Acoustic Modeling using Deep Belief Networks

1 Citation

Mohamed, Abdel-rahman, George E. Dahl, and Geoffrey Hinton. "Acoustic modeling using deep belief networks." IEEE Trans. Audio, Speech Language Processing 20.1 (2012): 14-22.

`http://www.cs.toronto.edu/~asamir/papers/speechDBN_jrnl.pdf`

2 Abstract

By pretraining a neural network in an unsupervised manner on windows of spectral feature vectors and then doing supervised training on the TIMIT dataset, we are able to outperform a traditional Gaussian Mixture Model (GMM) + Hidden Markov Model (HMM) system.

3 Introduction

Ttraditional Automatic Speech Recognition (ASR) systems compute a spectral representation, usually Mel Frequency Cepstral Coefficients (MFCC), on 25 ms windows, model these representations with a GMM, and apply an HMM on the sequence. Neural Networks can help because they combine features in richer ways, are more sample efficient than GMM, and they can model hierarchical structure.

We use a combination of unsupervised (i.e. train as generative model) and supervised training. We do this because we think the key to speech recognition is to model the hierarchical concepts of speech, which can be done with a generative model.

4 Learning a Multilayer Generative Model

Let's consider the "undirected" view first and then consider the "directed" view. For the former, we create a Restricted Boltzmann Machine (RBM), which is a weighted bipartite graph that goes from visible Gaussian units to hidden Bernoulli units. Let's first assume the visible units are also Bernoulli (i.e. they take on binary values). We define energy and probability as follows ($\theta = (\mathbf{w}, \mathbf{b}, \mathbf{a})$):

$$E(\mathbf{v}, \mathbf{h}|\theta) = -\sum_{i=1}^{\mathcal{V}}\sum_{j=1}^{\mathcal{H}} w_{ij}v_i h_j - \sum_{i=1}^{\mathcal{V}} b_i v_i - \sum_{j=1}^{\mathcal{H}} a_j h_j$$

$$p(\mathbf{v}|\theta) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{u}}\sum_{\mathbf{h}} e^{-E(\mathbf{u},\mathbf{h})}}$$

Marginalize to get: $p(h_j = 1|\mathbf{v}, \theta) = \sigma(a_j + \sum_{i=1}^{\mathcal{V}} w_{ij}v_i)$ and $p(v_i = 1|\mathbf{h}, \theta) = \sigma(b_i + \sum_{j=1}^{\mathcal{H}} w_{ij}h_j)$

To train the weights, we use "contrastive divergence": $\triangle w_{ij} \propto [v_i, h_j]_{data} - [v_i h_j]_{reconstruction}$. The $[\cdot]$ is the frequency (in a minibatch) that visible unit i and hidden unit j are both 1 (i.e. expected value). The

first term comes from setting visible units using training data and inferring the hidden units with our marginal probability. The second term comes from further inferring the visible units from the inferred hidden units.

Coming back to our Gaussian-Bernoulli RBM, the energy is:

$$E(\mathbf{v}, \mathbf{h}|\theta) = -\sum_{i=1}^{\mathcal{V}}\sum_{j=1}^{\mathcal{H}} w_{ij} v_i h_j - \sum_{i=1}^{\mathcal{V}} \frac{(v_i - b_i)^2}{2} - \sum_{j=1}^{\mathcal{H}} a_j h_j$$

and we have $p(v_i|\mathbf{h}, \theta) = \mathcal{N}(b_i + \sum_{j=1}^{\mathcal{H}} w_{ij} h_j, 1)$. Hidden units can be visible units for a new RBM layer.

Now let's consider the directed view where we have a "sigmoid belief net" starting at a visible layer $\mathbf{v} = \mathbf{h}^{(0)}$ ending at layer L. We can use the bias terms of the deepest layer as the log odds and sample the values for its units. Then we can recursively compute the other units:

$$p(h_i^{(k-1)}|\mathbf{h}^{(k)}, \mathbf{W}^{(k)}) = \sigma(b_i^{(k-1)} + \sum_j w_{ij}^{(j)} h_j^{(k)})$$

Unfortunately, training such a model with gradient descent is hard because the posterior distribution of the deeper layers given the visible layer is expensive to compute, even with Markov Chain Monte Carlo sampling. So, we can simplify by using tied weights. That is, we assume our net is infinitely deep and that the weights alternate between $\mathbf{W}$ and $\mathbf{W}^T$. Then, the posterior for the first hidden layer given the visible units is: $p(h_j^{(1)} = 1|\mathbf{v}, \mathbf{W}) = \sigma(b_j^{(1)} + \sum_i w_{ij} v_i)$. It turns out that the net ends up being a long Markov chain that converges to a stationary distribution, so we can only consider two layers when doing a gradient ascent to set the matrix $\mathbf{W}$ to maximize the log likelihood. Our contrastive divergence rule is: $\triangle w_{ij} \propto [v_i h_j^{(1)}] - [h_i^{(2)} h_j^{(3)}]$. The tied weight matrix is restrictive, so after we learn it, we can freeze it for the first layer and then start learning new tied weights. Iteratively, we can then produce K layers of learned weights. We can chop off the rest of the weights and attach a softmax to get a feedforward neural net.

$$p(l|\mathbf{h}^K) = \frac{\exp(b_l + \sum_i h_i^{(K)} w_{il})}{\sum_m \exp(b_m + \sum_i h_i^{(K)} w_{im})}$$

The softmax weights are learned by training on the supervised data. To handle real values in the visible layer, we replace the binary units with linear units with Gaussian noise of variance 1. The mean of this noise given $\mathbf{h}^{(1)}$ for visible unit i is: $\mu_i = b_i^{(0)} + \sum_j w_{ij}^{(1)} h_j^{(1)}$.

5 Using Deep Belief Nets in Phone Recognition

Consider a window of n frames. Feed the spectral representation over these frames as our visible units. Use the unsupervised techniques above to build a feedforward net. Then, freeze the net, add a softmax layer, and train the softmax layer to predict the label for the central frame. At test time, to decode a sequence, we can apply Viterbi decoding.

6 Experimental Setup

We use the TIMIT corpus. Our window is 25 ms wide and slides 10 ms at a time. Our features are 12th-order MFCCs and their first and second temporal derivatives. We normalize each of these to have zero-mean and unit variance across the training set. We have 183 classes (3 states for each of the 61 phones). The 61 phones are then mapped to to 39 classes. We infer a bigram language model over phones. We train on an NVIDIA Tesla.

7 Experiments

Unsupervised pretraining is done with SGD (learning rate 0.002 for Gaussian-binary RBM and 0.02 for binary-binary RBM) on 128 example minibatch for 225 epochs for Gaussian-binary and 75 for binary-binary. Supervised training uses a learning rate of 0.1 that gets halved whenever validation error rises. We use momentum of 0.9 and weight decay of 0.0002 for all the above cases. We tried between 1 and 8 hidden layers (more hidden layers yields better performance). Using 2048 or 3072 hidden units helped. Pretraining was essential with more hidden layers (otherwise they give no benefit). We are able to get 20.7% error rate.

Using a bottleneck layer might help. Using something that is less "processed" than MFCC might also help. Using 11, 17, or 27 frames gives best performance (recall that the difference between successive frames is 10ms). We beat the state of the art (Triphone HMMs trained with BMMI).

8 Conclusion

By unsupervised pretraining a deep belief net and then fine-tuning the softmax layer in a supervised manner, all done on a MFCCs representation of window of audio, we're able to get a great deep network for classifying the phone in a window. Using Viterbi allows us to then label the entire sequence. We get state of the art speech recognition on the TIMIT dataset.