Lectures on Machine Learning (Fall 2017)

Hyeong In Choi Seoul National University

Lecture 16: Autoencoders (Draft: version 0.7.2)

Topics to be covered:

- Simple autoencoder
- Stacked autoencoder
- Denoising utoencoder
- More autoencoders
- Why deep learning works

1 Autoencoders

One of the key factors that are responsible for the success of deep learning is the method, or a group of methods called unsupervised pre-training. It is a way of preparing deep neural networks in such a way to make the usual backpropagation algorithm work in the intended manner. The unsupervised pre-training idea in one form or another has been around since the mid 1980s and was used here and there. However, it was Hinton and his colleagues who systematically exploited this idea to demonstrate that deep neural networks (DNNs) can indeed be successfully trained. They originally used the generative model called the restricted Boltzmann machine. Shortly afterwards, however, people realized that the much simpler and straightforward feedforward network, called autoencoders, work just as well. There are many forms and variations of autoencoders, but in this lecture, we will cover some of the most popular and most basic forms of autoencoders.

1.1 Simple autoencoder

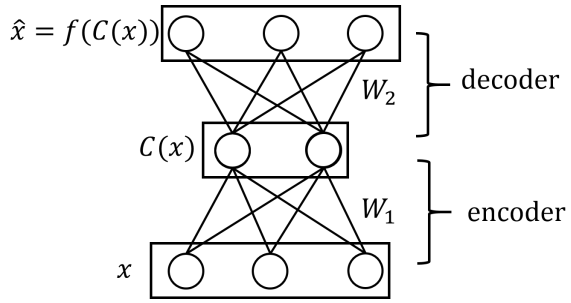

Figure 1: Encoder and decoder

Figure 1 shows a simple autoencoder with one hidden layer in the middle. Although the bottom and top layers show only three neurons and the middle only two, they may contain as many neurons as one designates. The bottom and top layers have the same number of neurons, and typically, but not always, the middle layer has fewer neurons. In such a case, the autoencoder is called an undercomplete autoencoder, and if the middle layer has more neurons, it is called an overcomplete autoencoder. The matrix W_1 is the collection of weights connecting the bottom and the middle layers and W_2 the middle and the top. They are usually, but not always, tied, i.e. $W_2 = W_1^T$. So now let $W_1 = W$ and $W_2 = W^T$. The input x is fed into the bottom layer to produce

$$h = \sigma(Wx + b),$$

in the middle layer, and from this is gotten the output $\hat{x}$ in the top layer

$$\hat{x} = \sigma(W^T h + c).$$

The basic goal of an autoencoder is to make the output $\hat{x}$ as close to the input x as possible. Now it cannot always be done if the number of neurons in the middle layer is much smaller than in the bottom or the top.

In this case, the middle layer value h is sort of like compressed data of the input, and because of this reason, we write

$$h = C(x).$$

So in this sense, the neural network consisting of the bottom and the middle layers is called an *encoder*, and the network made up of the middle and top

layers is called a *decoder*, and we write

$$\hat{x} = f(C(x)).$$

To see how to make this encoder-decoder combination work, let the dataset be $\mathfrak{D} = \{x^{(i)}\}_{i=1}^{T}$. We write

$$\hat{x^{(i)}} = f(C(x^{(i)})).$$

The error is usually the L^2-error given by

$$\begin{aligned} E &= \frac{1}{2}\sum_i |\hat{x}^{(i)} - x^{(i)}|^2 \\ &= \frac{1}{2}\sum_i \sum_k |\hat{x}_k^{(i)} - x_k^{(i)}|^2, \end{aligned}$$

and the training algorithm is the one utilizing the standard backpropagation algorithm of the feedforward network.

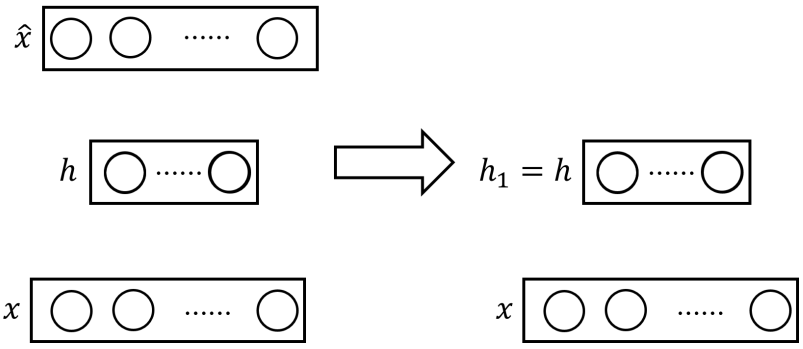

Figure 2: Remove top layer

1.2 Stacked autoencoders

The simple autoencoder described in the above section can be stacked to produce a **deep autoencoder**. We first describe the so-called **layerwise pre-training**. It goes as follows. First, as in Figure 2, the top layer of the simple autoencoder is removed while the bottom and the middle layers remain intact. The value of the middle layer is now renamed as h_1 and by

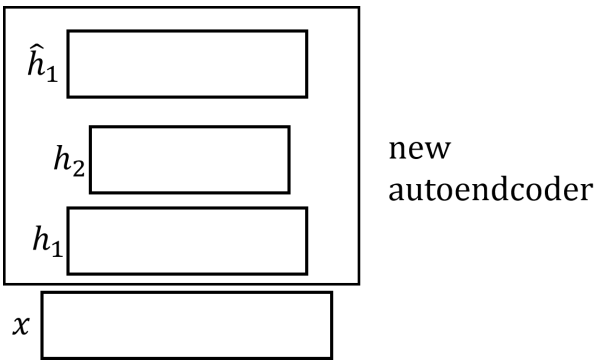

Figure 3: New autoencoder

abuse of language we also denote this layer by h_1. In what follows, we also abuse the language to use the same symbol to represent the name of the layer and its value at the same time.

Second, use layer h_1, the old middle layer, as an input and put two more layers h_2 and $\hat{h}_1$ as in Figure 3. Then the three layers, h_1, h_2 and $\hat{h}_1$, can be regarded as another simple autoencoder on its own right and is trained likewise. In particular, given an original input $x^{(i)}$, h_1 is now regarded as the input for the newly created simple autoencoder $h_1 - h_2 - \hat{h}_1$ as shown in the box of Figure 3.

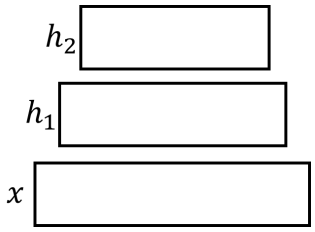

Figure 4: Remove top layer

Now remove the top layer $\hat{h}_1$ and keep h_2. We now have a three-layer neural network consisting of x, h_1, h_2, which is depicted in Figure 4. The connection matrix between h_1 and h_2 is the one gotten from the simple autoencoder $h_1 - h_2 - \hat{h}_1$; and the connection matrix between x and h_1 is the one gotten from the original simple autoencoder $x - h_1 - \hat{x}$.

Now keep repeating the same process to add more layers one at a time to get a multi-layered autoencoder, hence the name deep autoencoder as in

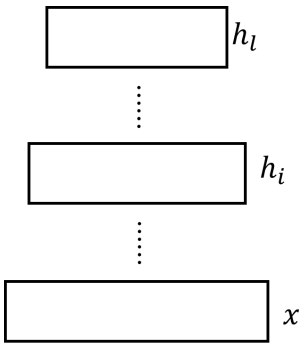

Figure 5: Keep adding new layers

Figure 5. Note that up to this, everything was done using only the input x. Since we have no need for the label y, what we have done so far can be dubbed **unsupervised learning**.

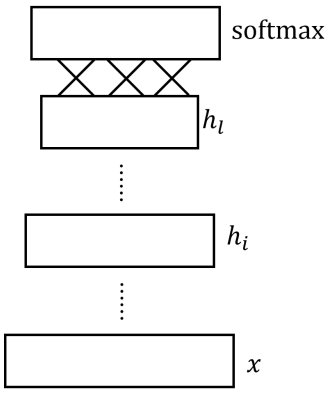

Figure 6: Classifier

After this deep autoencoder is constructed, we can now put it to use for, say, a classification task. Figure 6 shows a softmax layer put on top of the deep autoencoder given as in Figure 5. It is customary to make the top two layers of Figure 5 fully connected. Furthermore, we may add many layers onto the top of the deep autoencoder of Figure 5. If it is a classfication problem, the only requirement is that the topmost layer must be a softmax layer. To train the deep neural network as in Figure 6, we also use the backpropagation algorithm with cross entropy error. In fact, the training is

done in such a way that all the weights of the entire network is relearned. However, if the deep autoencoder as in Figure 5 is trained in such a way to act as a good encoder, the information loss should be small enough through this compression process so that reasonable variations in the input data do not result in adverse effects. Therefore, the weights of the deep autoencoder as in Figure 5 should be good initial weights for the classification problem at hand. This is one of the reasons why deep autoencoder works so effectively.

1.3 Denoising autoencoders

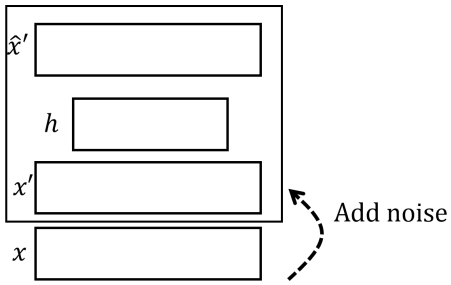

Figure 7: Denosing autoencoder

In order to avoid overfitting autoencoders, various regularization techniques are applied. One of them is intentionally adding noise, in which case the resulting autoencoder is called a denoising autoencoder. Figure 7 shows an example of a simple autoencoder to which noise is added. It works as follows. Instead of using the input x, one adds noise, typically Gaussian noise, and this corrupted input x' is used as an input for the simple autoencoder enclosed in the box in Figure 7. However, when it comes to decoding, its target is not the corrupt x' but the uncorrupted original input x. So the error is

$$\begin{aligned} E &= \frac{1}{2}\sum_i |\hat{x'}^{(i)} - x^{(i)}|^2 \\ &= \frac{1}{2}\sum_i \sum_k |\hat{x'}_k^{(i)} - x_k^{(i)}|^2, \end{aligned}$$

where $\hat{x'}^{(i)} = C(f(x'^{(i)}))$, i.e. the decoded value of $x'^{(i)}$ that is the corrupted $x^{(i)}$. One can also apply the same denoising method for each layer in the layerwise pre-training.

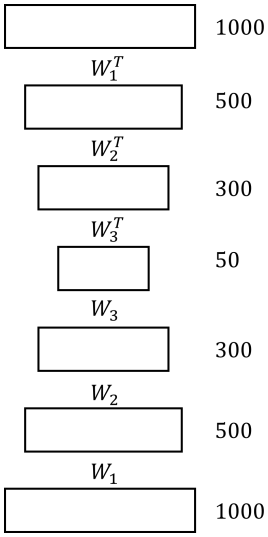

Figure 8: Stack them all at once

The training we have described above is done layer-by-layer, hence the laywerwise training. And we also call it pre-training if we want to emphasize the fact that the supervised learning involving the output (label) is the final goal and the construction of the autoencoder is only a preparatory step.

There is another way to train autoencoders. Figure 8 shows the construction of a deep autoencoder. In here, instead of building up layer-by-layer, we put up the whole structure at once and train the whole network (find good connection weights) by minimizing the L^2-error between the input x at the bottom layer and the output $\hat{x}$ at the top layer. This is a feasible problem because nowadays training deep neural networks can be routinely done.

Once the network in Figure 8 is constructed, we remove the top half to get the network in Figure 9. This again is the deep autoencoder.

1.4 More autoencoders

There are many variations of autoencoders. Quite popluar nowadays is the variational autoencoder. It is a generative model that works well for a dataset with continuous latent variables. Because of this reason, it is being widely used in problems like natural language processing. An interested reader is referred to the paper by Kingma and Welling [1].

Another interesting class of autoencoders is the so-called sparse autoen-

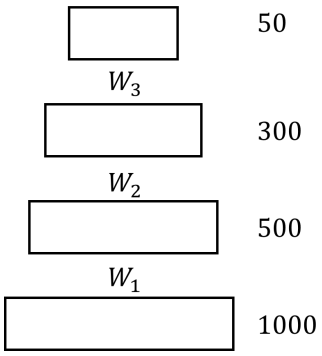

Figure 9: Remove top half

coders. Sparsity, though we did not have time to cover it adequately, is a very important concept in deep learning and machine learning in general. In our context, sparse autoencoder has the characteristic of getting only a few neurons (say, 5%) in any given computation in an encoding layer. This makes it easier for sparse autoencoders to extract features. To force sparsity, one introduces sparsity measure and incorporate its minimization in the training time. A fairly nice introduction to sparse autoencoder is given in [2] to which the reader is referred.

2 Why deep learning works

References

[1] Kingma,D., Welling, M., *Auto-Encoding Variational Bayes*, https://arxiv.org/pdf/1312.6114.pdf

[2] Ng, A., *Sparse autoencoder*, https://web.stanford.edu/class/cs294a/sparseAutoencoder.pdf