Stochastic Gradient Adaptation of Front-End Parameters

Sreeram V. Balakrishnan, Karthik Visweswariah, Vaibhava Goel

IBM T. J. Watson Research Center, Yorktown Heights, NY - 10598
{sreevb,kv1,vgoel}@us.ibm.com

Abstract

This paper examines how any parameter in the typical front end of a speech recognizer, can be rapidly and inexpensively adapted with usage. It focusses on firstly demonstrating that effective adaptation can be accomplished using low CPU/Memory cost stochastic gradient descent methods, secondly showing that adaptation can be done at time scales small enough to make it effective with just a single utterance, and lastly showing that using a prior on the parameter significantly improves adaptation performance on small amounts of data.

It extends previous work on stochastic gradient descent implementation of fMLLR [1] and work on adapting any parameter in the front-end chain using general 2nd order optimization techniques [2].

The framework for general stochastic gradient descent of any front-end parameter with a prior is presented, along with practical techniques to improve convergence. In addition the methods for obtaining the alignment at small time intervals before the end of the utterance are presented. Finally it shown that experimentally online causal adaptation can result in a 5-15% WER reduction across a variety of problems sets and noise conditions, even with just 1 or 2 utterances of adaptation data.

1. Introduction

Adapting a speech recognizer automatically to new environments is often a critical real world product requirement. Several highly effective techniques for online adaptation such as Maximum Likelihood Regression (MLLR) [3] and feature space MLLR [4] have been developed. However the existing techniques suffer from the disadvantages of 1) requiring 10-20 seconds of data to be effective, 2) requiring large amounts of computation to collect $O(N^3)$ statistics and to estimate the adaptation parameters, 3) operating at time scales greater than one utterance.

This paper tackles all of these disadvantages by extending the novel stochastic gradient descent approach to adaptation presented in [1] to the general case of any front end parameter. In addition it looks at applying adaptation updates at small time intervals so that effective adaptation can be achieved with purely causal adaptation on just one utterance.

The key assumption of this paper is that adaptation is best achieved by maximizing a suitably defined objective function of the test data with respect to the free parameters of the front end chain. Typically, this objective function is the likelihood of the test data computed from the underlying continuous density HMM. The acoustic models and task grammar are assumed to be fixed and are not updated during the adaptation. The front-end chain used in this paper is a typical cepstra plus LDA chain. The parameters of the front-end are optimized by computing the gradient of the objective function of the test data with respect to these parameters at fixed time intervals. At the end of the interval, the gradient is used to compute a stochastic update of the front-end parameter.

Further improvement in performance is achieved by using a prior on the parameter being adapted, which has previously been found effective for MLLR [5]. In our case, the prior is obtained from the training data by adapting the parameter for each speaker/condition in the training data, and then creating a Gaussian prior from the resulting distribution of values.

The likelihood functions used in the objective function, are computed from the alignment of the recognized output. To allow adaptation to occur at short time intervals, we show that partial alignments computed from the best scoring state at frame $t$, can be used for adaptation purposes at $t - d$ where $d$ can be as small as 10 frames.

The rest of this paper is organized as follows. In section 2 and 3 we present our formulation of the speech front-end chain and the mathematical framework for the stochastic gradient descent of the test data objective function. Section 4 discusses the use of a prior during adaptation. Sections 5 describes the method used to compute alignments at regular intervals before the end of the utterance. Section 6 gives details of how the stochastic descent is implemented and how pre-scaling can be used optimize the rate of convergence. Finally, section 7 presents the results on both our Embedded in-car test sets and Telephony bandwidth in car test set.

2. Front-End Chain

All the results in this paper are based on adapting a standard speech front-end chain using MFCCs with mean normalization and an LDA transform. This is described below as a set of functions $f^i_{-d} = \Lambda^i(f^i_{-d_1} \cdots f^i_{-d_t} \cdots f^i_{+d_t})$ where the output of function $\Lambda^i$ becomes the input to function $\Lambda^{i-1}$. The value of $d_i$ determines how big a time window of the output of $\Lambda^i$ is fed into the input of $\Lambda^{i-1}$. Typically, it is 0 except for the normalization and splicing steps.

For this paper the scope of adaptation is limited to the parameters in the front-end chain between the output of the mel filter banks and the final speech observation vector $\alpha_f$. This is just for convenience and the approach presented in this paper can be extended further back in the front-end chain all the way to the raw pcm. Let $f^i$ be the final observation $\alpha_f$ at the end of the front-end chain. The front-end chain between the final output $\alpha_f$ and mel filter output is given by:

1. Feature space transform $f^i_o = Af^i_f + b$
2. LDA transform $f^i_o = \phi f^i_o$
3. Splice frames $f^i_o = [f^i_{-d_1} \cdots f^i_{+d_t}]$
4. Transform normalized cepstra $f^i_o = \Gamma(f^i_{-T_1} \cdots f^i_{+T_2})$
5. Normalize Cepstra $f^i_o = C f^i_o$
6. Cepstral rotation $f^i_o = \Phi f^i_o$
7. log of mel filter $f^i_{\text{mel}} = k \log(f^i_{\text{mel}})$

In our formulation, the free parameters are: the feature space transform $A, b$, the LDA transform $\phi$, and the normalized cepstra transform $\Gamma$. The cepstral rotation $\Phi$ could also be adapted, however this is not covered in this paper. Constrained or Feature space Maximum Likelihood Linear regression (fMLLR) introduced in [4], is just the special case of adapting the feature space transform.
3. Computing Gradient of the objective function

Our approach to adaptation requires the computation of the gradient of a suitably defined objective function of the test data, with respect to the free parameters that are being adapted. We follow the derivation given in [2], where the full description of the framework can be found.

We assume that the underlying model is a continuous density Hidden Markov Model. Based on [2] we compute the gradient of an objective function of the form

$$\log(L) = \log\left(\sum_S P(O_T^T, S_T^T | \omega) \right) + T \log(|J|)$$  \hspace{1cm} (1)

or

$$\log(L) = \log\left(\sum_S P(O_T^T, S_T^T | \omega) \right) + \frac{T}{2} \log|\text{cov}(O_T^T)|$$  \hspace{1cm} (2)

where $O_T^T$ is the observation sequence $o_1 \cdots o_T$, $S_T^T$ is the state sequence $s_1 \cdots s_T$ and $\omega$ are the CDHMM model parameters, comprising the means, variances and mixture weights of the Gaussian Mixture models tied to each state, and the transition probabilities.

The first objective function is used for case where the adaptation of the parameters can be expressed as a linear transform of the observation vectors. This is the case for the feature space transform $A$, $b$. The second term is a Jacobian correction term of the form

$$J_{ij}(\bar{o}, o) = \frac{\partial o_i}{\partial o_j}$$

where $\bar{o}$ is the observation vector after the front-end parameters have been adapted.

The second objective function is for the more general case where $\bar{o}$ cannot be expressed as a function of $o$. In this scenario we add a second term that is the log of the determinant of the covariance of the test data $O_T^T$. Another way of viewing this term is as an approximate denominator term for an MMI objective function [2]. According to this view we are using an MMI objective of the form

$$\log(L) = \log(L_{num}) - \log(L_{denom})$$

where the numerator likelihood is just the log probability of the test data from the recognition model and the denominator likelihood is computed for a confusion model which is more flexible than the numerator model. As an extreme we could consider the denominator to be a full covariance Gaussian estimated from the test data. In this case the likelihood would be exactly $\frac{T}{2} \log|\text{cov}(O_T^T)|$

Following [2] we express the gradient of $\log(L_{num})$ in terms of the auxiliary function of the underlying CDHMM.

$$\nabla_\alpha \log(L_{num}) = \sum_\tau \sum_m \gamma_{\tau m} \times \frac{\partial}{\partial \alpha} \exp\left(-\frac{1}{2} (o_\tau - \mu_m)^T P_m (o_\tau - \mu_m) \right)$$

where $\gamma_{\tau m}$ is the fractional allocation of Gaussian $m$ with mean $\mu_m$ and precision matrix $P_m$ to frame $\tau$.

To further simplify the computation, following [2] we use the chain rule to separate the gradient computation into two parts. The first computes the gradient of $\log(L)$ with respect to the each observation vector, the second computes the gradient of each observation vector with respect the the front-end parameter we are adapting.

Let:

$$Y_{\tau i} = \nabla_o \log(L)$$

$$= \frac{\partial}{\partial \tau i} \left(\sum_\tau \sum_m \gamma_{\tau m} \exp\left((o_\tau - \mu_m)^T P_m (o_\tau - \mu_m) \right) \right)$$

$$+ \frac{\partial}{\partial \tau i} \frac{T}{2} \log|\text{cov}(O_T^T)|$$

$$= [-P_m (o_\tau - \mu_m) + \text{cov}(O_T^T)^{-1} (o_\tau - \bar{o})]_i$$

where $[.]_i$ picks out the $i$th element of a vector and $\bar{o}$ is the mean of $O_T^T$.

The gradient now becomes

$$\nabla_\alpha \log(L) = \sum_\tau \sum_i Y_{\tau i} \nabla_o Y_{\tau i}$$

The computation of $\nabla_o Y_{\tau i}$ itself can be simplified by applying the chain rule recursively to each stage in the front-end, up to the stage that uses the parameter $\alpha$.

$$\nabla_o Y_{\tau i} \Big|_{f_{j+1}^{i-1}} = \sum_\mu \sum_j \frac{\partial f_{j+1}^{i-1}}{\partial o_{j+1}^i} \nabla_o f_{j+1}^i$$

For most cases $f_{j+1}^{i-1}$ just a function of $f_j^i$ and the summation over $\mu$ can be dropped.

For the linear stages 1 and 4 we can apply the chain rule to get the following expressions for the gradients:

$$\nabla_A \log(L) = \sum_{w=1}^W \hat{w}^T \hat{\Phi}^1$$

$$\nabla_R \log(L) = \sum_{w=1}^W \hat{w}^T \hat{\Phi}^1 \hat{\Phi}^2 \hat{\Phi}^6 (w)$$  \hspace{1cm} (3)

where $w$ picks out each time shift used in splicing $\hat{f}_w^i$ to get $f_w^i$ with $F_w^i (w) = f_w^i (w + 1)/2$ and $\hat{\phi} = [\phi_1 \cdots \phi_W]$.

The advantage of this approach is that the gradient computation can be easily modularized. The first module computes $\hat{\Phi}$ independently of which parameter is going to be adapted. The second module combines $\hat{\Phi}$ with the gradient of $O_T^T$ for the relevant parameter. This module itself can be implemented as individual modules for each stage of the front-end. However, it may be computationally more efficient to use a single module when $f_{j+1}^{i-1}$ is only dependent on a subset of the variables in $\alpha$.

While front-end stages 1-4 are linear, stage 5 (cepstral normalization) and stage 7 (log compression) are non-linear. However, as long as the stage has an analytic transform we can still apply the chain rule and compute the appropriate gradients. Unfortunately, the likelihood with the adapted front-end may no longer be easily comparable with the non-adapted likelihood. This is because non-linear transforms affect the Gaussian pdfs very differently depending on the exact part of the observation space the Gaussian is centered on. The fixed Jacobian or test covariance correction terms defined above may not compensate for this effect correctly. For this reason, we save adaptation of the non-linear transform for future work, although the general stochastic gradient descent and chain rule framework can be easily extended to cover these stages.

4. Using a prior on the adaptation parameters

Since we are interested in adaptation with very small amounts of data, it is beneficial to constrain the adaptation space as much as possible. We estimated a prior on the feature space transform $A$ from the training data. Specifically, standard fMLLR [4] was used to estimate $A$ and $b$ for each speaker/condition in
the training data (approximately 4500 different transforms). We then computed the covariance $\Theta$ of $\text{vec}(A)$ assuming the mean is identity transform. We then used the following gaussian prior for $A$ given the training observations $O_t$

$$P(A|O_t) = |\Theta| \exp\left(-\frac{1}{2}(\text{vec}(A - I))^{T} \Theta^{-1} (\text{vec}(A - I))\right)$$

(4)

In our system, we used 39 dimensional observations, hence, $\Theta^{-1}$ is a $1521 \times 1521$ matrix. To reduce the computation and memory requirements we approximated $\Theta^{-1}$ using factor analysis so that

$$\Theta^{-1} = D\alpha - ZZ^{T}$$

(5)

where $D\alpha$ is a diagonal matrix and $Z$ is a rectangular matrix $1521 \times N$. We found that setting $N = 39$ gave good results, while enormously reducing the computation and memory requirements of using the prior.

5. Computing the partial alignments

It is necessary to compute the alignment of the recognized result in order to compute the gradient of the likelihood. Normally the alignment can only be computed after the whole utterance is recognized. Hence for adaptation to be effective for a single utterance, two recognition passes would be needed, causing a potentially unacceptable delay. In addition, it is desirable to be able to adapt alignment updates to the front-end parameters multiple times per utterance, so that online adaptation within a single utterance is possible, without the need for a second recognition pass. In this scenario partial alignments will be required multiple times before the utterance has ended.

In order to handle this, we divide the utterance into blocks of $B$ frames. At frame $t = Bk$ we compute the state alignment $a_{t}^{(k)}$ from $t = 1$ to $t = Bk$ for the best scoring state at time $t = Bk$. Based on the assumption that the further back in time we go the more reliable the alignment becomes, we use the alignment $a_{t}^{(k)}, \cdots, a_{B_k}^{(k)}$ for adaptation purposes. The parameter $d$ is the delay beyond which we assume the alignment becomes reliable. Hence whatever adaptation can be done based on all the frames up to $t = Bk - d$, can be applied to the decode of frames $Bk$ onwards.

We save computation for multiple alignments, by computing the alignment $a_{t}^{(k)}$ backwards in time from $t = Bk$ to $t = 1$. If we find a $t$ such that $a_{t}^{(m)} = a_{t}^{(m-1)}$, based on the first order Markov assumption we can set $a_{t}^{(m)} = a_{t}^{(m-1)}$ for all $t \leq t$. With this simplification, we find that the extra overhead in computing the partial alignments every 10 – 20 frames is only 20-30% over the cost of computing a single alignment at the end of the utterance.

6. Stochastic Gradient Descent

In [2] the objective function defined above is optimized using a second order approximation technique that uses the BFGS algorithm with the More-Tuente line search. This requires the collection of sufficient statistics of order $O(N^3)$ per frame and an $O(N^4)$ iterative optimization every time the transform is updated. In [1] it was noted that using stochastic gradient descent could achieve the same results at a fraction of the CPU and memory cost. We now extend those results to the general case of adapting any front-end parameter.

As in [1] the basic idea is to compute the gradient of the objective function with respect to a parameter $\alpha$ for a block of $B$ frames and at the end of each block update $\alpha$. If $B$ is equal to the total number of frames of adaptation data $T$, then stochastic gradient descent reduces to simple gradient descent. Stochastic

for $k = 1 \text{...num_blocks}$

for iter = 1 to num_updates(k)

compute $\Delta(k, \alpha) = \beta \sum_{r} \sum_{j} Y_{rj} \nabla_{\alpha} o_{rj}$

$+ \kappa \Delta(k - 1, \alpha)$

update $\alpha$ using $\alpha = \alpha + \Delta(k, \alpha)$

end

adjust $\beta$ to balance speed of convergence with stability

end

Figure 1: Pseudo-Code for implementation of Stochastic Gradient Descent

Gradient Descent is widely used in the Machine Learning community, where it is also known as online learning and is one of the primary methods of learning the parameters of a feed-forward neural network.

6.1. Details of Implementation

Let the observation stream be broken in blocks of length $B$ frames and let $k$ denote the $k^{th}$ block spanning frames $\tau = B(k - 1) + 1 \cdots Bk$.

Let $\Delta(k, \alpha)$ be the total gradient for parameter $\alpha$ for frames $B(k - 1) + 1 \cdots Bk$. The stochastic approximation consists of computing $\Delta(k, \alpha)$ for each block $k$ of feature vectors, and updating $\alpha$ using $\alpha = \alpha + \Delta(k, \alpha)$ where

$$\Delta(k, \alpha) = \beta \nabla_{\alpha} \log(L) + \kappa \Delta(k - 1, \alpha)$$

The scale factor $\beta$ is set to sufficiently small to ensure stability, but is large enough to get meaningful gain in likelihood. Figure 1 shows the Pseudo-Code that describes the algorithm for Stochastic Gradient Adaptation.

The term $\kappa \Delta(k - 1, \alpha)$ is required to include the effect of previous blocks. It is equivalent to the momentum term used in back-propagation training of neural networks. It serves to both smooth out some of the randomness in the stochastic gradient descent and account for the earlier gradients from previous blocks [6]. The values of $\kappa = 0.9$ and $\beta = 0.00002$ were found to give good results. The pseudo-code allows for multiple iterations on the same block. In practice two iterations were found to give good results. A larger number of iterations were found to be useful with the initial seconds of adaptation data, but once enough data is seen, the number of iterations can be reduced.

6.2. Incorporating the prior

For our experiments with the feature space transform $A$ we incorporated the prior $P(A|O_t)$ by adding to gradient update $\Delta(k, \alpha)$ a scaled amount of the gradient of the log of the prior with respect to $A$: so that $\Delta(k, A)$ was modified to

$$\Delta(k, A) = \beta \sum_{r} \sum_{j} Y_{rj} \nabla_{A} o_{rj}$$

$$+ \beta \sigma (ZZ^{T} - D\alpha)(\text{vec}(A - I))$$

$$+ \kappa \Delta(k - 1, A)$$

(6)

where $\sigma$ was set to 0.1.

7. Improving the rate of convergence

The biggest concern with stochastic gradient descent is slow convergence and instability. Short of using second order techniques such as Conjugate Gradients or other Quasi-Newton methods, it is important to ensure that the Hessian of the objective function is was as well conditioned as possible. [1, 6].
For the linear transforms \( \mathbf{A} \) and \( \mathbf{R} \) we found it essential to estimate the transforms in a space where the input feature for the transform has average covariance 1 for each dimension.

This can be implemented by substituting \( \mathbf{D}_\mathbf{A}^{-1} \mathbf{A} \mathbf{D}_\mathbf{R} \) for \( \mathbf{A} \) and \( \mathbf{D}_\mathbf{A}^{-1} \mathbf{D}_\mathbf{R} \) for \( \mathbf{R} \). \( \mathbf{D}_\mathbf{A} \) is a diagonal matrix chosen so that diagonal of the covariance of \( \mathbf{D}_\mathbf{A} \mathbf{D}_\mathbf{A}^{\top} \) is unity and similarly for the covariance of \( \mathbf{D}_\mathbf{R} \mathbf{D}_\mathbf{R}^{\top} \), where \( \mathbf{O}_\mathbf{X} \) is the training data observations and \( \mathbf{F}_\mathbf{X} \) is the training data normalized cepstra. Note if the training data is not available \( \mathbf{D}_\mathbf{A} \) and \( \mathbf{D}_\mathbf{R} \) can be estimated from the test data.

8. Results

We ran two sets of experiments. The first was on a telephony system with 160K Gaussians and the second was on an embedded system with 10k Gaussians. Both systems used the front-end chain from section 2 with final features of 39 dimensions, cepstra of 13 dimensions (including C0) and splicing of 9 frames (center ± 4).

The sizes of \( \mathbf{A}, \mathbf{R} \) and the LDA transform \( \phi \) are 39 × 39, 13 × 13 and 39 × 117 respectively. For the embedded system \( \phi \) was used to emulate the Delta Delta cepstra transform.

For our telephony system we compared the performance of purely online causal adaptation of the feature transform \( \mathbf{A} \) plus bias \( \beta \), with or without a prior, and the normalized cepstra transform \( \mathbf{R} \) plus bias \( \beta \). All tests were done on a data base of 7753 telephony commands spoken in a noisy car using a cellphone. We looked at the performance of doing online causal adaptation on \( N = 1, 2, 4, 8 \) utterances from the same speaker, by resetting the front-end parameters to the default values every \( N \) utterances.

The results for the telephony system in terms of number of utterance errors for different values of adaptation rate scale \( \beta \) are shown in Table 1.

<table>
<thead>
<tr>
<th>Adapt</th>
<th>Scale</th>
<th># adapt utts before reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Adapt</td>
<td>1</td>
<td>558</td>
</tr>
<tr>
<td>R</td>
<td>0.000005</td>
<td>537</td>
</tr>
<tr>
<td>R</td>
<td>0.00001</td>
<td>533</td>
</tr>
<tr>
<td>R</td>
<td>0.00002</td>
<td>521</td>
</tr>
<tr>
<td>A</td>
<td>0.00004</td>
<td>527</td>
</tr>
<tr>
<td>A+prior</td>
<td>0.00004</td>
<td>529</td>
</tr>
</tbody>
</table>

Table 1: Telephony: Sent Errors (out of 7753) for adapting normalized cepstra and feature (with/without prior) transform

On the embedded system, we evaluated online adaptation for our in car noisy test set, consisting of 73K utterances in low/medium noise conditions (0,30,60 mph windows up, no ac/radio/wipers) and 99K utterances in high noise conditions. (e.g. ac on with windows down at 60 mph). The tasks consisted of saying addresses, commands for control of the radio and digit strings.

In addition to comparing adaptation on \( \mathbf{A} \) and \( \mathbf{R} \) we also looked at the effect of varying the number of iterations in the stochastic update and the effect of varying the alignment delay and block size \( B \). The results for adaptation on a single utterance are shown in Table 2.

<table>
<thead>
<tr>
<th>Adapt</th>
<th>Block Size</th>
<th>Delay</th>
<th>Num iter</th>
<th>low/med noise</th>
<th>high noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Adapt</td>
<td>10</td>
<td>0</td>
<td>5</td>
<td>2.39</td>
<td>6.14</td>
</tr>
<tr>
<td>R</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>2.32</td>
<td>4.61</td>
</tr>
<tr>
<td>R</td>
<td>10</td>
<td>10</td>
<td>4</td>
<td>2.30</td>
<td>4.64</td>
</tr>
<tr>
<td>R</td>
<td>10</td>
<td>10</td>
<td>3</td>
<td>2.28</td>
<td>4.68</td>
</tr>
<tr>
<td>R</td>
<td>10</td>
<td>10</td>
<td>2</td>
<td>2.34</td>
<td>5.04</td>
</tr>
<tr>
<td>R</td>
<td>10</td>
<td>20</td>
<td>2</td>
<td>2.31</td>
<td>5.10</td>
</tr>
<tr>
<td>R</td>
<td>20</td>
<td>10</td>
<td>2</td>
<td>2.33</td>
<td>5.35</td>
</tr>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>2</td>
<td>2.35</td>
<td>5.98</td>
</tr>
</tbody>
</table>

Table 2: Embedded System: Comparison of WER for adapting different parameters in the front-end chain on a single utterance

9. Conclusions

We developed a framework for doing continuous rapid adaptation of any front-end parameter using a low cost stochastic gradient descent algorithm. Using this we showed that on just one utterance with purely causal adaptation we could achieve a 5-15% reduction in word error rate. The result held across a variety of systems and tasks, especially for high noise conditions. We also demonstrated that using a prior on the parameter being adapted, improves performance when test data is sparse. The framework developed is general and in the future we intend to apply it to the non-linear transforms in the front-end chain.

10. References