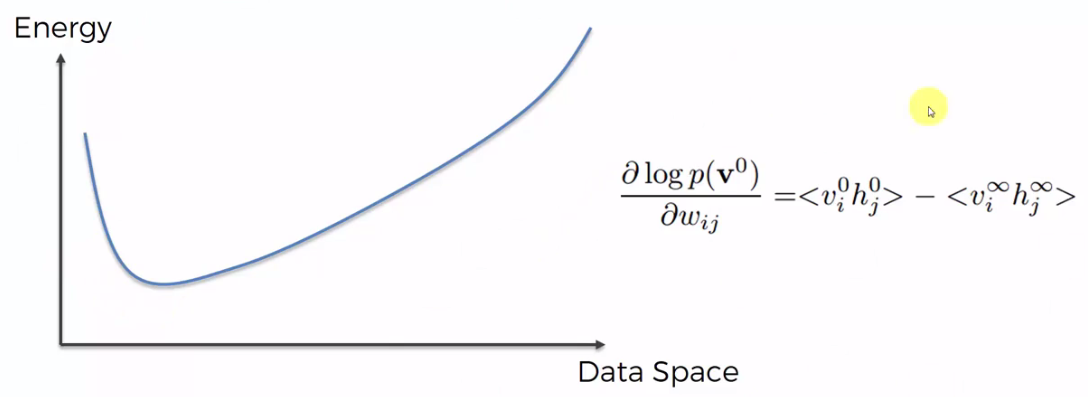
* If *only one input-node* is used to create *all* *hidden-node* then the *reconstructed input-node* would be same to input-node.
* Since all *initial-input-nods* contributed to create *each* of *hidden nodes*, and all hidden nodes are trying to reconstruct each of input-nodes. That’s why reconstructed input-nodes won't be same as initial-input-nods *(because indirectly all initial-input-nods are using to reconstruct an input-node)*.
* That's very important to understand, that's why this whole Contrastive Divergence process exists.
* The RBM repeats this process: feed *reconstructed node values* of our inputs into the *RBM* and we're going to get some *values* for *hidden nodes*. Then based on these *hidden* *values*, we're going to *reconstruct* the *inputs* again, and again. This whole process is called GIBBS SAMPLING.
* At the *end*, we're going to get some *reconstructed* *input* values such that when we feed them into the *RBM*, and then we try to *reconstruct* them again, we will get those *same* *values*. That is the *reconstructed* *input* values *converges* to some values.



* In this final scenario, our network is modeling exactly our inputs. So basically, we can input in and we will always get the same output. This process has finally converged and our network is finally trained.

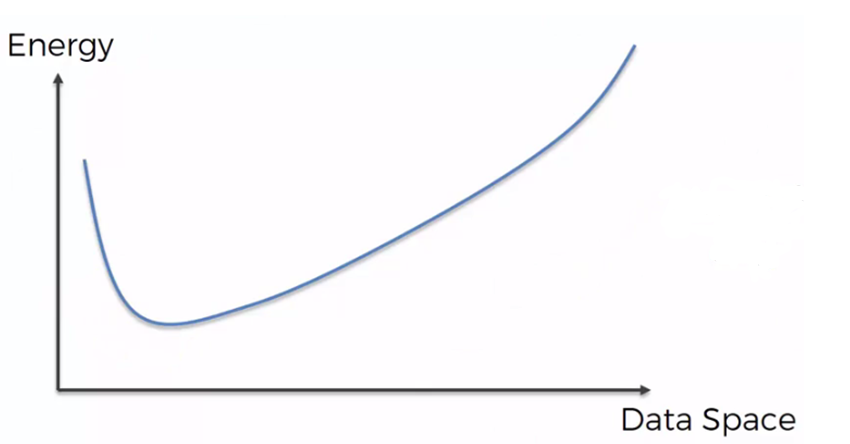
**14.3.2 Contrastive Divergence:** Energy graph

In terms of the curve, Contrastive Divergence looks like follows. We've got two parts here, we'll start with the formula.



|  |  |
| --- | --- |
| * Gradient Formula: This is a gradient formula, can see the gradient on the left here. We've got the *gradient* of the *log probability* of a *certain* *state* of our *system*, based on the *weights* in the *system*. |  |

* Weights are Constant: Remember, through this whole process, the weights are constant. We're not changing the *weights*, we're just using *random* *weights*.
* When we talked about energy based systems we said in RBM energy is defined through the weights. Following is the curve for the weights, "Energy" is actually weights here.



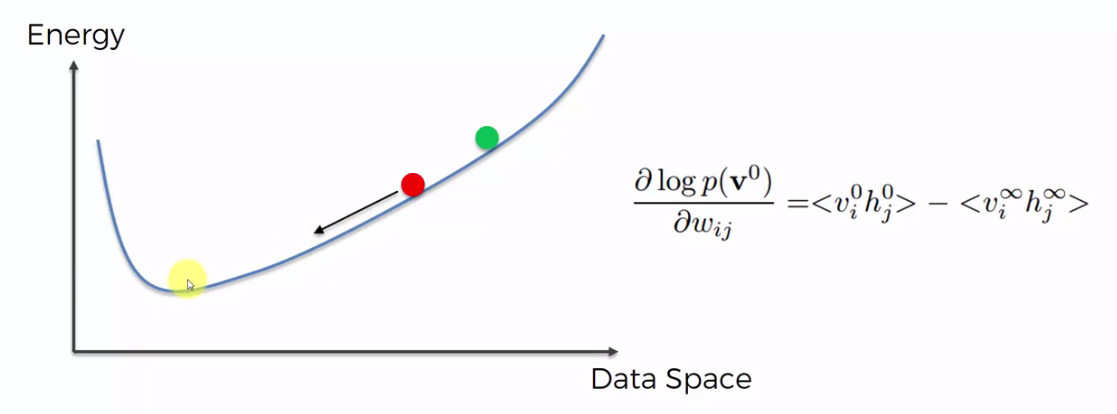
* Since we are using random weights, reconstructed input-values won't be same to our real-input-values even though the system is converged. We need to adjust the weights in such a way that the system reach to the low energy state.
* It's telling us is how the weights affect the *log probability*.
* is the *initial* *state* of the system, is *visible* *vector*, is *hidden* *vector*. represents *next* *states*.

|  |  |
| --- | --- |
| * The *weights* are *fixed* and we're going to define this *energy* *curve* is based on the *weight*. * *Weights* dictate the *shape* of this *energy* *curve*, * For our *first* *pass* through the *RBM* we place our *initial* *inputs* (the *green* *point*, since the *weights* are initialized *randomly* it could be anywhere). |  |
| * After the *second* *pass*, we end up at the *red* *point*. Since a system which is governed by its energy will always try to *end* up in the *lowest energy state* possible. * So as you can see, this *ball/point* is rolling towards the *bottom* (minima). * That's exactly what's happening through that Contrastive Divergence process, where *reconstructed* *input* values *converges* to some values. * We're going *closer* and *closer* to our *lowest* *energy* *state*. |  |

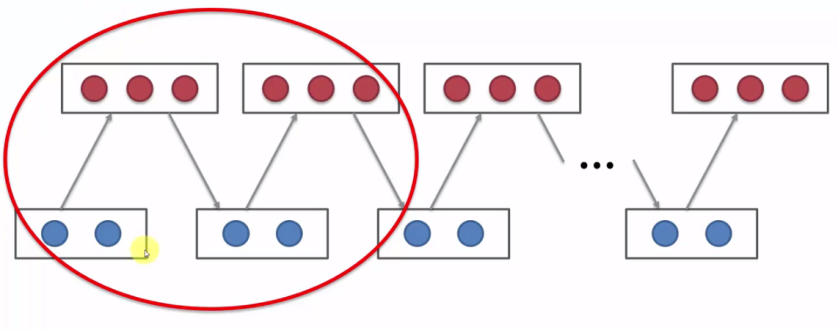
* But remember, the weights are not changed, but we get certain *reconstructed* *input* values and certain hidden values, that bring the system to the minimal energy state at the end of this Contrastive Divergence process.
* And what this formula is telling us is, once you have that *lowest* *energy* *state* if you subtract value from value, it will tell you how adjusting your weights will affect the log probability of the system being in this *lowest* *energy* *state*.
* So basically, this formula is a recipe for *adjusting* your *curve* or for modifying your *energy* *curve*, so that you can make sure that *initial* *state* (where we have *initial-input-values*) is inside an *lowest* *energy* *state*, so that you can get an desired effect.

[From this formula we adjust the weights in such a way that the *reconstructed* *input* values are same as initial-input-values.]

|  |  |
| --- | --- |
| * Right now (before adjusting weights) it's like getting towards a certain minimal energy state but the inputs are completely different to our real-inputs, we want to change that. * We want to use this gradient formula to *adjust* our *curve*, so this, the *energy minimum* is actually *next* to our inputs rather than some *random reconstructed* inputs which are defined by the *randomly initialized* weights. |  |



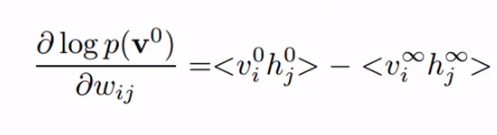
* Hinton's- shortcut: In 1998, Jeffrey Hinton discovered a shortcut that: *"Even if you take just the* ***first******two******passes****, you* ***don't wait*** *until it* ***convergences*** *to the* ***end****. This is* ***sufficient*** *to understand how to* ***adjust*** *your* ***curve*** *as far as is the* ***initial******stage****."*
* CD-1 means Contrastive Divergence **1** pass. You might hear that term CD-1, CD-3, CD-5, CD-9, So if you do a CD-1, Contrastive Divergence one pass, its enough for you to know which way the ball/point is rolling.



* It is kind of similar to what we had in gradient descent's downhill/uphill. In *gradient* *decent* we just had a curve and we were like *finding* the *minimum*.
* But here we have *control* over the *curve*, now want *to adjust/change* this *curve* so that the *minima* will be at the *point* where *we want*. We are *adjusting* the *weights* because it's an *energy based process*.
* We're adjusting the weights so that minimum is actually going to be at our *initial state* rather than some *random* *state*.
* Since in our case the *balls* are going *downhill*, we *pull* this curve *down* where the *green-ball* is, make it *minimum-energy-state* and we want to *push* it up over the point where the *red* *ball* is.

|  |  |
| --- | --- |
|  |  |

* So you can see green-ball (initial state) is already inside the minimum, and we don't even have to go through *the long process of sampling* to get to that *recipe* of how to *adjust* the *curve*, but you can just *adjust* the *weights* using *Hinton's- shortcut*.
* So in a summary:
* We have an *energy* *curve* and the *shape* of this *energy* *curve*, is governed by the *weights* in the system,
* We also know the *RBM* will always get to the *minimal* *energy* *state*. Minimal energy state found through the *Gibbs* *sampling* *process*.
* Gibbs sampling process: *where* ***RBM*** *feeds* ***input****, set the* ***hidden******values*** *and then it* ***reconstruct*** *the* ***inputs*** *then it* ***feeds*** *again from* ***reconstructed******inputs*** *and the process goes on until* ***reconstructed******inputs*** *converges to* ***some******values*** (at **lowest energy state**).
* We then *redesign* the *system* by *redesigning* the *energy* *curve*. We *adjust* the *weights* in such a way that when we *input* our values, our *training* values, the system is *already* going to be in the *lowest energy possible*.
* And for that we use this Gradient Formula.



1. We start with some randomly initialize weights,
2. We input a value like one of our rows into the RBM,
3. We go through this process of Gibbs sampling, we calculate, we find out how to adjust our curve.
4. Plus on top of all of that, there's *Hinton's- shortcut*. We don't actually have to go through to the very end of the sampling process,

* We can just do *two* *passes*, we go *first* *pass*, *second* *pass*, we do a CD-1, Contrastive Divergence one, and that will tell us how to adjust the curve.

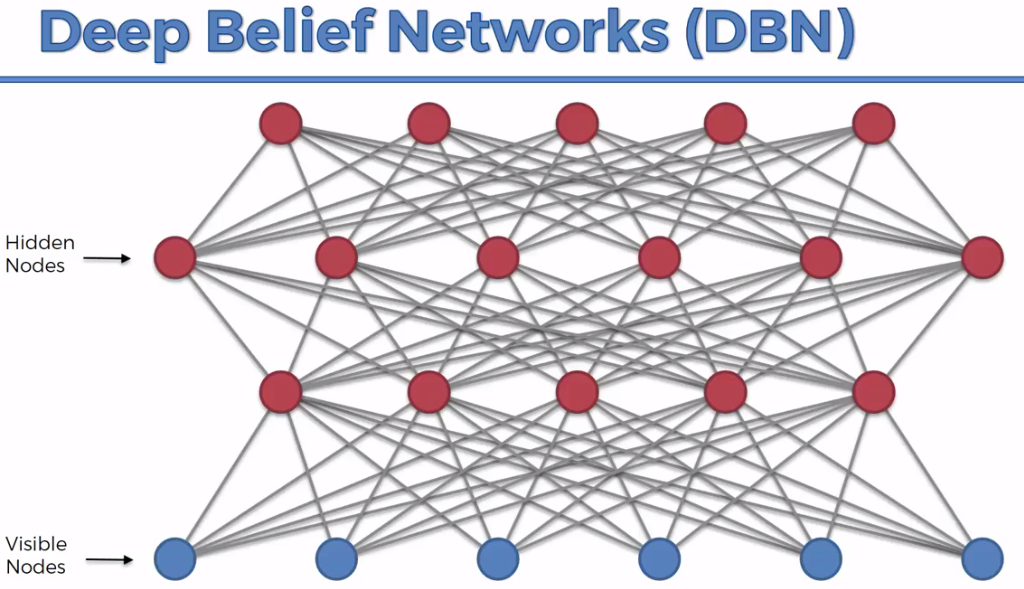
1. We're trying to adjust the energy curve by modifying the weights, in order to create a system which in the best way possible resembles our input values, our training values and we do that, using above gradient formula.

**14.3.3 Additional Reading**

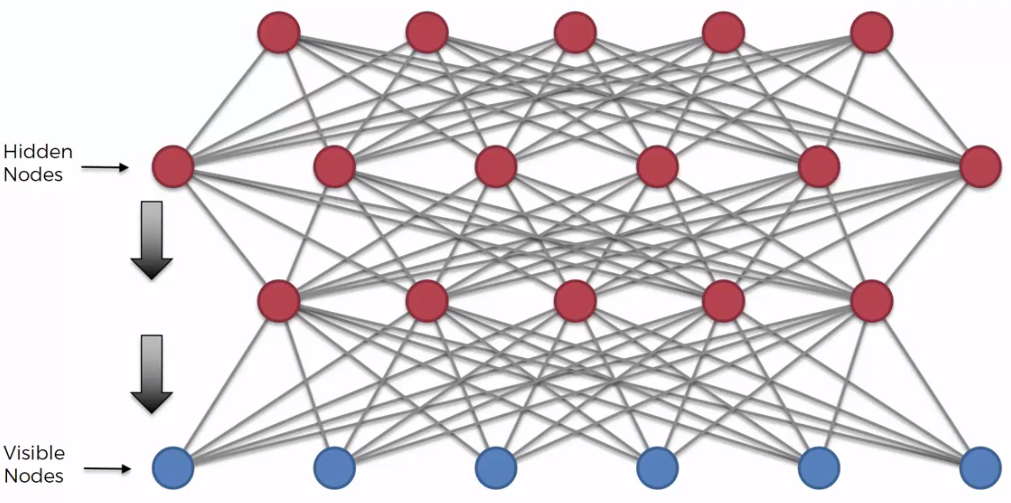
|  |  |
| --- | --- |
| * If you'd like to get into more depth on this topic, on ***Contrastive Divergence***. The first paper is by Jeffrey Hinton and others 2006, it's called a Fast Learning Algorithm For Deep Belief Nets. And you can see exactly the diagram here which we discussed. |  |
| * Another paper if you'd like to get a bit more mathematical on the Contrastive Divergence and really understand the math behind it. And what's exactly going on with the gradients and so on, a good paper to look at is called Notes on Contrastive Divergence, it's not actually a paper it's just some notes is a three pager by Oliver Woodford. |  |

**14.3.4 Deep Belief Network (DBN)**

* DBN: A Deep Belief Network (DBN) is a stack (on top of each other) of several RBMs. Where the 1st RBM's hidden nodes are the input nodes of 2nd RBM and 2nd RBM's hidden nodes are the input nodes of 3rd RBM and so on.



* Directed & Undirected connections in DBN: In a Deep Belief Network, you make sure that your directionality is in place for all of the layers except for the top two.
* Basically you make sure that these layers *one*, *two* and *three*, and there are *directed connections* between them, and they are directed downwards. Whereas there is no direction in the top two layers *(undirected connections).*
* It's quite hard to explain what's going on here because this is a very *complex*, a very *advanced* type of *network*. When *Hinton* and his *team* find that's *DBN idea* when the whole *interest* in *Deep Learning* got revived in *2000's*.



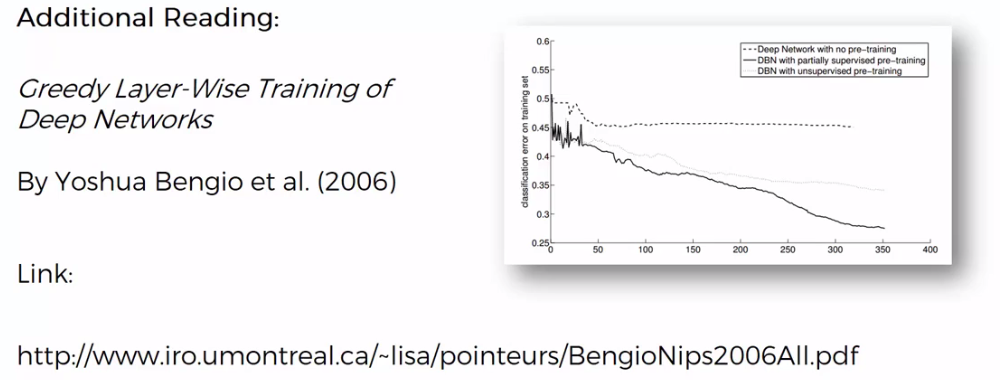
* In terms of training, there's two types of algorithms in DBN

1. Greedy layer-wise training algorithm: First you train the RBMs, with the undirected connections. You train them layer by layer as RBMs, 1st layer, 2nd layer and then 3rd layer (down to top). The directionality is set it up after you've trained up the weights.
2. Wake-sleep algorithm: It is basically you train all the way up, then you train all the way down your layers.

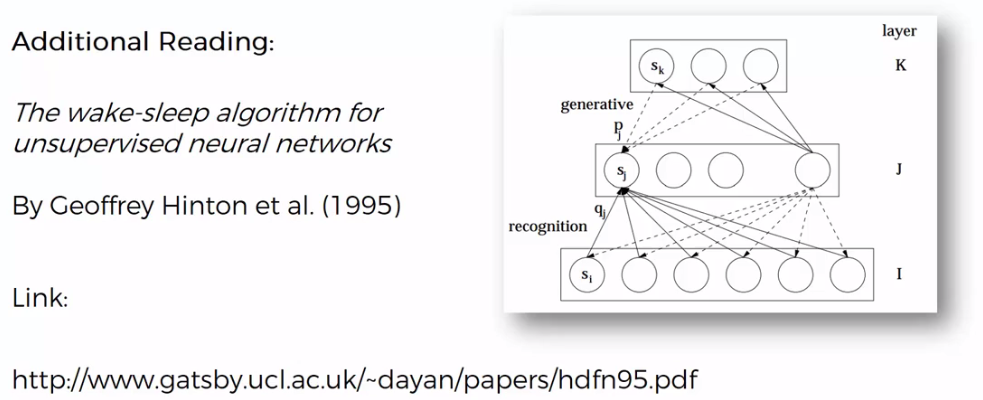
* In simple word DBN is: You stack up RBMs, you train them up, then you, once you've got the weights, you make sure that these connections only work downwards (except top two layers).
* If you ever do need to use it in practice, if you have to design your own DBN, then you have to do some research and read some papers and understand the design that goes onto it.

**14.3.5 Additional Readings**

* The *Greedy layer-wise training of Deep Belief Networks* is an article by *Yoshua Bengio* and others.

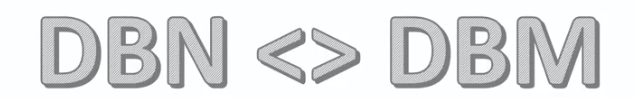


* But the article that we mentioned in the previous section (14.3.3) by Jeffrey Hinton and others 2006, it's called a Fast Learning Algorithm For Deep Belief Nets. Where we were talking about how the *training* of an *RBM* happens. Well that article actually also has, that's where the *Greedy layer-wise* is described before it was used in *Bengio's* article.
* So, make sure to check out that article first, and then move onto the above paper by Bengio. There they explore the whole concept of *Greedy layer-wise training* further.
* If you do want to get into DBNs more then those two articles, those two papers, will help you get, actually do exactly that. You will also learn about Greedy layer-wise training and how it's happening.
* Another one is the *wake-sleep algorithm* by Hinton. This one is about the *wake-sleep algorithm*.

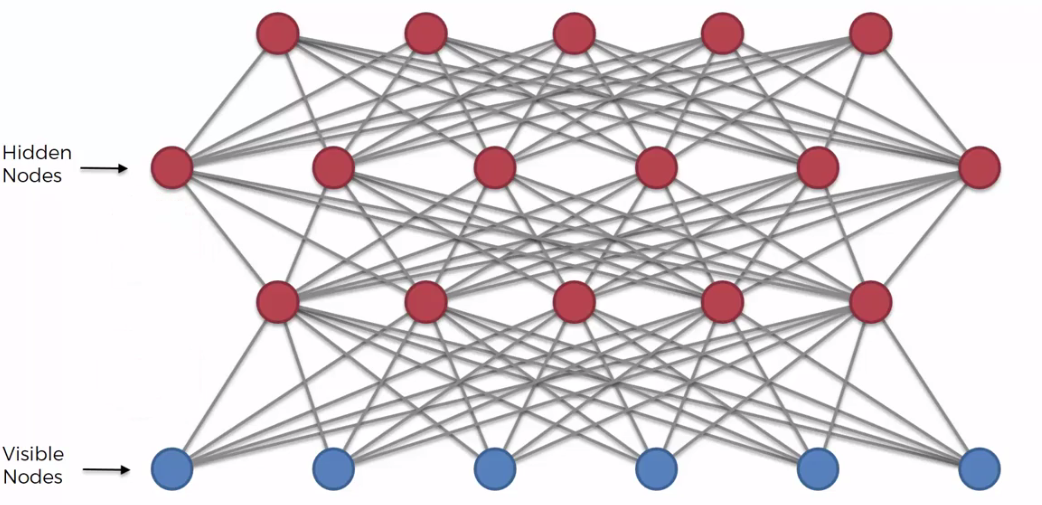


* If you feel that you need to get into DBNs, and it's something that you need for your work or something you need for a project or you want to explore further, these are the 3 papers that are a good start to get you into the world of Deep Belief Networks.

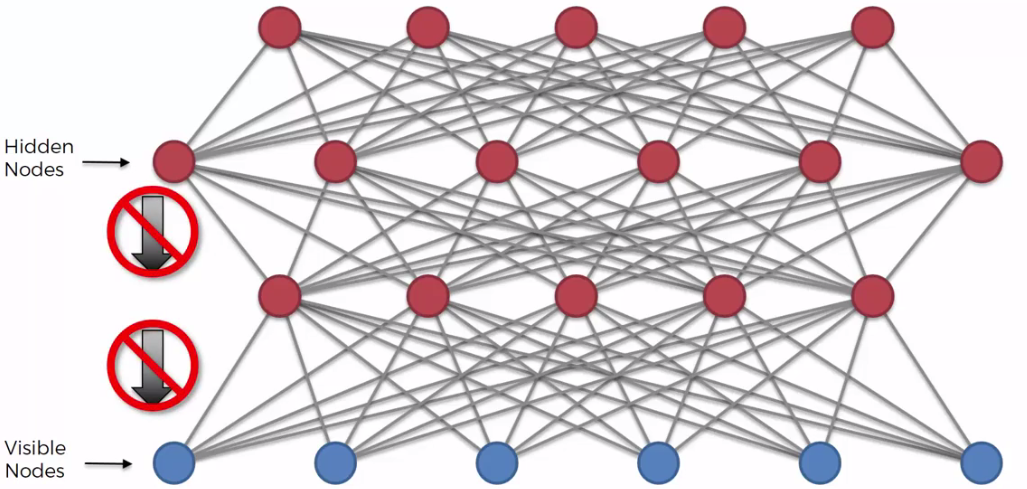
**14.3.6 Deep Boltzmann Machines (DBM)**



* *Deep Boltzmann Machines (DBM)* is another topic, just like the *DBNs*. *Deep Belief Networks (DBN)* are not the same as *Deep Boltzmann Machines (DBM)*. Deep Boltzmann Machines actually can extract features that are more sophisticated, more complex, and therefore they could potentially be used for more complex tasks.



* The main difference is, in DBM is totally undirected network: Previously in DBN we had stacked RBMs, after the training has happened, you make sure that *all of your layers*, *except* for the *top* *two*, are *directed* *layers* (connections between them are directed downwards).
* In *Deep Boltzmann Machine (DBM)*, you don't have that, *no* *directed* *layers*. But in *DBM* we also have *stacked* *RBMs*.



* Additional readings: There's a great paper, which we're going to direct you towards in terms of Deep Boltzmann Machines. It's called *Deep* *Boltzmann Machines* by *Rusian Salakhutdinov*. Hinton is also a co-author here.
* *Ruslan* actually has *quite* *few* *papers* on *Boltzmann* *Machines* and *RBMs* and stuff like that.

