Chapter 15 : Part 2

**Deep Learning**

**Auto-Encoders (AE) : Regularization & Other types**

3 Regularization techniques & Other types of Auto-Encoders

**15.2.1 Sparse Auto-Encoders:** *Regularization technique 1*

|  |  |
| --- | --- |
| * A sparse AE is an auto-encoder which looks like this diagram, where the ***hidden*** ***layer*** is ***greater*** than the ***input*** ***layer***. But a ***regularization*** technique which introduces ***sparsity*** has been applied. * A *regularization* *technique* basically means something that helps *prevent over-fitting* or stabilizes the algorithm. * In this case, if it was just sending the values through the hidden layer, it would be over-fitting in a way.   *Sparse AE* is one of regularization techniques that prevent over-fitting. |  |
| * Basically *Sparse auto-encoder* introduces a *constraint* on the *loss function*, or a *penalty* of a *loss* *function*, which doesn't allow the auto-encoder to use all of its *hidden* *layer* every *single* *time*. * So that the *Auto-Encoder* can only use a certain number of nodes from it's *hidden* *layer*. * For instance, in this image, *Auto-Encoder* can use two nodes in this case. * When the values *go* through these *disable-nodes*, *output* will be very *small*. |  |

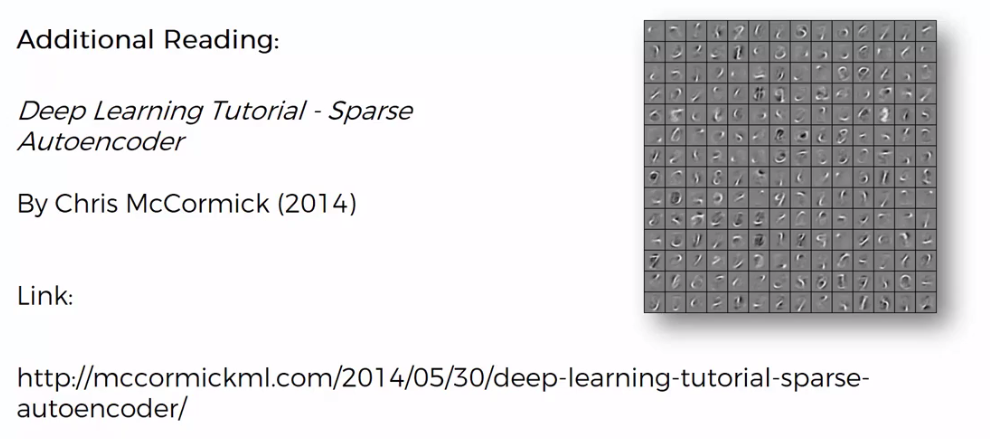
* In following two passes the disable nodes aren't participating on encoding.

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

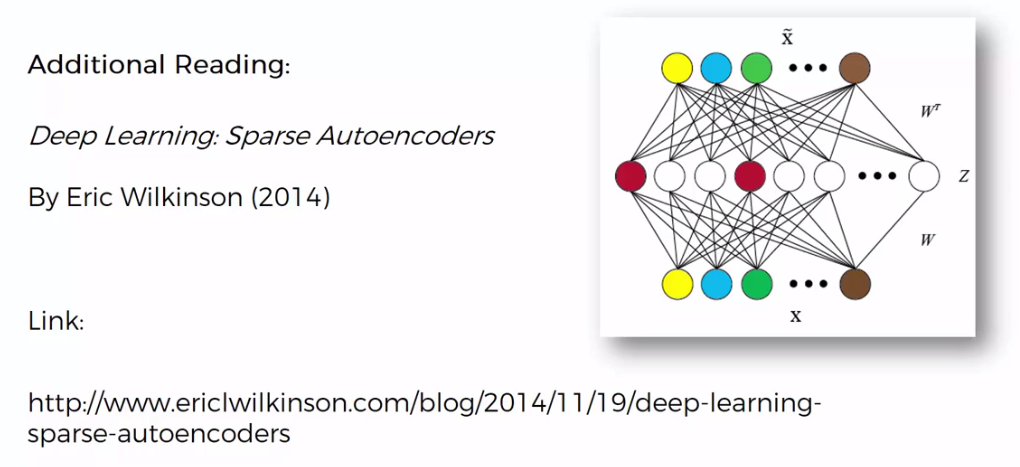
* While you training this whole hidden layer, you are extracting features from each one of these hidden-nodes, but at the same time, you're not using all of these nodes.
* So the *auto-encoder* cannot *cheat* because even though it has *more* nodes in the *hidden layer* than in the *input* *layer*, but it is *not able to use* all of them at any given pass. And that's how the sparse auto-encoder works.
* In reality, it is still compressing the information but just every time is using different nodes.

**15.2.2 Additional Reading**

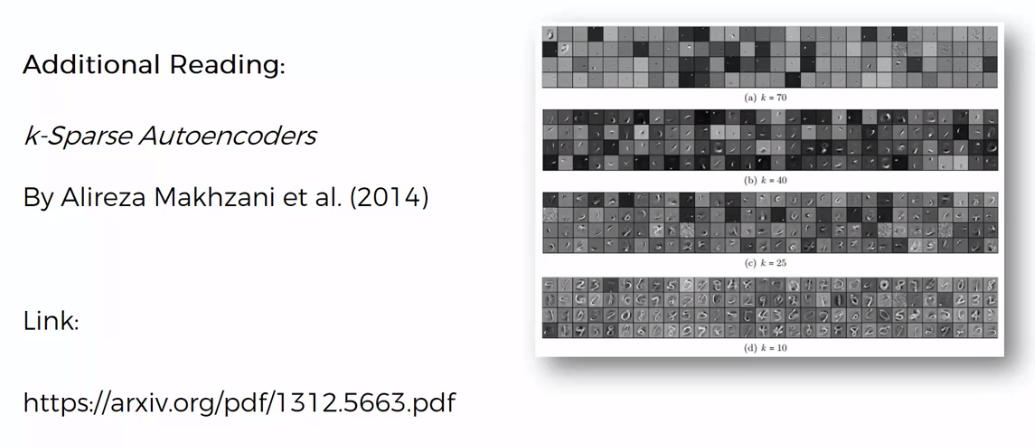
* We've got a interesting tutorial here by *Chris McCormick* is called *Deep Learning tutorial Sparse Autoencoder*. And this is a *MATLAB* tutorial. He will walk you through how to do it in *MATLAB*.
* There's some ***introductory*** ***mathematics***, not too complex, but some basic ***level formulas*** which allow you to gently get introduced to the ***mathematics*** behind the ***sparse Auto-Encoders***.



* If you want to understand how the equations work which don't allow the auto-encoders to switch on all of its nodes at the same time, at any given point, at any given pass.
* The next one is called, *Deep Learning Sparse Auto-encoders* by *Eric Wilkinson*. It's a very short blog post on the essence of sparse auto-encoders.



* A very strong powerful, heavy artillery paper on sparse Auto-Encoders. It's called, *K-Sparse Auto-encoders* by *Alireza Makhzani*, 2014. You might find this paper interesting if you want to learn more about sparse encoders.
* *K-Sparse Auto-Encoders* used all over the place and it's constantly mentioned.



**15.2.3 Denoising Auto-Encoders:** *Regularization technique 2*

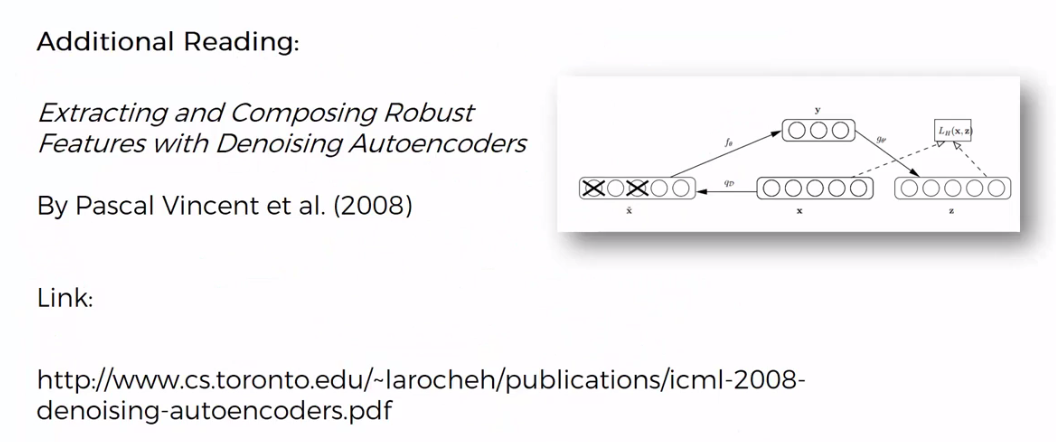
Denoising Autoencoders is another regularization technique, a quite a popular technique actually, which resolves the problem when ***"we have more nodes in the hidden-layer than in the input-layer and the Auto-encoder simply copy input values to output-values without finding any meaningful features"***.

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

* Here we are going replace the *actual input-values* with *noised-input-values*.
* *Noised-input-values* are the *input-values* where we randomly make some of the *input-values* equal to ***zero***.
* *Noised-input-values* are modified version of our ***input values***. So, let's say we have input values ***{X1, X2, X3, X4}***. Now we're going to take these inputs randomly and turn some of them into zeros, ***{X1, 0, X3, 0}***.
* You can specify the number of **0**'s in the *setup* of your *Auto-encoder*, it can be, for instance, ***half*** of your ***inputs*** that you have are turned into ***zeros*** every *single time* and it happens randomly. At every single pass it can be different variables.
* Also it is important to note that because this happens randomly, this type of Auto-Encoder is a Stochastic Auto-Encoder.

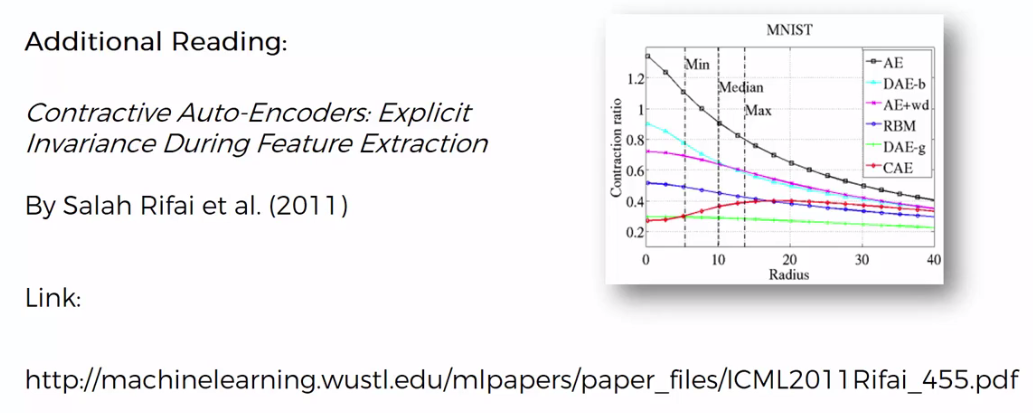
|  |  |
| --- | --- |
| * Once you put this data through your Auto-Encoder then in the end you compare the output, not with the *modified/noised-input-values*, but with their *original input-values*. * And that *prevents* the *Auto-encoder* from simply just *copy* inputs all the way through to the *outputs* because it's actually comparing the *output*, not with the *noisy* but with the *original inputs* and that helps *resolve* the problem that we are facing. |  |

* Additional Reading: In terms of additional reading, here is a great paper by Pascal Vincent and others, 2008, it's called *Extracting and Composing Robust Features with Denoising Autoencoders*.



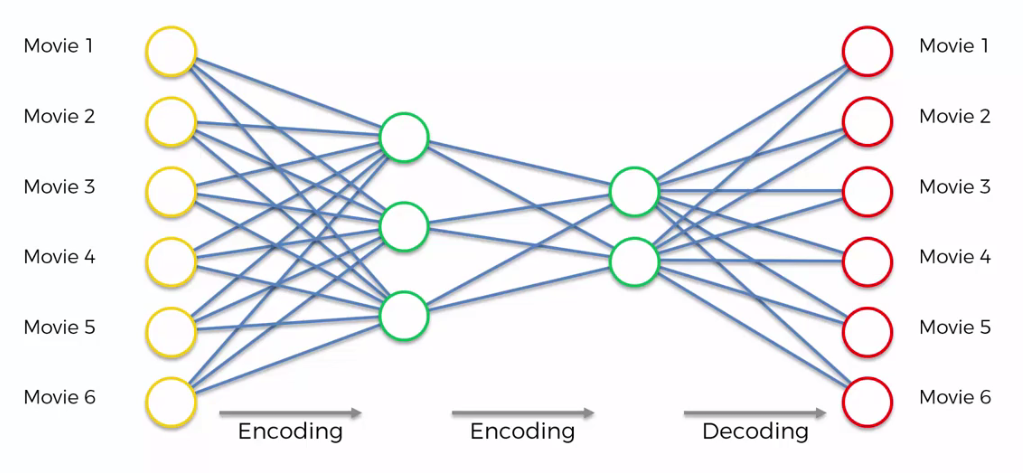
**15.2.4 Contractive Auto-Encoders:** *Regularization technique 3*

|  |  |
| --- | --- |
| Contractive Auto-Encoders is another regularization technique just like *sparse Auto-Encoders* and *denoising Auto-Encoders*. Which is designed to *resolve* the same problem of *over-complete hidden layer* in the Auto-Encoders.   * We are not going too much detail, the *Contractive Auto-Encoders* add a *penalty* into the *loss-function* during the *back* *propagation*. It doesn't allow the Auto-Encoders to just simply copy these values across. * Additional Reading: A wonderful article by **Salah** **Rifai**, it's called *"Contractive Auto-Encoders: Explicit Invariance During Feature Extraction"*. This article has quite a lot of authors and one of them is Yoshua Benjio, so quite an in-depth study into the topic. * Based on this article they say that they can even get better results than Denoising Auto-Encoders on some certain data sets. |  |

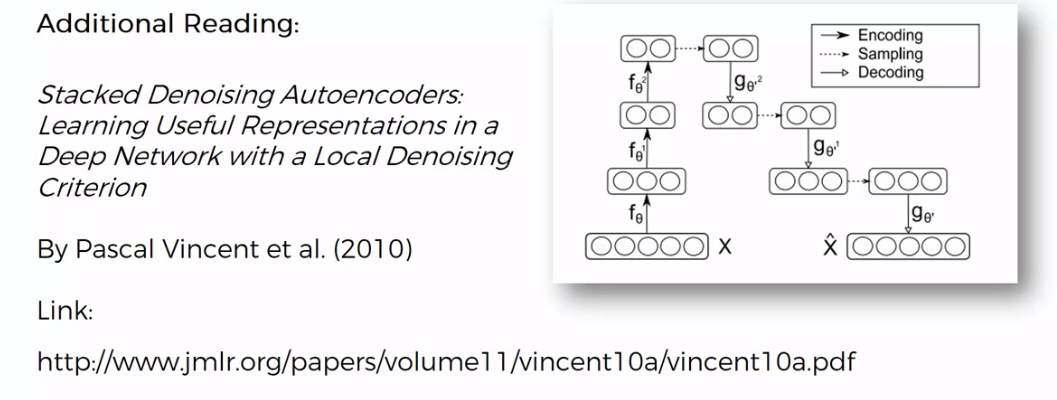


**15.2.5 Stacked Auto-Encoders (AE)**

Here is a Stacked AE. A *stacked AE* is an *AE* with *another* *hidden* *layer*. So we have ***two*** stages of ***encoding*** and ***one*** stage of ***decoding*** here. And what we get is a very, very powerful algorithm.



* In fact it's been shown that sometimes *stacked AE* can supersede the results that are achieved by DBN. Where DBN is undirected NN and is a Boltzmann Machines. But *stacked AE* is a directed NN.
* Additional reading: A great paper to look into on this topic is by *Pascal Vincent* and others is called *Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion*. It's quite a large paper.
* It also has Yoshua Bengio one of the co-authors on this paper as well, and it builds upon their 2008 paper which we have already referenced before.



**15.2.6 Deep AE**

|  |  |
| --- | --- |
| Note that ***Stacked AE*** are not the same thing as ***Deep AE***. |  |

|  |  |  |
| --- | --- | --- |
| * Right side image is a Deep AE. These are actually stacked ***pre-trained***, layer by layer RBMs, * Deep AE basically are RBMs that are stacked. They're pre-trained layer by layer. Then they're unrolled. Then they're fine tuned with back propagation. * Here you get directionality in your network and then you have back propagation. But in essence a Deep AE, comes from RBMs. * Note : * *Stacked AE* are just *normal* *AE* that are *stacked*. * A *deep AE* is *RBMs* *stacked* on top of each other, and then certain things are done with them in order to achieve a *AE* *mechanism*. | |  |
| * Additional reading: If you'd like to learn more about deep AE, a great paper by *Geoffrey Hinton* and others called, *Reducing the Dimensionality of Data with Neural Network*s. |  | |