Chapter 15 : Part 1

**Deep Learning**

**Auto Encoders (AE)**

Introduction

**15.1.1 What we will learn in this chapter**

1. Auto Encoders: What are ***Auto*** ***Encoders***, what is the architecture and how do they work.
2. Training of an Auto Encoder: We'll talk about Training of an Auto Encoder and the steps that go into that.
3. Overcomplete Hidden Layers: We'll talk about a situation where we have ***Overcomplete*** ***hidden*** ***layers*** in an ***auto*** ***encoder***.

We'll And then we'll talk about three regularization techniques:

1. Sparse Autoencoders
2. Denoising Autoencoders
3. Contractive Autoencoders
4. Stacked Autoencoders
5. Deep Autoencoders

Note: Starting from over *Complete* *Hidden* *Layers* to *Deep Auto Encoders*, are more of a high level overview. We'll go through a quick introduction to each one of these variations, at least we will know what is the reference to when we hear them.

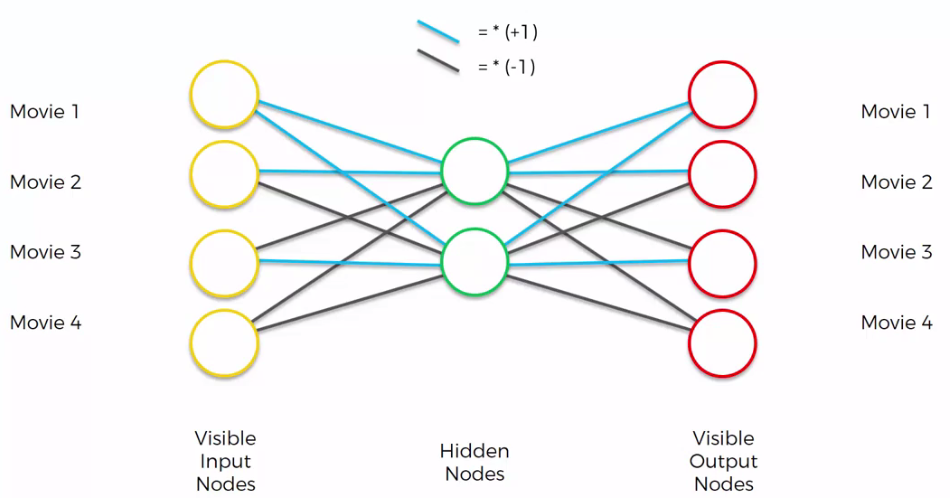
**15.1.2 Auto Encoder (AE)**

|  |  |
| --- | --- |
| Right-side diagram is what an *Auto* *Encoder* looks like. This is a *directed* types of *neural networks*.   * The *blue* *lines* don't have *arrows* on the *ends*, but we'll consider it as a *directed type* of *network* and everything's *moving* from *left* to *right*. * How Auto Encoder work: An Auto Encoder encodes itself, that's the whole philosophy behind the Auto Encoder. * It takes some sort of *inputs*, put some through a *hidden* *layer*, and then it gets *outputs*. The point is: it aims for the *outputs* to be *identical* to the *inputs*. * Training an Auto Encoders: We set Auto Encoders in such a way that on the output you get values which are equivalent to your inputs. |  |

* Auto Encoders are self-supervised: We can say that, *Auto* *Encoders* are not a *pure* type of *Unsupervised Deep Learning algorithm*. They are actually a *Self-Supervised Deep Learning Algorithm*, because they are comparing to something on the end.
* In BMs we *didn't even have outputs*. We didn't have to *compare* to any kind of *labels or anything*.
* In SOMs, we *didn't have anything to compare* *to,* we're just looking for *features*.
* However, in Auto Encoders we are looking for *hidden* *layer* (also called the *coding* *layer*, or the *bottleneck*). We're looking for how to structure the *hidden* *layer*, but at the same time, we are *comparing* the *outputs* to the *inputs*.
* In short ward: You have *inputs*, they get *encoded*, and then they get *decoded* and then *compared* to *inputs*, that's how the training happens.
* You can already imagine how information is *propagated* *forward*, and then you have *gradient* *descent* then it *propagate* *backward*. We'll talk about all those things next.
* Usage of Auto Encoders: Well, there's a couple of things that they can be used for.
* They can be used for feature detection: Once you've *encoded* your data, these *hidden-nodes* will represent *certain* *features* which are *important* in your *data*, and then you can just look at them and get those *features* out of them, or use those *features* in the *future*.
* They can also be used to build powerful recommender systems.
* They can be used for encoding: Actually *Auto* *Encoders* are designed to *encode* data. You feed it with lots and lots of values, *encoded* into a *smaller* *representation* and then all you'll have to carry around the *decoder part* and *encoded* *data* it would take up *less* space.
* Example (walk through): Now let's have a look at an example of how they actually work, so we can understand them better on an *intuitive* *level*.

|  |  |
| --- | --- |
| * Here is our simplified Auto Encoder with four input nodes, and two nodes in the hidden layer. * As we can see, we've got four movies at left and four movies at right. * These are the movies that a person has watched, and we're going to encode the rating for those movies. **1** = liked that movie, and **0** = didn't like that movie. |  |
| * This example actually comes from this blog, "PROBABLY DANCE". It's a great blog, it's by a person who's actually a *programmer* who isn't a *Deep Learning scientist*, but he really broke it down into good steps. * Now let's have a look at how this information can be encoded through the *Auto* *Encoder*. |  |

* Firstly, we establish *connections* with certain *weights*. To prove that it is possible to take *four* *values* and *encode* them into actually *two* *values*, and carry your data around, *save space* and *extract* those *features*.
* For now we're ***not gonna worry*** about the training.
* We're going to color our synapses in two different colors, ***blue*** and ***black***.
* ***Blue*** is basically a multiplication by "**+1**". Weight is plus one.
* ***Black*** is a multiplication by "**-1**". Weight is minus one.
* Also in Auto Encoders, we normally use the Hyperbolic Tangent Activation function (ranges between **[1, -1]**).

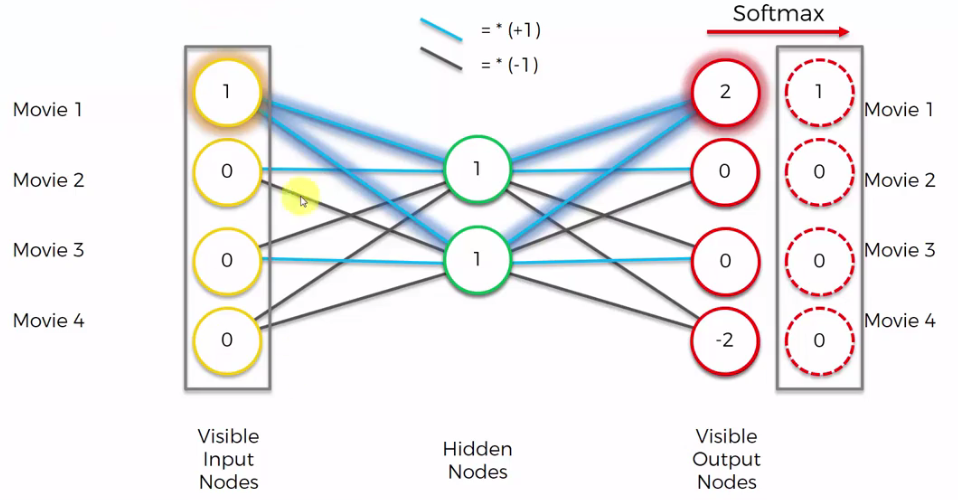


* Let's have a look at an input. Let's say as an input, we've got **1** for first input, and **0** for others. Means that the person just like Movie 1, and dislikes the rest of the movies.

|  |  |
| --- | --- |
| * In that case, hidden nodes will be, "**+1**" and "**+1**" because blue synapse is multiplying by "**+1**". And the **0'**s in the *input* *nodes*, they will always just *add* *zero*, and *not* going to *contribute* to the *hidden* *nodes*.   For the first output-node:  1\*(blue-synapse) + 1\*(blue-synapse) =  (1\*1) + (1\*1) = 2  For the 2nd and 3rd output-node:  1\*(blue-synapse) + 1\*(black-synapse) =  (1\*1) + (1\*-1) = 0  For the 4th output-node:  1\*(black-synapse)+ 1\*(black-synapse) =  (1\*-1) + (1\*-1) = -2 |  |

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

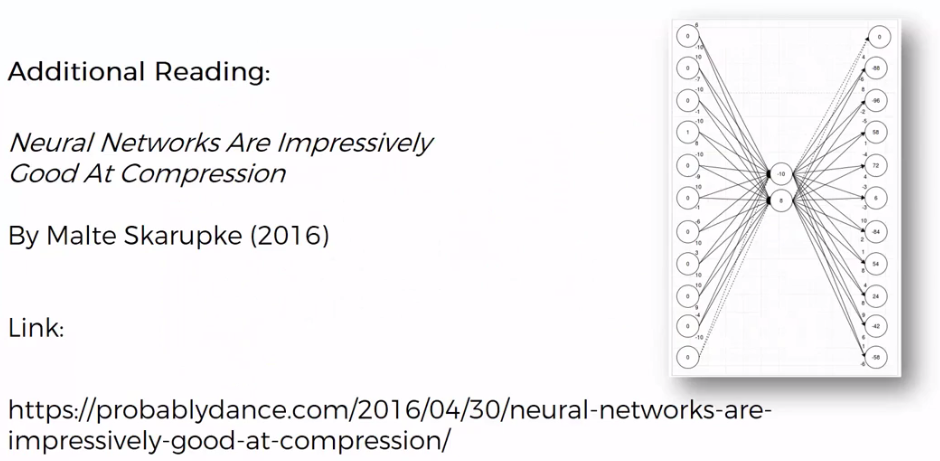
* So those are your *outputs* but those are actually *preliminary outputs*. In Auto Encoder we also have a SOFTMAX function on the end. As you can see, *final* *output* *result* is indeed identical to our *input*.
* Softmax function takes the highest value, so in this case it is "**2**" and it turns that into "**1**" and everything else into "**0**".



* Let's have a look at some other cases. We don’t go to the details.

|  |  |
| --- | --- |
|  |  |
| * Notice how the values inside hidden nodes changes from "**1**" to "**-1**". * You can see, in our data set, we've got rows with three **0'**s and an **1** using *4 nodes*. Then we can encode them into a small format where we just have *2 hidden nodes*. |  |

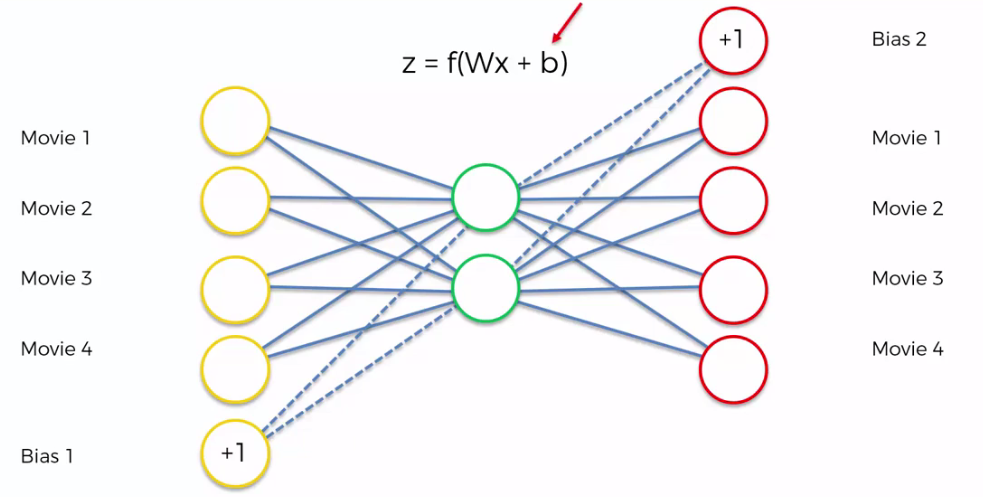
* Every state, is represented by a hidden layer using these hidden nodes *{1, 1}* or *{1, -1}* or *{-1, 1}* or *{-1, -1}*.
* Then you just need these weights and Softmax function to reconstruct your output.
* It is a very *simplified* example, but this gives a overview of how *Auto* *Encoders* work. Of course *Auto* *Encoders* are much more complex than that.
* Additional reading: For additional reading this is the same blog that we already mentioned it's called, *"Neural Networks Are Impressively Are Good At Compression"* by *Malte Skarupke* and will include it in the additional resources.
* Nicely written very *easy* *introduction* into *Auto* *Encoders*. Here you can see a much more *sophisticated* *example*.
* It's not even actually mentioned that it's Auto Encoders from the very start, just Neural Networks, and then in the comments you can read that indeed they were talking about Auto Encoders.



**15.1.3 Biases in Auto Encoder (AE)**

We've got our *simplified version* of *Auto Encoders* and sometimes you'll see a bit of a *different* *representation* as follows. The "**+1**" on both side with the *dashed connection* are just *biases*.

* *Biases* means that in your *activation* *function* you've got a constant **b** in weights, bias

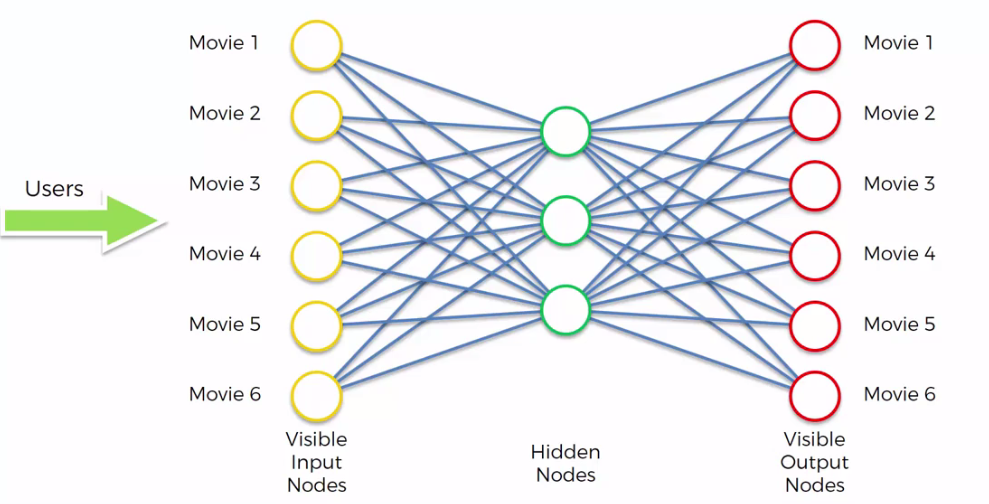


|  |  |
| --- | --- |
| * The more *correct* *representation* is given in right. This is how it should be represented if people are showing biases. * *Bias* is just added there because *bias* *affects* the *layer* *ahead*, the layer that's in front. * If you see that just remember it's just basically a *bias* in it. It's just a *constant* in that *equation*. |  |

**15.1.4 Training an AE**

Here we walk through the steps to training an Auto Encoder. Here we've got an Auto Encoder, where we've got some inputs: the *ratings* of lots and *lots of users* for these *six movies*, and then we'll get some outputs.

* Based on those ratings, Auto Encoder will come up with a way to *compress* that *data* and *adjust* the *weights* so that it set itself up to be able to *encode* *the data and decode the data in the future*.



***Steps for Training an Auto Encoder***

* STEP 1: We start with an *array* where the *lines/rows* (the observations) correspond to the *users* and the *columns* (the *features*) correspond to the *movies*. Each cell **(u, i)** contains the rating (from ***1*** to ***5***, ***0*** if no rating) of the ***movie i*** by the ***user u***.
* Here every single row in your data set is a unique user who rated those movies.
* STEP 2: The *first* *user* (first row) goes into the network. The *input vector* contains all its *ratings* for *all* the *movies*. i.e all the ratings of that user for all of the movies.
* STEP 3: The *input* *vector* is *encoded* into a *vector* of *lower* *dimensions* by a *mapping* *function* (e.g: sigmoid function or a hyperbolic tangent):

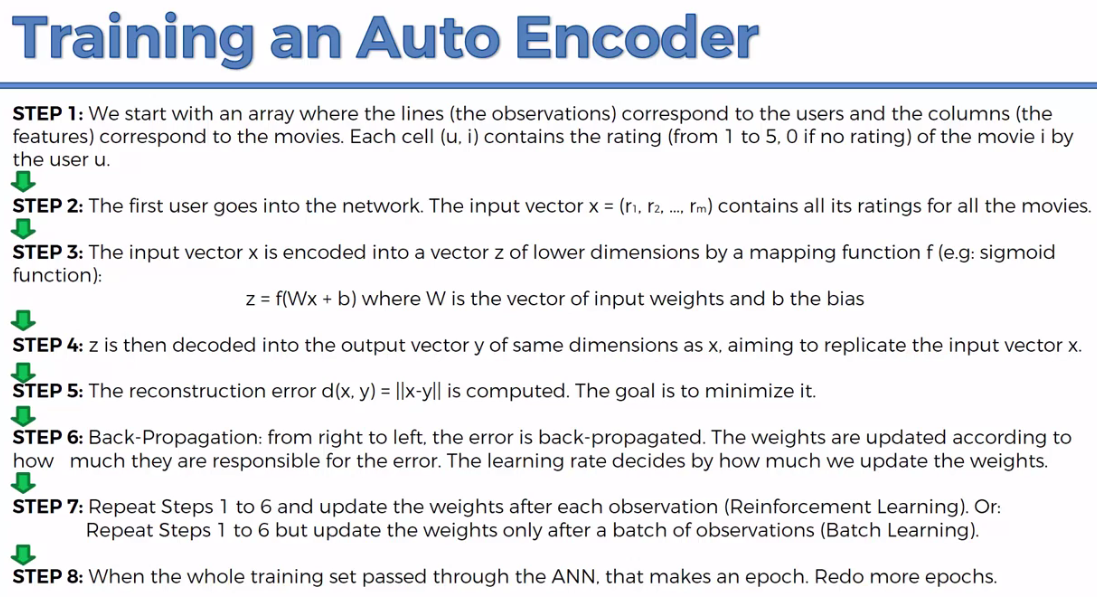
where is the *vector* of *input weights* and the *bias*

* STEP 4: is then *decoded* into the *output vector* of ***same dimensions*** as *input-vector* , aiming to *replicate* the input *vector* .
* STEP 5: The reconstruction error (which is how ***different*** is the ***output*** compared to the ***input***)is *computed*. The goal is to minimize .
* STEP 6: Back-Propagation: from *right* to *left*, the error is back-propagated. The *weights* are *updated* according to how much they are *responsible* for the *error* ( so you've got a ***Gradient Decent process*** happening there) The *learning* *rate* decides by *how much* we *update* the *weights* (that's a parameter you can tune during decoding and you will see that in the practical example).
* STEP 7: Repeat *Steps 1* to *6* and *update the weights* after each observation (*Reinforcement* Learning).

Or

Repeat Steps 1 to 6 but *update the weights* only after a batch of observations (*Batch* Learning).

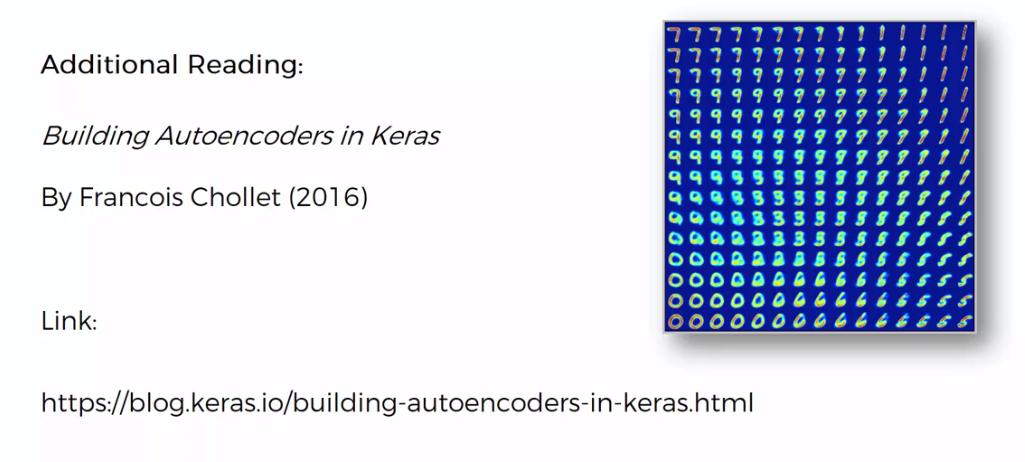
* STEP 8: When the whole *training* set passed *through* the *ANN*. that makes an *epoch*. Redo more *epochs*.



* Steps walkthrough an example: Here we have ratings ***0*** and ***1*** but we can have ratings from ***1*** to ***5*** and ***0*** for empty (it will be a more powerful recommender system). In this case we've got 1, 0 and empty, we can easily change them to, "**+1**", "**-1**" for ratings and **0** for the empty cell.

|  |  |
| --- | --- |
| 1. That's our *input*. |  |
| 1. We take the *first* *row*, we put it into our *Auto* *Encoder*, |  |
| 1. We calculate the *hidden* *nodes*. At the very start you'll have some *randomly* *initialized* *weights*. This process is called *encoding*. |  |
| 1. We're going to calculate our *visible* *output* *nodes*, again initially it will be some *random* starting *weights*. This process is called decoding.   That’s the *forward* *propagation*. We've just put the information, the data through our *Auto* *Encoder* from *left* to *right*. |  |
| 1. We're going to *compare* the results to the *actual* *ratings* for those *six* *movies*. And then we'll calculate the *error*. |  |
| 1. And then we will *propagate* it *back* through the network, we'll *adjust* the *weights* accordingly. |  |
| 1. We will take the *next* *row* in our data set. And we'll do the same thing and so on. We'll *continue* through all the *rows*. |  |
| 1. It means we've finished a whole *epoch*.   And then we just *repeat* these *epochs*.  Hopefully, our training *converges* to some sort of *value*. |  |

* Additional reading: Here is a blog by *Francois Chollet*, the creator of *Keras*, it's called Building Autoencoders In Keras, check it out, it's got some actual practical applications so even if you want to get some hands on experience before you start to Practice.



**15.1.5 Overcomplete hidden layers in AE**

|  |  |
| --- | --- |
| *Overcomplete hidden layers* is a underlying concept in most of the variations of Autoencoders. So here we've got an *autoencoder*, four *input* *values*, two *nodes* in the *hidden* *layer*, and four *nodes* in the *output* *layer*.   * What if we wanted to increase the number of *nodes* in the *hidden* *layer*? * What if we wanted actually to have more nodes in the *hidden* *layer* than in the *input* *layer*? |  |
| * Why would we do that? We said that an *auto encoder* can be used as a *feature* *extraction* tool, what if we want *more* *features*. * Number of nodes in hidden layer in ANN doesn't matter: Remember in ANN it was very easy for us to do, we had a *certain* number of *inputs*, then we could have a whole *layer* of *hidden* *nodes* that could have *more* *nodes* than *input* *layer* (It could be six, it could be 10, it could be 100, doesn't matter). We could add more layers and so on. * We weren't restricted to how many nodes we would have in the hidden layer. And that allowed us to extract more features in ANN. |  |

* Increasing nodes in hidden layer in Auto-Encoder can cause problem: If we increase nodes in hidden layer in *auto encoder*, obviously the *auto-encoder* can cheat.
* In the *auto-encoder*, the goal is to get the *outputs* to equate to the *inputs*.
* As soon as you give it a *hidden* *layer* which is the *same* *size* or *greater* than the *input* *layer* (in this example four or more hidden nodes), makes *auto-encoder* able to cheat.
* The information is just gonna ***fly*** ***through*** the ***hidden*** ***nodes*** , and no encoding-decoding happens, also ***extra hidden-nodes*** remain ***unused***.
* After the training and is going to be just useless it's not going to extract any valuable information, any valuable features for us through that process.

Next we're going to look at three different approaches that are used to solve that problem.