Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition

1 Citation

Dahl, George E., et al. "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition." IEEE Transactions on audio, speech, and language processing 20.1 (2012): 30-42.

`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.337.7548&rep=rep1&type=pdf`

2 Abstract

We make a Context Dependent, Deep Neural Network Hidden Markov Model (CD-DNN-HMM) that predicts a distribution over senones (tied triphones) for a given speech sample. It beats state of the art CD-GMM-HMMs by a relative improvement of of between 16 to 23.2%. Unsupervised pretraining the DNN is important for generalization.

3 Introduction

Deep Belief Nets use an unsupervised technique for training one layer at a time. Then, the entire model is fine-tuned with supervised learning. Even if the unsupervised dataset is the same as the supervised dataset, pretraining still helps because it acts like a regularizer.

Previous work has used neural networks to estimate HMM posterior probabilities. They use an embedded Viterbi algorithm for efficient training. They gave a small improvement over GMM-HMMs. They were hard to train because they were fully supervised. We use pretrained deep nets to avoid this issue. Other previous work introduced the TANDEM approach, where some of the GMM input features come from neural nets. Autoencoders are another unsupervised pretraining approach, but we find that Deep Belief Nets work better.

Our CD-DNN-HMM is deeper than previous models and we also output senone probabilities, which is different than context-independent approaches that predict a context and a phone. The senones combine the phone and context information. We evaluate on the Bing voice search dataset.

4 Deep Belief Networks

Start with a Restricted Boltzmann Machine (RBM) and train it in an unsupervised manner. Then, stack another RBM on top. Repeat several times and then stack a fully connected + softmax layer. Fine-tune the whole system on supervised data (with backpropagation and SGD). This is a Deep Belief Net. This system is more sample-efficient than Gaussian Mixture Models (GMMs). Now, let's discuss RBMs. An RBM is a bipartite graph between visible and hidden units. Assume both visible and hidden units are binary (Bernoulli) for now. We have an energy function $E(\mathbf{v}, \mathbf{h}) = -\mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h}$, which can be used to define a probability $P(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z}$ for partition function $Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$, we can then

marginalize to get: $P(h_i = 1|\mathbf{v}) = \sigma(c_i + \mathbf{v}^T\mathbf{W}_{*,i})$ and $P(v_j = 1|\mathbf{h}) = \sigma(b_j + \mathbf{h}^T\mathbf{W}^T_{*,j})$. Now, we can use these equations to define a log likelihood. Then, we can write a gradient descent (with momentum) update operation: $\triangle w_{ij}(t+1) = m\triangle w_{ij}(t) - \alpha\frac{\partial l}{\partial w_{ij}}$. In practice, the log likelihood is expensive to compute, so we approximate it with contrastive divergence:

$$-\frac{\partial \ell(\theta)}{\partial w_{ij}} \approx E_{data}[v_i h_j] - E_1[v_i h_j]$$

The first term is the relative frequency with which visible unit i and hidden unit j are both on in the data. The second term is the relative frequency under the RBM reconstruction (i.e. initialize visible units from data example, infer hidden with marginal distribution, infer visible with marginal distribution). To generalize to Gaussian (linear) visible units, we do $E(\mathbf{v}, \mathbf{h}) = \frac{1}{2}(\mathbf{v} - \mathbf{b})^T(\mathbf{v} - \mathbf{b}) - \mathbf{c}^T\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h}$ and our marginal distribution becomes $P(\mathbf{v}|\mathbf{h}) = \mathcal{N}(\mathbf{v}; \mathbf{b} + \mathbf{h}^T\mathbf{W}^T, I)$. Once we train one RBM, we can freeze its weights, treat its hidden units as input to a new RBM, and repeat.

5 CD-DNN-HMM

An HMM with states $S = \{s_1, ..., s_K\}$ requires an initial state distribution $\pi = \{p(q_0 = s_i)\}$, transition probabilities $a_{ij} = p(q_t = s_j|q_{t-1} = s_i)$, and emission probabilities $p(\mathbf{x}_t|s_i)$. Traditional models use GMMs for emission probablities. We model it with our neural network instead, where $\mathbf{x}_t$ is the senone probability distribution. Now, given $\mathbf{x}$, how do we find the most likely word (or word sequence) $\hat{w}$? $\hat{w} = \text{argmax}_w p(w|\mathbf{x}) = \text{argmax}_w p(\mathbf{x}|w)p(w)/p(\mathbf{x})$. The $p(w)$ is the language model. The $p(\mathbf{x})$ is a normalization constant. The acoustic model is: $p(\mathbf{x}|w) = \sum_q p(\mathbf{x}, q|w)p(q|w) = \max \pi(q_0) \prod_{t=1}^T a_{q_{t-1}q_t} \prod_{t=0}^T p(\mathbf{x}_t|q_t)$. The emission probability is $p(\mathbf{x}_t|q_t) = p(q_t|\mathbf{x}_t)p(\mathbf{x}_t)/p(q_t)$, where $p(q_t|\mathbf{x}_t)$ comes from the DNN, $p(\mathbf{x}_t)$ is independent of the word sequence and can be ignored, and $p(q_t)$ comes from the training set.

Here's the algorithm to train our model. First, we train a CD-GMM-HMM, where the tying of triphones is handled by a decision tree - this model is called gmm-hmm. Each of the tied triphones is a senone - we give each one an ID. We make a state2id mapping that turns a tied triphone into a senone. Use the senones, state2id, and transition probabilities for the DNN. Pretrain the DNN using the Deep Belief Net technique. Use gmm-hmm to make a state level alignment on training set, (*) then convert each triphone into a senone. Use these senones to backpropagate to fine-tune the neural network. Now, with the DNN, estimate senone prior probabilities and recompute the transition probabilities. If the resulting model shows validation improvement, that's great. If not, use it to generate a new raw alignment and repeat from (*).

6 Experimental Results

We train on a Bing voice search dataset where people look up a business name (e.g. McDonald's) by voice. We do a train, test, development (validation) set split by timestamp so training set comes before dev set. We use the Carnegie Mellon language model. Our train/test/dev set has 24/9.5/6.5 examples. We use a 65K word vocabulary. We compare against state of the art GMM-HMMs. Our features are MFCC coefficients (and energy) and their first and second derivatives over 11-frame windows. We have 761 senones.

For pretraining, we use a learning rate of 0.004. For fine-tuning, we use 0.08 for the first 6 epochs and 0.002 for the last 6 epochs. We have 256 item minibatches and 0.9 momentum. We use 2K hidden units. We get 69.6% accuracy. Removing pretraining hurts accuracy and not using senone labels really hurts accuracy. In the future, we might pretrain on a much larger unlabeled dataset. We use 5 hidden layers. Using a better technique to get raw alignments (see (*)) helps accuracy. Re-doing the alignment with the CD-DNN-HMM and training again also helps. Re-estimating the transition probabilities also helps.

We train on an NVIDIA Tesla and pretrain for 62 hours and train for 16.8 hours. Without a GPU, it would take 3 months to train. We can decode at realtime speeds.

7 Conclusion and Future Work

Our CD-DNN-HMM sets state of the art on speech recognition, but is more expensive to train than CD-GMM-HMMs. We train on only 24 hours of data. If you want to scale to thousands of hours of data, we need a way to better parallelize backpropagation. Our embedded Viterbi algorithm isn't optimal - maybe there's a better algorithm. Our Gaussian RBMs assume diagonal covariances, so maybe we can relax this assumption?