Speech Recognition via Deep Neural Networks

Maurice Diesendruck, Bowei Yan
December 4, 2014

1 INTRODUCTION

Automatic speech recognition is the science of building computer models to convert audio recordings of speech into their corresponding text. For much of the past 30 years, researchers have used Hidden Markov Models (HMM) to describe the sequential underlying states of speech, and Gaussian Mixture Models (GMM) to describe how each state creates the raw audio signal. Researchers have since introduced an alternative method to the GMM, neuroscience-inspired algorithms called Deep Neural Networks (DNN), which have outperformed the state-of-the-art in many audio classification tasks.

2 HIDDEN MARKOV MODEL

The Hidden Markov Model is a graphical model technique used to describe a series of observations, where each observation is the outcome of an underlying hidden state. The model consists of three parts: (1) a set of initial probabilities for each hidden state, (2) a matrix of transition probabilities among hidden states, and (3) an emission or "output" function that models how each hidden state produces each observed variable. In Figure 2.1, hidden states are X's, observed variables are E's, transition probabilities are for movements between X's, and the emission function is from each hidden state X to its corresponding observed variable E.

Consider the following example: A researcher has a sequence of DNA ("observed variable"), and knows that a gene's chemical environment ("hidden state") influences which DNA bases are more likely to occur. Using HMMs, a researcher can answer three questions: (1) Given a model, how likely is a given sequence of DNA? (2) What is the most likely sequence of chemical environments, given a sequence of DNA? and (3) Which model parameters would best fit the observed DNA sequences and chemical environments? These three questions have been answered using the Forward, Viterbi, and Baum-Welch algorithms, respectively.

Figure 2.1: Graph representation for hidden markov model

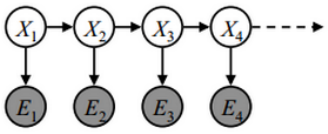
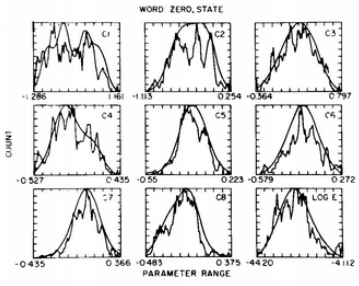


Figure 2.2: Gaussian Mixture Model



2.1 GAUSSIAN MIXTURE MODEL

GMMs are a popular choice for the emission function in the HMM because of their flexibility, and their ability to represent complex relationships between states and the observed variables. The model is easily trained using the Expectation Maximization algorithm, but needs large amounts of data to prevent overfitting and is statistically inefficient for modeling non-linear manifold data. Figure 2.2 shows how outputs for a single state can be multimodal, and lend themselves to Gaussian mixture modeling.

3 ACOUSTIC REPRESENTATION

In most audio modeling applications, raw audio is converted into lower-dimensional features that communicate the spectral characteristics of audio. Commonly used features include Mel Frequency Spectral Coefficients (MFCC), Perceptual Linear Predictive coefficients (PLP), and first and second temporal derivatives, all of which throw away a large part of the

information in the signal, on the belief that these features contain most of what is needed for recognition. In fact, years of research in the field of Knowledge Engineering was dedicated to designing features that would yield better performance. Newer DNN algorithms utilize larger windows of the audio input, and aim to learn higher-order features, as opposed to explicitly defining features. The eventual goal of any model here is, given audio input, to get a better model of posterior probabilities over hidden states. In a typical application, audio signals are split into frames of 25ms, and encoded as vectors of spectral features.

4 Deep Neural Network

The overall aim of the DNN is to first learn the higher-order structure of the inputs, and then attempt classification using those more complex characteristics (learned features). DNNs achieve this by considering several input vectors at once and generatively pre-training layers of a feed forward neural network. A feed-forward neural network, also called a "multilayer perceptron", is a scaled up version of a fully-connected bi-partite graph between input units and hidden units, where a weighted sum of inputs is fed to an activation function that determines the value of each output.

Recent speech recognition applications of DNNs have roughly implemented the following architecture: several frames of spectral vectors as inputs, hidden layers of 1024 nodes each, fewer than 10 total layers, and a softmax output layer for multiclass classification. In Figure 4.1, hidden units are called "Neurons", and the activation function is derived from the concept of a neuron firing once inputs surpass a certain threshold. In the case of probabilistic inference, however, it is critical to use a smooth and differentiable activation function (e.g. sigmoid, hyperbolic tangent). Each hidden unit's value is determined by the output of the activation function, and the value is often between 0 and 1, similar to a probability for that unit.

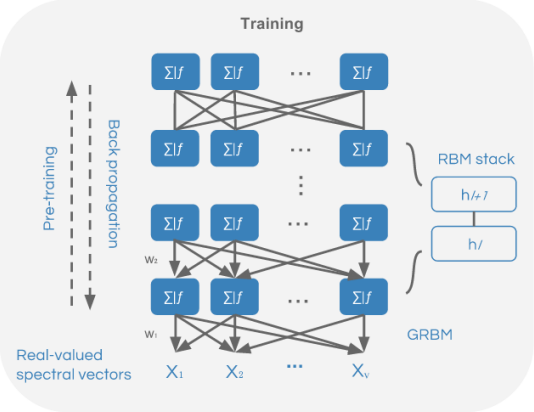
Learning can be difficult for densely-connected, directed belief networks that have many hidden layers, because weights are coupled together in the marginal distribution of the visible nodes. We can, however, use an undirected graphical structure, which can be effectively trained as a building block for the DNN.

4.1 Architecture

This section will introduce the entire architecture of the speech recognition system with deep belief networks. Figure 4.1 shows a multilayer generative model in which the top two layers interact via undirected connections and all other connections are directed. At the top, the undirected connections are equivalent to having infinitely many higher layers with tied weights. There are no intra-layer connections, and for simplicity, all layers have the same number of units.

Five steps summarize this process. First, the raw signals are translated into features like MFCCs, which are set as the inputs to the system.

Figure 4.1: Graph Structure for Deep Neural Network



Second, a Gaussian-Bernoulli Restricted Boltzmann Machine (GRBM) is trained to produce binary states for the first hidden layer.

Third, the binary output of the previous step is used as an input for a Restricted Boltzmann Machine (RBM) that learns to model the features of the first hidden layer. This process can be repeated many times to produce many layers of nonlinear feature detectors.

Such a stack of RBMs can be combined into a single, multilayer generative model called a deep belief network(DBN), where the network's top two layers are undirected, and its lower layers are directed downward.

Fourth, after learning a DBN via steps 2 and 3, generative weights are used in the reverse direction to initialize each feature-detecting layer of the deterministic feed-forward DNN.

Finally, a final softmax layer is added on top of the network, and the entire DNN is trained discriminatively.

5 Learning Procedure for Deep Neural Network

There are two main types of generative neural network. Given some binary stochastic neurons, some visible and some hidden, if connected in an acyclic graph, they produce a Sigmoid belief network; if connected using symmetric connections, they produce a Boltzmann machine. This section introduces both of these network structures and derives the surprising equivalence between Restricted Boltzmann Machines and tied-weight infinite belief networks.

5.1 Boltzmann Machine

A Boltzmann machine is a graph with two-valued variables linked by symmetrical connections. The probability distribution of a Boltzmann machine is defined through the energy function. Setting restrictions on the connectivity makes inference easier. First, there is only one layer of hidden units. Second, there is no connection within hidden units or within

visible units. This bipartite graph is called a Restricted Boltzmann Machine (RBM). The energy function for the RBM is thereby simplified to

$$\begin{aligned} E(v,h) &= -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i h_j w_{ij} \\ P(v,h) &= exp(-E(v,h))/Z \end{aligned} \tag{5.1}$$

where Z is the normalizing factor. The probability over visible units is

$$P(V=v) = \sum_h P(S=<h,v>)$$

Based on 5.1 the conditional probability of each node is written as

$$\begin{aligned} P(h_j = 1|v) &= \sigma(b_j + \sum_i v_i w_{ij}) \\ P(v_i = 1|h) &= \sigma(a_i + \sum_j h_j w_{ij}) \end{aligned} \tag{5.2}$$

where $\sigma(x)$ is the sigmoid function, defined as

$$\sigma(x) = \frac{1}{1+exp(-x)}$$

The learning problem for RBM is to adjust the weights, so that the distribution over visible variables matches as closely as possible to the distribution of some real-world attributes. Define the log-likelihood function given the training set as

$$L = log(\prod_{v \in \mathcal{T}} P(V=v) = \sum_{v \in \mathcal{T}} logP(V=v)$$

The maximum-likelihood estimation for the model parameters is calculated by taking the partial derivative of this function with respect to each w_{ij},

$$\begin{aligned} \frac{\partial L}{\partial w_{ij}} &= \sum_{v_0 \in \mathcal{T}} \frac{\partial}{\partial w_{ij}} log \sum_h P(S=<v_0,h>) \\ &= \sum_{v_0 \in \mathcal{T}} (\sum_s P(S=s|V=v_0) s_i s_j - \sum_s P(S=s) s_i s_j) \\ &=< v_i h_j >_{data} - < v_i h_j >_{model} \end{aligned} \tag{5.3}$$

where the angle brackets are used to denote expectations under the distribution specified by the subscript that follows. This provides a simple learning rule for performing stochastic steepest ascent in the log probability of the training data,

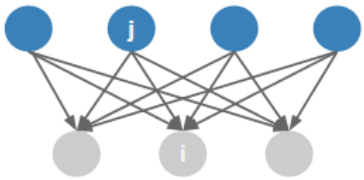
$$\Delta w_{ij} = \epsilon(< v_i h_j >_{data} - < v_i h_j >_{model}) \tag{5.4}$$

where ϵ is the learning rate. Then, 5.2 is used to generate an unbiased sample with Gibbs sampling, and with that sample, the gradient descent algorithm can be run to get the best estimation of w. It may, however, take the Markov Chain Monte Carlo procedure infinite many steps to converge to the true model posterior, which is too time-consuming. Instead, the gradient in (5.4) is approximated using a fast algorithm called Contrastive Divergence (CD).

The main idea for CD is to replace $< v^T h >_\infty$ with $< v^T h >_1$, which is the first time reconstruction by Gibbs sampling; then update the weights by

$$\Delta w_{ij} = \epsilon(< v_i h_j >_0 - < v_i h_j >_1)$$

Figure 5.1: Belief Network



where ϵ is the learning rate.

This gives an approximation for the gradient of the log probability of the training data, and works well in pre-training feature detectors. CD runs only one full step of Gibbs sampling after the initial update of the hidden states.

5.2 Belief Network

A Belief Network (BN) is a directed graphical model, which is also called Bayesian network, see Figure 5.1. In this type of network, the joint distribution of all nodes in a graph can be written as the product of the conditional probabilitiy of each node given its ancestors.

In this case, networks are composed of stochastic binary variables with weighted connections. The probability of turning on each unit given all the other nodes is defined by the weighted input from other units and a bias for this particular unit.

$$p(s_i = 1|s_{-i}) = \frac{1}{1+exp(-b_i - \sum_j s_j w_{ij})}$$

With an unbiased sample from the posterior distribution over hidden states given the observed ones, inference over the hidden states is easy. The existence of v-structures, however, prevents hidden states from being independent, and because the posterior contains interaction between different weights, inference is again very hard.

To solve the inference problem, hidden variables are added with a complementary prior that has exactly the opposite correlation to those in the likelihood terms, making the posterior probability factorial. After sampling from the true posterior, derivatives of the log probability of the data can be computed. For instance, the derivative for a generative weight from unit j in layer H_0 to unit i in layer V_0 can be computed,

$$\frac{\partial log p(v^0)}{\partial w_{ij}^{00}} =< h_j^0 (v_i^0 - \hat{v_i^0}) >$$

where the angle brackets denote an average over the sample states and $\hat{v_i^0}$ is the probability that unit i would be turned on, if the visible vector was reconstructed from the sampled hidden states.

When compute the derivative for each pair of hidden and visible layers and stacking them together, the full derivative for a generative weights is obtained by summing the derivatives of each of them together.

$$\frac{\partial log p(v^0)}{\partial w_{ij}^{00}} =< v_i^0 h_j^0 > - < v_i^\infty h_j^\infty >$$

The cross terms cancel out, leaving the same form as the Boltzmann machine learning rule of 5.3.

5.3 Stacking up the RBMs

Why does the greedy learning algorithm work on a stack of RBMs? The objective is to find the model that maximizes p(v).

$$p(v) = \sum_h p(h)p(v|h)$$

In the right hand side of this equation, the first term is the prior of hidden states, and the second term is the likelihood function defined by the weight matrix W. By leaving the likelihood term alone and improving the prior, the probability of visible nodes improves. In the system, p(h) is the aggregated posterior distribution of all higher layer RBMs. After training the first layer of hidden states and getting the weight W_0, the weight is frozen, and a sample is generated by W_0^T from the data. This value is then used as the input to the second layer RBM.

One problem here is that the weight matrix in second layer would differ from that of the first layer. It is proved, though, that each time a new RBM is added to the stack, the variational bound on the log probability of the training data is better than the previous bound.

5.4 Model Real-valued data with GRBM

Thus far, the focus has been on binary-valued data. In constrast, the raw input data of signal is always continuous, as is typical with MFCCs. In order to suppress noise in the learning steps, the model requires a real-valued probability instead of a binary one, in modeling the first layer of the hidden states. This is generally achieved using linear variables with Gaussian noise. The RBM energy function can be modified to accommodate such variables, giving a Gaussian-Bernoulli RBM(GRBM),

$$E(v,h) = \sum_i \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_j b_j h_j - \sum_{i,j} \frac{v_i}{\sigma_i} h_j w_{ij}$$

where σ_i is the standard deviation of the Gaussian noise for visible units. In practice, this data is first normalized so that standard deviations are set to 1.

As was the case for binary RBMs, the conditional probability distribution for CD learning is

$$\begin{aligned} P(h_j|v) &= logistic(b_j + \sum_i \frac{v_i}{\sigma_i} w_{ij}) \\ P(v_i|h) &= N(a_i + \sigma_i \sum_j h_j w_{ij}, \sigma_i^2) \end{aligned} \tag{5.5}$$

Then, as for the binary RBM, weights are learned for each pair of hidden-visible nodes.

5.5 Fine-tuning with back propagation

Back propagation is a general algorithm for learning labeled data. However it is not a global optimization algorithm, so if starting from some random initial points, it will probably get stuck in poor local minima. Besides, the learning can be very slow if the structure has many layers of hidden variables. Pre-training gives a good initial value for the weights in the network, and backpropagation only needs to run once to do a local search. This helps overcome the limitation of the back propagation and get much better performances in experiments.

5.6 Inference

The objective for speech recognition is to find out the label of the data, which can be represented by maximizing a softmax function.

$$\begin{aligned} P(Label_i|x,\theta) &= SoftMax_i(Wx+b) \\ &= \frac{e^{W_i x + b_i}}{\sum_j e^{W_j x + b_j}} \\ Label_{pred} &= argmax_i P(Label_i|x,\theta) \end{aligned} \tag{5.6}$$

In the softmax layer, each binary node represents one possible category of the data to classify. And after the inference step a probability of turning on for each of those units is given, and the data is labeled with the one with the highest probability.

6 Conclusion

Despite the emergence of Deep Neural Networks in the 1980s, the algorithm's recent popularity and performance is mainly due to improved hardware for computation, and improved technique in implementing and tuning the overall system. Experiments demonstrated that DNNs now classify phones and words with an error rate of 16%, down from the previous state-of-the-art performance of 20%. While impressive, the DNN still requires significant tuning to work optimally, and the training procedure is still relatively hard to parallelize. Future work will likely involve combinations of techniques, variations on the initial inputs, and variations on algorithms to further improve speed and performance.

References

[1] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[2] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[3] Radford M Neal. Connectionist learning of belief networks. *Artificial intelligence*, 56(1):71–113, 1992.