Deep Neural Networks for Acoustic Modeling in Speech Recognition

1 Citation

Hinton, Geoffrey, et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." IEEE Signal processing magazine 29.6 (2012): 82-97.

`http://www.cs.toronto.edu/~asamir/papers/SPM_DNN_12.pdf`

2 Abstract

Traditional speech recognition uses Hidden Markov Models (HMM), where the posterior probabilities over HMM states are modeled with a Gaussian Mixture Model (GMM). Our recent work replaces the GMM with a neural network.

3 Introduction

Acoustic signals are represented by Mel-frequency cepstral coefficients (MFCC). GMMs are statistically inefficient (e.g. imagine modeling points on the surface on a sphere - that would take a LOT of GMMs). Previous work would use neural nets to provide some features to the GMM, but we replace it altogether.

Our models output a probability distribution over triphone states. We actually tie triphone states to reduce the dimensionality of the output. To train the neural net, we first use unsupervised generative training to train one layer at a time. Finally, we add a fully-connected layer and softmax layer and fine-tune the whole network. The ground truth is generated by the GMM-HMM.

4 Training Deep Neural Networks

We feed our input vector through many hidden layers. and produce a vector $\mathbf{h}$. We then apply a fully-connected layer: $x_j = b_j + \sum_i h_i w_{ij}$, and a softmax: $p_j = \frac{\exp(x_j)}{\sum_k \exp(x_k)}$. Suppose that our ground truth target is one-hot encoded vector $\mathbf{d}$, then the cross-entropy loss is: $C = -\sum_j d_j \log(p_j)$. We then minimize this using stochastic gradient descent with momentum: $\triangle w_{ij}(t) = \alpha \triangle w_{ij}(t-1) - \epsilon \frac{\partial C}{\partial w_{ij}(t)}$. We regularize with L2 regularization and early stopping. We initialize weights from a zero-mean Gaussian. Initializing the entire network this way makes it hard to train, so we do unsupervised pretraining. This also makes the network generalize better to other tasks because it learns more about the structure of the data. So, how do we get the hidden vector $\mathbf{h}$?

We build our unsupervised network one layer at a time. You can use a directed model, but this is problemeatic because of the "explaining away" phenomenon, so we use a Restricted Boltzmann Machine (RBM) instead. We have a bipartite graph between visible and hidden units.. We define a probability based on energy function (E),

$$p(\mathbf{v}, \mathbf{h}; \mathbf{W}) = \frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h}; \mathbf{W})}$$

where $Z = \sum_{\mathbf{v}', \mathbf{h}'} e^{-E(\mathbf{v}', \mathbf{h}'; \mathbf{W})}$ is the partition function. Assuming all units are Bernoulli (i.e. binary), we have energy

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in visible} a_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

where $\mathbf{a}$ and $\mathbf{b}$ are bias vectors. Our likelihood function is thus: $p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}$. The gradient of a minibatch is thus

$$\frac{1}{N} \sum_{n=1}^{n} \frac{\partial \log(p(\mathbf{v^n}))}{\partial w_{ij}} = E_{data}[v_i h_j] - E_{model}[v_i h_j]$$

This yields the update rule: $\triangle w_{ij} = \epsilon(E_{data}[v_i h_j] - E_{model}[v_i h_j])$. For the first term, we use the visible units from the data and use the marginal distribution to get the hidden units. For the second, in order to approximate it efficiently (i.e. we use $E_{recon}[v_i h_j]$ instead of $E_{model}[v_i h_j]$), we use the marginal distribution to get the hidden units, use another marginal distribution to get the visible, and then get the hidden ones again. This is called contrastive divergence. The marginal distributions are:

$$p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_i v_i w_{ij}) \text{ and } p(v_i = 1|\mathbf{h}) = \sigma(a_i + \sum_j h_j w_{ij})$$

For Gaussian (linear) visible units, we have

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

and our two marginal distributions are:

$$p(h_j|\mathbf{v}) = \sigma(b_j + \sum_i \frac{v_i}{\sigma_i} w_{ij}) \text{ and } p(v_i|\mathbf{h}) = \mathcal{N}(a_i + \sigma_i \sum_j h_j w_{ij}, \sigma_i^2)$$

If we ensure our visible units have zero-mean and unit-variance, we can simply set $\sigma_i = 1$. Once you train an RBM, you can feed its hidden units as input to a new RBM. With this approach, you can build a deep net layer by layer - this is called a Deep Belief Net. We can then add the fully-connected layer and softmax layer and fine-tune on supervised data to get a DBN-DNN.

Our DNN computes $p(\text{HMM state}|\text{Acoustic Input})$. To turn this into a HMM emission probability $p(\text{Acoustic Input}|\text{HMM state})$, we simply note:

$$p(\text{Acoustic Input}|\text{HMM state}) = \frac{p(\text{HMM state}|\text{Acoustic Input}) p(\text{Acoustic Input})}{p(\text{HMM state})}$$

$p(\text{HMM state})$ comes from the training data and $p(\text{Acoustic Input})$ can be dropped because it does not affect the final alignment. You can then train the HMM initial state distribution and transition probabilities from the data. To generate the alignment, we apply a GMM-HMM. Once the DBN-DNN is trained, we use it to regenerate the alignment and retrain everything on this new alignment.

5 Phonetic Classification and Recognition on TIMIT

The TIMIT dataset is small, so we might want to use larger datasets. Previous work is trained on TIMIT, so it's good to train on it so you can compare against previous work. Our work tries between 1-8 hidden layers, 512-3072 hidden units per layer, and 7-37 audio frames per input vector. Our model is robust to hyperparameters, so changing them won't change accuracy too much (just 2% change in error).

Typically, MFCCs and their first and second temporal derivatives over a fixed size window of frames are used as the input vector because the vector has independent components, which is helpful for GMMs. Since neural nets don't need independent components, you can actually use the raw filter banks as input vectors (gives 1.7% error reduction).

In our model, the DBN-DNN is trained separately from the HMM parameters. Let us see if we can incorporate maximum mutual information (MMI) to make the DBN-DNN aware of the sequential nature of the data. Given utterance $v_{1:T}$ (or equivalently, the hidden features from the DBN-DNN $h_{1:T}$) with labels $l_{1:T}$, the MMI probability is:

$$p(l_{1:T}|v_{1:T}) = p(l_{1:T}|h_{1:T}) = \frac{\exp(\sum_{t=1}^{T} \gamma_{ij}\phi_{ij}(l_{t-1}, l_t) + \sum_{t=1}^{T}\sum_{d=1}^{D} \lambda_{l_t,d} h_{td})}{Z(h_{1:T})}$$

where ϕ_{ij} is an indicator indicating whether the transition happened and γ_{ij} is the transition probability. h_{td} is the dth dimension of the hidden vector for frame t (D is the dimensionality of the hidden vector). We can then take the gradient of the log-likelihood and backpropagate to update the parameters. We initialize our DNN weights by training it on the per-frame cross entropy, as discussed before. Our transition parameters come from HMM transition matrices and the phone language model. We can further tune them once the DBN-DNN has been trained. Then, we apply the the MMI training. This can give a 5% error reduction.

So far, we've discussed phone recognition where the input is the utterance (i.e. audio signal) and output is the phone sequence. In phone classification, the input is an audio signal bounded to contain a single phone, and the output is simply the phone. A convolutional DBN-DNN (shares weights over time and uses max-pooling) can solve this problem. Phone classification models cannot translate easily to phone recognition because they can't handle long sequences as well.

The RBMs are a product of experts while the GMM components are a sum of experts. The former is more sample efficient. A DNN can model highly nonlinear data, while a GMM is not so flexible (assumes each point comes from one component). GMMs with diagonal covariances require uncorrelated input vector dimensions, which is restrictive. GMM learning is more easily parallelized.

6 Comparing DBN-DNNs with GMMs for Large Vocabulary Speech Recognition

TIMIT is small, so we consider five larger datasets. For these, it's essential that we use tied triphone (computed from decision tree) outputs instead of monophone outputs because they have more information per bit. We have 761 tied triphones and our model has 5 hidden layers, 2048 hidden units per layer, and an 11-frame context window. For the below, a baseline GMM-HMM system generates the alignment used for training.

On the Bing voice search dataset, we train on 24 hours of data to get 69.6% accuracy vs. the 63.8% from a GMM-HMM. Training on 48 hours of data does not make much of a difference in the unsupervised step, but makes a huge difference for the supervised step (71.7% accuracy).

On the Switchboard dataset, we train on 300 hours of data and thousands of tied triphone states. The wins here came from the direct modeling of tied triphone states and the modeling power of deeper networks - pretraining does not help much. You don't need preprocessing techniques like LDA for DNNs.

On the Google voice dataset, we train on 5870 hours of data with 7969 tied triphone states. The input was 40 log filter banks in an 11-frame window. We used both regular per-frame training and MMI training here. The word error rate was 12.3%.

On the YouTube dataset, we train on 1400 hours of data with 17552 tied triphone states. We use fewer hidden layers with fewer units because there are so many tied triphone states.

On the English Broadcast News dataset, we train on 50 hours of data with 2200 tied triphone states.

Overall, the experiments show that DNN-HMMs beat GMM-HMMs, even if the latter is trained on more data.

To make the DNN-HMM fast, you can make all its weights 8 bits (instead of 32 or 64) and use fast SIMD operations on your CPU. Alternatively, you can just use a GPU.

The RBM is not the only way to train a network. You can start with a single hidden layer and supervised train the whole network. Then you can add another hidden layer and continue supervised training. Repeat this until your network is deep enough. You can also replace an RBM with a well regularized Autoencoder.

It's hard to parallelize backpropagation. The best we can do is parallelize the matrix operations with a GPU.

7 Other Ways of Using Deep Neural Networks for Speech Recognition

You can feed the neural network output as input features to the GMM-HMM - this is called a TANDEM system. Ensembling them with GMM-HMM also helps.

8 Summary and Future Directions

The traditional GMM-HMM system can be improved by replacing the GMM with a DBN-DNN or using it in tandem. The DBN-DNN is trained with a combination of unsupervised generative training (with RBMs) and supervised fine-tuning. Future work should try new architectures, hidden units, and training/pretraining algorithms. Figuring out how to parallelize training across machines would also help a lot.