Comparing Computation in Gaussian mixture and Neural Network based Large-Vocabulary Speech Recognition

Vishwa Gupta and Gilles Boulianne
Centre de recherche informatique de Montréal (CRIM)

Abstract

In this paper we look at real-time computing issues in large vocabulary speech recognition. We use the French broadcast audio transcription task from ETAPE 2011 for this evaluation. We compare word error rate (WER) versus overall computing time for hidden Markov models with Gaussian mixtures (GMM-HMM) and deep neural networks (DNN-HMM). We show that for a similar computing during recognition, the DNN-HMM combination is superior to the GMM-HMM. For a real-time computing scenario, the error rate for the ETAPE dev set is 23.5% for DNN-HMM versus 27.9% for the GMM-HMM: a significant difference in accuracy for comparable computing. Rescoring lattices (generated by DNN-HMM acoustic model) with a quadgram language model (LM) and then with a neural net LM reduces the WER to 22.0% while still providing real-time computing.

Index Terms: Speech recognition, ETAPE evaluation, large vocabulary recognition, real-time computing.

1. Introduction

There are many applications that require real-time transcription of audio. Closed-captioning of broadcast audio in Japanese [1], in French [2] and in Portuguese [3] are some of the examples. Other applications currently using near real-time automated transcription are companies that extract news stories automatically from broadcast audio [4]. The requirement here is fast near real-time transcription of hundreds of channels of broadcast audio. All these applications require low error rate with minimal computing resources.

Many papers have been published recently that optimize computing for on-line or off-line transcription of broadcast audio. Breslin et al. [5] show how to perform on-line speaker clustering and adaptation in order to reduce error rate without significant increase in computing or delay. Prazak et al. [6] use over a million word dictionary and multi-core machines in order to get accurate real-time transcription of re-spoken Czech broadcasts. In [7], the authors investigate on-line fMLLR adaptation for a small vocabulary telephone speech application.

All the published papers on on-line or real-time speech recognition use Gaussian mixture models. Only recently, Mohamed et al. [8] and Dahl et al. [9] showed that deep neural nets (DNN) can replace the Gaussian mixtures in HMMs to produce state posterior probabilities that give significantly reduced word error rate (WER). In [9], the authors use CD-DNN-HMM system and compare both accuracy and compute time with their CD-GMM-HMM system. The task is US-wide business and web search from mobile phones via voice. They recognize short phrases (average sentence length of 2.1 words) using a 65k vocabulary. For this task, the CD-GMM-HMM system decodes in 0.54 times real time, while the CD-DNN-HMM system decodes in 0.58 times real time without GPU and 0.17 times real time with GPU. They were the first to show that DNN-HMM system can reduce WER without significant increase in computing.

In this paper we investigate both the GMM-HMM-based and DNN-HMM-based speech recognition systems in order to optimize computing speed versus accuracy. We do not consider latency. The goal is to find optimal word error rate (WER) versus computing time curve so that we can choose the operating point based on the application requirements. We investigated this in the context of French broadcast audio transcription task in the ETAPE 2011 evaluation [10]. We used state-of-the-art GMM-HMM and DNN-HMM algorithms. The GMM-HMM models were trained using discriminative training [11] that gave us very good results in ETAPE evaluation [12]. We experimented with DNNs with 2 to 6 hidden layers and both MFCC and filter-bank feature parameters.

We found that the GMM-HMM’s were very sensitive to beam-widths and reducing beams resulted in significant increase in WER. Reducing the number of mixtures in GMM-HMMs reduced the computing time, but also increased the WER significantly. The DNN-HMM hybrid recognition was the most effective way of reducing computing without affecting error rate. Reduced beamwidth for DNN-HMMs reduced computing with only marginal increase in error rate. In fact, we could reduce computing to a point where the computing time for the feature parameters becomes significant relative to the search and acoustic model scoring. The DNNs model the acoustics much better than the GMMs.

2. Acoustic training and test data

The ETAPE training data consisted of training data for ESTER 2 evaluation1 and the more recently transcribed audio for ETAPE evaluation [10] for a total of 300 hours of audio. We also had 178 hours of internally transcribed audio from French TV broadcasts in Quebec. Overall, we had 478 hours of transcribed audio for training. In all the training audio, speaker segments were manually labeled in order to facilitate speaker-adapted training.

The ETAPE development set consisted of 15 files of 10 minutes to 1 hour in duration for a total duration of 8.6 hours. They were recorded from French radio and TV programs.

---

1 The data set resulting from the ESTER 2 evaluation campaign comprises about 250 hours of radio broadcast transcribed by human listeners, as well as the newspaper corpus Le Monde from 1987 to 2003. These datasets are distributed by ELDA and by the DGA.
3. GMM-HMM-based System Optimisation

We tried two different strategies in order to optimize computing versus recognition accuracy. In the first strategy we generated the best possible acoustic models and reduced the computing by playing with the search beamwidth. In the second scenario, we generated acoustic models with GMMs containing fewer Gaussians and then reduced the computing by playing with the search beamwidth. We tried four different GMM-HMMs with 4148 states and 40k (model 2), 200k (model 3), 100k (model 4) and 50k (model 5) 40-dimensional Gaussian means and variances.

The strategy used model 2 that gave very good results in the ETAPE 2011 evaluation [12]. The models above represent input speech by 13 MFCCs and a context window of 9 frames. After speaker diarization, the features are mean and variance normalized on a per speaker basis. An LDA transform reduces the dimensionality to 40. There are 36 phones. Each phone has separate representation for word begin, word end, word middle, and word-begin-and-end (usual in the Kaldi toolkit [14]). Words are represented as a sequence of phones, and the phones are 3-state HMMs with self-loop and next state transitions. The models have a tri-phone cross-word context.

The speaker-adapted acoustic models 2, 3, 4 and 5 are discriminatively trained using feature-space maximum mutual information (fMMI) training interleaved with boosted maximum mutual information (BMMI) training (fMMI+BMMI training) [11]. (We got better WER versus compute time curve with fMMI+BMMI trained models than with BMMI trained models). The fMMI+BMMI discriminative training of the largest model took about a week, even though the training was distributed among 50 processors. The smaller models trained much faster.

We also trained a model with 2.5k states and 15000 Gaussian means (model 1) for computing fMLLR transform for each speaker. Model 1 gives fMLLR transforms that achieve lower WER and take less time to compute [15].

The GMM-HMM recognition scenario includes:

1. Diarization of each audio file into speaker clusters (no cross-file diarization).
2. speaker-independent (SI) decoding of each speech segment using a small model (model 1). Resulting lattices are used to estimate initial fMLLR transform/speaker.
3. speaker-adapted decoding using fMLLR transform from previous step. Resulting lattices are used to re-estimate fMLLR transforms/speaker.
4. speaker-adapted fixed frame rate decoding with larger models using fMLLR from the previous step.

Note that we do not carry out variable frame-rate decoding [17] after fixed frame-rate decoding, as it increases the computing time significantly while reducing the WER by only around 0.25% [12]. The fixed frame-rate decoding always uses 10 ms frame advance and 25 ms frame length.

The diarization (step 1) used a multi-stage GMM-based segmentation and clustering algorithm [18]. The diarization took 0.25 times real-time on Intel Xeon CPU E5-2630@2.3GHz processors. Steps 2 and 3 are needed for computing the fMLLR transform for each speaker cluster obtained in step 1. For these steps, we used a speaker-adaptively trained GMM-HMM model with 2.5k states and 15000 mixtures (model 1). We experimented with the search beamwidths in steps 2 and 3 in order to reduce computing time. We measured the tradeoff between WER versus compute time for different beam widths. Table 1 shows these tradeoffs. In this Table, the WER in the third column is computed using model 2 and fMLLR transforms from model 1.

<table>
<thead>
<tr>
<th>SI beam</th>
<th>SA beam</th>
<th>WER</th>
<th>x RT for fMLLR computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>25.68%</td>
<td>0.91</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>25.67%</td>
<td>0.48</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>25.72%</td>
<td>0.32</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>25.71%</td>
<td>0.20</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>25.84%</td>
<td>0.13</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>25.93%</td>
<td>0.11</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>26.31%</td>
<td>0.09</td>
</tr>
</tbody>
</table>

Note that we get 30.06% WER when we decode using model 2 without any fMLLR transforms (speaker-independent decoding). (For a large discriminatively trained SI model, we achieved 29.6% WER [12]). From the Table, we can see that the fMLLR computation time can be reduced from 0.9 times real-time to 0.2 times real-time with only a marginal increase in WER. For the remaining experiments, we generated the fMLLR transforms with a beamwidth of 6 for both steps 2 and 3.

The GMM-HMM recognition scenario includes:

1. Diarization of each audio file into speaker clusters (no cross-file diarization).
2. speaker-independent (SI) decoding of each speech segment using a small model (model 1). Resulting lattices are used to estimate initial fMLLR transform/speaker.
3. speaker-adapted decoding using fMLLR transform from previous step. Resulting lattices are used to re-estimate fMLLR transforms/speaker.
4. speaker-adapted fixed frame rate decoding with larger models using fMLLR from the previous step.

Note that we do not carry out variable frame-rate decoding [17] after fixed frame-rate decoding, as it increases the computing time significantly while reducing the WER by only around 0.25% [12]. The fixed frame-rate decoding always uses 10 ms frame advance and 25 ms frame length.

The diarization (step 1) used a multi-stage GMM-based segmentation and clustering algorithm [18]. The diarization took 0.25 times real-time on Intel Xeon CPU E5-2630@2.3GHz processors. Steps 2 and 3 are needed for computing the fMLLR transform for each speaker cluster obtained in step 1. For these steps, we used a speaker-adaptively trained GMM-HMM model with 2.5k states and 15000 mixtures (model 1). We experimented with the search beamwidths in steps 2 and 3 in order to reduce computing time. We measured the tradeoff between WER versus compute time for different beam widths. Table 1 shows these tradeoffs. In this Table, the WER in the third column is computed using model 2 and fMLLR transforms from model 1.

<table>
<thead>
<tr>
<th>model beam</th>
<th>WER</th>
<th>times-real-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>25.71%</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>26.16%</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>27.02%</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>29.18%</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>34.03%</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>25.92%</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>26.37%</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>27.20%</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>29.32%</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>34.32%</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>26.69%</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>26.98%</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>27.92%</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>30.26%</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>35.15%</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>27.48%</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>27.73%</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>29.02%</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>31.63%</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>36.56%</td>
</tr>
</tbody>
</table>

The GMM-HMM recognition scenario includes:

1. Diarization of each audio file into speaker clusters (no cross-file diarization).
2. speaker-independent (SI) decoding of each speech segment using a small model (model 1). Resulting lattices are used to estimate initial fMLLR transform/speaker.
3. speaker-adapted decoding using fMLLR transform from previous step. Resulting lattices are used to re-estimate fMLLR transforms/speaker.
4. speaker-adapted fixed frame rate decoding with larger models using fMLLR from the previous step.

Note that we do not carry out variable frame-rate decoding [17] after fixed frame-rate decoding, as it increases the computing time significantly while reducing the WER by only around 0.25% [12]. The fixed frame-rate decoding always uses 10 ms frame advance and 25 ms frame length.
Fig. 1 plots WER versus compute time for these models. From this figure it is clear that between 0.9 times and 1.6 time real-time, model 3 provides the best tradeoff. At lower computing, we have to use models 4 or 5. We can also see that as we lower the beamwidth, the WER increases rapidly.

To be complete, we tried GMM-HMMs trained with PLP features instead of MFCC, and also subspace Gaussian mixture models (SGMMs) to see if they have better WER versus compute time. For PLP, the fMLLR transforms took 0.21 times real-time. We trained PLP models with 4148 states and 200k mixtures using fMMI+BMMI (model-3-PLP). We trained an SGMM model using BMMI training and MFCC’s as feature parameters (model-3-SGMM). Fig. 2 plots WER versus compute time for model-3, model-3-PLP and model-3-SGMM. We can see that GMM-HMMs trained with MFCCs give the best compute time versus WER curve. Even though the WER for PLP and MFCC features is comparable, the compute time versus WER curve for PLP is significantly worse than that for MFCCs.

4. DNN-HMM based System Optimisation

The deep neural net HMM hybrid acoustic model (DNN-HMM) has been shown recently to be very effective in reducing WER [9] [8] [20]. We ran DNN-HMM acoustic modeling experiments with 4, 5, 6, 7 and 8 layer neural nets. For training the neural net, we divided the 478 hours of training audio into 475 hours for training and 3 hours for validation. We got only small improvements with neural nets having more than 7 layers. So we carried out all the experiments with a 7 layer neural net. For input features, we experimented with filter-bank and MFCC features. We tried both the TRAP features [21] and the features with delta and delta-delta added to them as input to the neural net. The TRAP features extracted from filter-bank or MFCC features gave lower WER than the filter-bank or the MFCC features enhanced with delta and delta-delta coefficients. TRAP features derived from filter-bank features gave significantly lower WER than from MFCC features (23.40% vs 24.62% for a 7-layer NN). The 7-layer neural net for the filter-

For recognition with the 7-layer DNN-HMM hybrid acoustic models, Fig. 3 compares WER versus compute time for CPU (Intel Xeon CPU E5-2630@2.3GHz) only case (NN-7L-noGPU), for combined CPU/GPU case (NN-7L-GPU), and for GMM-HMMs using model 3. The GPU (Tesla M2075 with...
We trained the neural net LM with roughly 350M words of text using the recurrent neural network language modeling toolkit (RNNLM) [22]. The vocabulary size was reduced from 140k [12] to 40k most common words and the neural net (NN) LM had 300 neurons in the hidden layer. This NN also contains a class based maximum entropy model [23]. The maximum entropy model is a weight matrix that directly connects the input and output layers. To represent the weights for the trigram features, a hash function maps the huge sparse matrix into one-dimensional array. We chose this array size to be $2 \times 10^9$. Out of the 350M words of training corpus, only 2M words are in-domain text. We tried RNNLM’s with just 2M in-domain words versus 350M words of text. The RNNLM with 350M words outperformed the RNNLM with 2M in-domain text.

We combined the neural net LM with the quadgram LM using a weight of 0.25. On 76,500 words of the Dev set, the quadgram LM perplexity is 113, while that of the neural net LM is 109, and the combined perplexity is 92 [12]. Fig. 5 compares WER versus compute time for quadgram LM rescoring (NN-7L-GPU-resLM) versus quadgramLM+neural net LM rescoring (NN-7L-GPU-resLM-nnetLM). We see from the plot that the neural net LM is only effective at WERs below 22.5%. Even at WER of 21.5% the compute time is well below real-time.

5. Conclusions

We show here that the DNN-HMM hybrid acoustic recognition significantly lowers the WER compared to the state-of-the-art GMM-HMM acoustic models. At 26.4% WER, DNN-HMMs are 3.5 times faster than GMM-HMMs. The DNN-HMMs are easier to implement in a real-time recognition scenario as we do not have to compute an fMLLR transform. We also show that rescoring output lattices from DNN-HMM recognition using quadgram LM reduces WER and computing. However, rescoring with the neural net LM is compute effective only at the low end of WER range. In the future, we will investigate reducing latency for on-line recognition with DNN-HMMs.
6. References


