5

Pre-Trained Deep Neural Networks — A Hybrid

In this section, we present the most widely used hybrid deep architecture — the pre-trained deep neural network (DNN), and discuss the related techniques and building blocks including the RBM and DBN. We discuss the DNN example here in the category of hybrid deep networks before the examples in the category of deep networks for supervised learning (Section 6). This is partly due to the natural flow from the unsupervised learning models to the DNN as a hybrid model. The discriminative nature of artificial neural networks for supervised learning has been widely known, and thus would not be required for understanding the hybrid nature of the DNN that uses unsupervised pre-training to facilitate the subsequent discriminative fine tuning.

Part of the review in this chapter is based on recent publications in [68, 161, 412].

5.1 Restricted Boltzmann machines

An RBM is a special type of Markov random field that has one layer of (typically Bernoulli) stochastic hidden units and one layer of (typically Bernoulli or Gaussian) stochastic visible or observable units. RBMs can be represented as bipartite graphs, where all visible units are connected to all hidden units, and there are no visible–visible or hidden–hidden connections.

In an RBM, the joint distribution $p(\mathbf{v}, \mathbf{h}; \theta)$ over the visible units $\mathbf{v}$ and hidden units $\mathbf{h}$, given the model parameters θ, is defined in terms of an energy function $E(\mathbf{v}, \mathbf{h}; \theta)$ of

$$p(\mathbf{v}, \mathbf{h}; \theta) = \frac{\exp(-E(\mathbf{v}, \mathbf{h}; \theta))}{Z},$$

where $Z = \sum_{\mathbf{v}} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}; \theta))$ is a normalization factor or partition function, and the marginal probability that the model assigns to a visible vector $\mathbf{v}$ is

$$p(\mathbf{v}; \theta) = \frac{\sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}; \theta))}{Z}$$

For a Bernoulli (visible)-Bernoulli (hidden) RBM, the energy function is defined as

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_{i=1}^{I} \sum_{j=1}^{J} w_{ij} v_i h_j - \sum_{i=1}^{I} b_i v_i - \sum_{j=1}^{J} a_j h_j.$$

where w_{ij} represents the symmetric interaction term between visible unit v_i and hidden unit h_j, b_i and a_j the bias terms, and I and J are the numbers of visible and hidden units. The conditional probabilities can be efficiently calculated as

$$\begin{aligned} p(h_j = 1|\mathbf{v}; \theta) &= \sigma\left(\sum_{i=1}^{I} w_{ij} v_i + a_j\right), \\ p(v_i = 1|\mathbf{h}; \theta) &= \sigma\left(\sum_{j=1}^{J} w_{ij} h_j + b_i\right), \end{aligned}$$

where $\sigma(x) = 1/(1 + \exp(-x))$.

Similarly, for a Gaussian (visible)-Bernoulli (hidden) RBM, the energy is

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_{i=1}^{I} \sum_{j=1}^{J} w_{ij} v_i h_j - \frac{1}{2} \sum_{i=1}^{I} (v_i - b_i)^2 - \sum_{j=1}^{J} a_j h_j,$$

The corresponding conditional probabilities become

$$\begin{aligned} p(h_j = 1|\mathbf{v}; \theta) &= \sigma\left(\sum_{i=1}^{I} w_{ij} v_i + a_j\right), \\ p(v_i|\mathbf{h}; \theta) &= N\left(\sum_{j=1}^{J} w_{ij} h_j + b_i, 1\right), \end{aligned}$$

where v_i takes real values and follows a Gaussian distribution with mean $\sum_{j=1}^{J} w_{ij} h_j + b_i$ and variance one. Gaussian-Bernoulli RBMs can be used to convert real-valued stochastic variables to binary stochastic variables, which can then be further processed using the Bernoulli-Bernoulli RBMs.

The above discussion used two of the most common conditional distributions for the visible data in the RBM — Gaussian (for continuous-valued data) and binomial (for binary data). More general types of distributions in the RBM can also be used. See [386] for the use of general exponential-family distributions for this purpose.

Taking the gradient of the log likelihood $\log p(\mathbf{v}; \theta)$ we can derive the update rule for the RBM weights as:

$$\Delta w_{ij} = E_{\text{data}}(v_i h_j) - E_{\text{model}}(v_i h_j),$$

where $E_{\text{data}}(v_i h_j)$ is the expectation observed in the training set (with h_j sampled given v_i according to the model), and $E_{\text{model}}(v_i h_j)$ is that same expectation under the distribution defined by the model. Unfortunately, $E_{\text{model}}(v_i h_j)$ is intractable to compute. The contrastive divergence (CD) approximation to the gradient was the first efficient method proposed to approximate this expected value, where $E_{\text{model}}(v_i h_j)$ is replaced by running the Gibbs sampler initialized at the data for one or more steps. The steps in approximating $E_{\text{model}}(v_i h_j)$ is summarized as follows:

- Initialize $\mathbf{v_0}$ at data
- Sample $\mathbf{h_0} \sim \mathbf{p}(\mathbf{h}|\mathbf{v_0})$
- Sample $\mathbf{v_1} \sim \mathbf{p}(\mathbf{v}|\mathbf{h_0})$
- Sample $\mathbf{h_1} \sim \mathbf{p}(\mathbf{h}|\mathbf{v_1})$

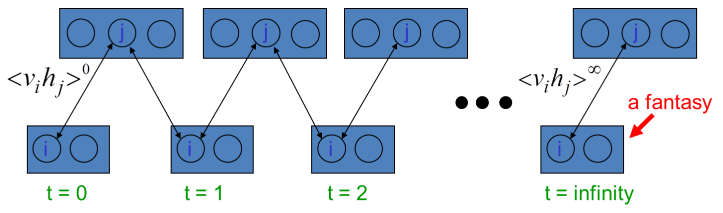

Figure 5.1: A pictorial view of sampling from a RBM during RBM learning (courtesy of Geoff Hinton).

Here, $(\mathbf{v_1}, \mathbf{h_1})$ is a sample from the model, as a very rough estimate of $E_{\text{model}}(v_i h_j)$. The use of $(\mathbf{v_1}, \mathbf{h_1})$ to approximate $E_{\text{model}}(v_i h_j)$ gives rise to the algorithm of CD-1. The sampling process can be pictorially depicted in Figure 5.1.

Note that CD-k generalizes this to more steps of the Markov chain. There are other techniques for estimating the log-likelihood gradient of RBMs, in particular the stochastic maximum likelihood or persistent contrastive divergence (PCD) [363, 406]. Both work better than CD when using the RBM as a generative model.

Careful training of RBMs is essential to the success of applying RBM and related deep learning techniques to solve practical problems. See Technical Report [159] for a very useful practical guide for training RBMs.

The RBM discussed above is both a generative and an unsupervised model, which characterizes the input data distribution using hidden variables and there is no label information involved. However, when the label information is available, it can be used together with the data to form the concatenated "data" set. Then the same CD learning can be applied to optimize the approximate "generative" objective function related to data likelihood. Further, and more interestingly, a "discriminative" objective function can be defined in terms of conditional likelihood of labels. This discriminative RBM can be used to "fine tune" RBM for classification tasks [203].

Ranzato et al. [297, 295] proposed an unsupervised learning algorithm called sparse encoding symmetric machine (SESM), which is quite similar to RBM. They both have a symmetric encoder and decoder, and a logistic nonlinearity on the top of the encoder. The main difference is that whereas the RBM is trained using (very approximate) maximum likelihood, SESM is trained by simply minimizing the average energy plus an additional code sparsity term. SESM relies on the sparsity term to prevent flat energy surfaces, while RBM relies on an explicit contrastive term in the loss, an approximation of the log partition function. Another difference is in the coding strategy in that the code units are "noisy" and binary in the RBM, while they are quasi-binary and sparse in SESM. The use of SESM in pre-training DNNs for speech recognition can be found in [284].

5.2 Unsupervised layer-wise pre-training

Here we describe how to stack up RBMs just described to form a DBN as the basis for DNN's pre-training. Before delving into details, we first note that this procedure, proposed by Hinton and Salakhutdinov [163] is a more general technique of unsupervised layer-wise pretraining. That is, not only RBMs can be stacked to form deep generative (or discriminative) networks, but other types of networks can also do the same, such as autoencoder variants as proposed by Bengio et al. [28].

Stacking a number of the RBMs learned layer by layer from bottom up gives rise to a DBN, an example of which is shown in Figure 5.2. The stacking procedure is as follows. After learning a Gaussian-Bernoulli RBM (for applications with continuous features such as speech) or Bernoulli-Bernoulli RBM (for applications with nominal or binary features such as black-white image or coded text), we treat the activation probabilities of its hidden units as the data for training the Bernoulli-Bernoulli RBM one layer up. The activation probabilities of the second-layer Bernoulli-Bernoulli RBM are then used as the visible data input for the third-layer Bernoulli-Bernoulli RBM, and so on. Some theoretical justification of this efficient layer-by-layer greedy learning strategy is given in [163], where it is shown that the *stacking* procedure above improves a variational lower bound on the likelihood of the training data under the composite model. That is, the greedy procedure

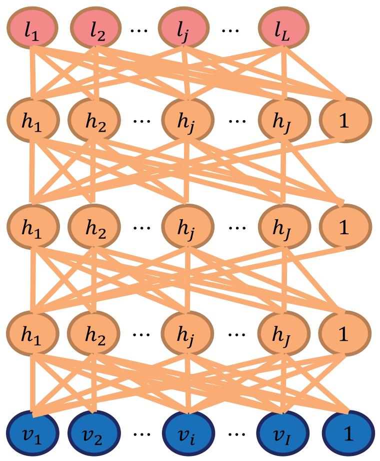

Figure 5.2: An illustration of the DBN-DNN architecture.

above achieves approximate maximum likelihood learning. Note that this learning procedure is unsupervised and requires no class label.

When applied to classification tasks, the generative pre-training can be followed by or combined with other, typically discriminative, learning procedures that fine-tune all of the weights jointly to improve the performance of the network. This discriminative fine-tuning is performed by adding a final layer of variables that represent the desired outputs or labels provided in the training data. Then, the back-propagation algorithm can be used to adjust or fine-tune the network weights in the same way as for the standard feed-forward neural network. What goes to the top, label layer of this DNN depends on the application. For speech recognition applications, the top layer, denoted by "$l_1, l_2, \ldots, l_j, \ldots, l_L$," in Figure 5.2, can represent either syllables, phones, sub-phones, phone states, or other speech units used in the HMM-based speech recognition system.

The generative pre-training described above has produced better phone and speech recognition results than random initialization on a wide variety of tasks, which will be surveyed in Section 7. Further research has also shown the effectiveness of other pre-training strategies. As an example, greedy layer-by-layer training may be carried out with an additional discriminative term to the generative cost function at each level. And without generative pre-training, purely discriminative training of DNNs from random initial weights using the traditional stochastic gradient decent method has been shown to work very well when the scales of the initial weights are set carefully and the mini-batch sizes, which trade off noisy gradients with convergence speed, used in stochastic gradient decent are adapted prudently (e.g., with an increasing size over training epochs). Also, randomization order in creating mini-batches needs to be judiciously determined. Importantly, it was found effective to learn a DNN by starting with a shallow neural network with a single hidden layer. Once this has been trained discriminatively (using early stops to avoid overfitting), a second hidden layer is inserted between the first hidden layer and the labeled softmax output units and the expanded deeper network is again trained discriminatively. This can be continued until the desired number of hidden layers is reached, after which a full backpropagation "fine tuning" is applied. This discriminative "pre-training" procedure is found to work well in practice [324, 419], especially with a reasonably large amount of training data. When the amount of training data is increased even more, then some carefully designed random initialization methods can work well also without using the above pre-training schemes.

In any case, pre-training based on the use of RBMs to stack up in forming the DBN has been found to work well in most cases, regardless of a large or small amount of training data. It is useful to point out that there are other ways to perform pre-training in addition to the use of RBMs and DBNs. For example, denoising autoencoders have now been shown to be consistent estimators of the data generating distribution [30]. Like RBMs, they are also shown to be generative models from which one can sample. Unlike RBMs, however, an unbiased estimator of the gradient of the training objective function can be obtained by the denoising autoencoders, avoiding the need for MCMC or variational approximations in the inner loop of training. Therefore, the greedy layer-wise pre-training may be performed as effectively by stacking the denoising autoencoders as by stacking the RBMs each as a single-layer learner.

Further, a general framework for layer-wise pre-training can be found in many deep learning papers; e.g., Section 2 of [21]. This includes, as a special case, the use of RBMs as the single-layer building block as discussed in this section. The more general framework can cover the RBM/DBN as well as any other unsupervised feature extractor. It can also cover the case of unsupervised pre-training of the representation only followed by a separate stage of learning a classifier on top of the unsupervised, pre-trained features [215, 216, 217].

5.3 Interfacing DNNs with HMMs

The pre-trained DNN as a prominent example of the hybrid deep networks discussed so far in this chapter is a static classifier with input vectors having a fixed dimensionality. However, many practical pattern recognition and information processing problems, including speech recognition, machine translation, natural language understanding, video processing and bio-information processing, require sequence recognition. In sequence recognition, sometimes called classification with structured input/output, the dimensionality of both inputs and outputs are variable.

The HMM, based on dynamic programing operations, is a convenient tool to help port the strength of a static classifier to handle dynamic or sequential patterns. Thus, it is natural to combine feed-forward neural networks and HMMs to bridge the gap between the static and sequence pattern recognition, as was done in the early days of neural networks for speech recognition [17, 25, 42]. A popular architecture to fulfill this role with the use of the DNN is shown in Figure 5.3. This architecture has been successfully used in speech recognition experiments as reported in [67, 68].

It is important to note that the unique elasticity of temporal dynamics of speech as elaborated in [45, 73, 76, 83] would require temporally

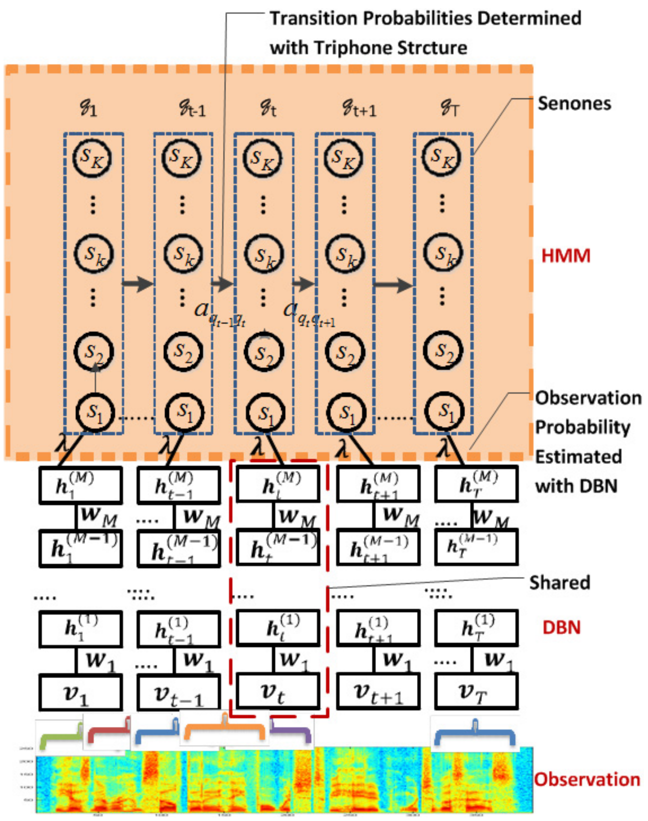
Transition Probabilities Determined with Triphone Strcture
Senones
HMM
Observation Probability Estimated with DBN
Shared
DBN
Observation

Figure 5.3: Interface between DBN/DNN and HMM to form a DNN–HMM. This architecture, developed at Microsoft, has been successfully used in speech recognition experiments reported in [67, 68]. [after [67, 68], @IEEE].

correlated models more powerful than HMMs for the ultimate success of speech recognition. Integrating such dynamic models that have realistic co-articulatory properties with the DNN and possibly other deep learning models to form the coherent dynamic deep architecture is a challenging new research direction.