Optimisation du graphe de décodage d'un système de reconnaissance vocale par apprentissage discriminant

Shiuan Sung Lin

TELECOM PARIS
école nationale supérieure des télécommunications

École Doctorale d'Informatique, Télécommunications et Électronique de Paris

Thèse

présentée pour obtenir le grade de Docteur
de l'École Nationale Supérieure des Télécommunications

Spécialité : **Signal et Images**

Shiuan-Sung LIN

Optimisation du graphe de décodage d'un système de reconnaissance vocale par apprentissage discriminant

Soutenue le 5 juin 2007 devant le jury composé de :

Jean-Paul Haton	Président
Paul Deléglise	Rapporteurs
Kamel Smaïli	
Claire Waast	Examinateurs
Guillaume Gravier	
Gérard Chollet	Co-directeur de thèse
François Yvon	Directeur de thèse

Quand deux chemins s'ouvrent à toi, choisis toujours le plus difficile.

« Himalaya, l'enfance d'un chef »

Remerciements

J'exprime ma profonde reconnaissance à Monsieur le Professeur François Yvon, pour son appui, les discussions enrichissantes et conviviales, le temps qu'il m'a consacré, mais aussi pour les responsabilités qu'il m'a données, je l'assure de ma profonde reconnaissance.

Je tiens à remercier tout particulièrement Monsieur le Professeur Gérard Chollet pour m'avoir accueilli dans son équipe de recherche et encadré durant ma thèse.

J'aimerais ensuite remercier tous les membres du jury: Monsieur Jean-Paul Haton, Monsieur Paul Deléglise, Monsieur Kamel Smaïli, Madame Claire Waast et Monsieur Guillaume Gravier, pour leurs remarques et leurs conseils sientifiques.

D'autre part, je remercie Hemant Misra, pour ses précieux conseils de rédaction, qui ont permis l'accomplissement de ce travail dans une ambiance sympathique et chaleureuse.

Je remercie également mes ami(e)s Eduardo Sanchez-Soto, Amit Kumar, Stéphane Renouard, Djamel Mostefa, Brice Donval et Leila Zouari, pour leur aide et leurs encouragements.

Enfin, je tiens à remercier mes parents et Jamie, pour leur soutien tout au long de mes études en France.

Contents

List of Figures

List of Tables

Résumé

Les trois principales sources de connaissance utilisées en reconnaissance automatique de la parole (*Automatic Speech Recognition*, ASR), à savoir les modèles acoustiques, les dictionnaires de prononciation et les modèles de langage sont habituellement conçues et optimisées de manière séparée. Le travail présenté dans ce rapport vise à proposer une méthodologie pour optimiser *de manière conjointe* les paramètres de ces différents modèles. Cette optimisation est réalisée en intégrant ces ressources dans un unique transducteur fini, dont les poids des transitions sont optimisés par apprentissage discriminant. Après avoir montré la faisabilité de cette approche sur une tâche de reconnaissance « petit vocabulaire » (voir les résultats présentés dans (Lin and Yvon, 2005)), nous étendons ici ce cadre méthodologique à une tâche à grand vocabulaire, la transcription automatique de dépêches d'information radio-diffusées. En particulier, nous proposons plusieurs techniques pour accélérer l'étape de décodage de la parole, ce qui permet de mettre en pratique ces techniques d'apprentissage. Nos expériences montrent qu'une réduction de 1% absolu de taux d'erreur-mot (*Word Error Rate*, WER) peut être obtenue. Nous concluons ce travail par une évaluation critique de l'apport de cette approche sur les tâches de reconnaissance vocale à grand vocabulaire.

1. Introduction

Une large part de l'effort de recherche ASR se concentre sur l'amélioration de la performance d'un composant spécifique du système, avec l'espoir qu'il en découlera une amélioration de la performance totale. Cette démarche ignore les multiples dépendances qui existent entre les diverses sources de connaissance impliquées dans un système de reconnaissance. Par exemple, le design et l'estimation des modèles acoustiques dépendent de la finesse des variantes de prononciation qui se trouvent effectivement dans le dictionnaire de reconnaissance: un petit dictionnaire peut supporter les modèles simples, mais un système à grand vocabulaire nécessitera des modèles plus complexes. De même, les modèles de langage sont estimés séparément des autres sources, en utilisant des corpus tirés parfois de domaines et/ou de registres différents, et toujours beaucoup plus volumineux que ceux disponibles pour estimer les modèles acoustiques. De surcroît, la plupart des approches de modélisation effectuent séparément l'estimation des paramètres de chaque ressource, en supposant que tous les autres paramètres sont fixés. La dépendance entre les différentes ressources est donc ignorée, résultant en des performances sous-optimales pour chacun des modules.

Disposer de procédures d'estimation fiables pour les paramètres des divers modèles impliqués dans le système reste donc une question clé pour obtenir de bonnes performances. Dans la littérature, la stratégie d'évaluation la plus généralement utilisée est l'estimation par optimisation d'un critère de maximum de vraisemblance (*Maximum Likelihood Estimation*, MLE). Cette approche conduit à estimer les paramètres d'une façon telle que la vraisemblance des observations soit rendue maximale. Les principes de MLE reposent sur la disponibilité de grands corpus d'apprentissage. Toutefois, l'amélioration des estimateurs sur un corpus d'apprentissage ne garantit pas que de meilleures performances seront obtenues au moment du décodage (Chen *et al.*, 2000). Cette constatation a mené des chercheurs à explorer d'autres techniques d'évaluation, notamment des techniques d'apprentissage discriminant (*Discriminative Training*, DT). Contrairement à MLE, les méthodes d'apprentissage discriminant visent à trouver des valeurs pour les paramètres qui optimisent la séparation entre les bonnes et mauvaises hypothèses de reconnaissance, et qui minimisent plus directement le taux d'erreur du système. Ces approches reposent sur la formulation d'une fonction objectif qui, d'une certaine manière, pénalise les paramètres qui conduisent à des confusions entre des mots corrects et incorrects.

Ces dernières années, diverses méthodes d'apprentissage discriminant telles que *Maximum Mutual Information* (MMI) (Bahl *et al.*, 1983) et *Minimum Phone Error* (MPE) (Povey and Woodland, 2002), ont permis d'améliorer, parfois de manière considérable, les performances des modèles acoustiques utilisés en reconnaissance vocale. Des techniques d'apprentissage discriminant ont également permis d'obtenir des améliorations significatives en modélisation du langage, s'appuyant sur des techniques d'optimisation telles que le *Minimum Classification Error* (MCE) (Chen *et al.*, 2000), *Minimum Sample Risk* (MSR) (Gao *et al.*, 2005), ou encore des techniques de reclassement (*reranking*) basées sur des variantes l'algorithme du perceptron (Roark *et al.*, 2004).

Les avancées récentes des systèmes d'ASR ont également mis en évidence l'utilité du formalisme des transducteurs finis pondérés (*Weighted Finite-State Transducers*, WFST), qui fournissent un formalisme commun pour représenter de façon homogène les diverses sources de connaissance utilisées en reconnaissance vocale (Mohri, 1997). Un WFST est une machine à nombre d'états fini, dont les transitions sont étiquettées par des symboles d'entrée, des symboles de sortie et des poids arbitraires. Une séquence de transitions de l'état initial à l'état final d'un WFST représente une relation pondérée entre un mot de l'alphabet d'entrée et un mot de l'alphabet de sortie. En utilisant des

transducteurs finis pour représenter les divers composants d'un système de reconnaissance, des ressources différentes peuvent être facilement intégrées dans un espace de recherche doté d'une structure simple. Divers algorithmes permettant d'optimiser cet espace de recherche, consistant, par exemple, à déterminiser[1] et à minimiser le transducteur, peuvent alors être appliqués hors-ligne, avant le décodage. Ces techniques d'optimisation éliminent les arcs et les états redondants, conduisant à un espace de recherche équivalent, mais qui pourra être exploré de façon plus efficace.

Ce travail de thèse vise à combiner ces deux techniques dans un cadre d'apprentissage unifié : les WFSTs sont employés pour représenter de manière homogène les diverses sources de connaissance dans un transducteur pondéré dont les paramètres sont estimés par apprentissage discriminant. Des résultats préliminaires, portant sur une tâche de reconnaissance « petit vocabulaire », ont été présentés dans (Lin and Yvon, 2005). Ces résultats positifs sur la reconnaissance de noms propres ont été récemment confirmés par (Kuo *et al.*, 2007), qui rend compte d'une réduction d'erreur significative sur une application de reconnaissance « grand vocabulaire ». Dans ce mémoire, nous présentons de manière complète le cadre d'apprentissage proposé, nous décrivons les adaptations algorithmiques rendues nécessaire par le traitement de gros volumes de données et nous détaillons et analysons les performances de notre système sur une tâche de reconnaissance d'émissions radio-diffusées. Nous discutons finalement le potentiel et les limites de cette approche, avant d'évoquer les diverses perspectives qu'elle s'ouvre.

2. Apprentissage discriminant des paramètres du graphe de décodage

2.1 Le critère MCE

Soit *G* est un graphe de décodage à états finis, intégrant les différentes ressources nécessaires à la mise en œuvre d'un système de reconnaissance vocale. *G*

[1] La question de la déterminisation d'un transducteur fini est une question complexe. Il existe plusieurs manières d'envisager le déterminisme de ces machines (déterminisme au sens des automates sous-jacents, déterminisme de l'entrée, etc) (voir (Roche and Schabes, 1997)). Il n'est par ailleurs pas toujours possible d'opérer une telle opération de manière exacte. Nous entendons donc ici 'déterminisation' en un sens assez lâche, désignant l'ensemble des techniques exactes ou heuristiques qui permettent d'éliminer certains chemins redondants dans le graphe d'état.

contient deux types de paramètres: les poids des transitions entre états et les paramètres des modèles acoustiques des densités Gaussiennes associés aux états émetteurs du graphe. Étant donnée une séquence de mots *W*, un ensemble de modèles acoustiques Λ, un ensemble de poids entre transitions Γ et une séquence d'observations acoustiques *X*, la log-probabilité conditionnelle de *X* sachant *W* est approchée par le score du meilleur chemin dans *G* pour l'entrée *X* et la sortie *W*. Ce score comprend les log-vraisemblances acoustiques et les poids de transition sur le chemin d'accès (décodé):

$$g(X,W,\Lambda,\Gamma) = a(X,W,\Lambda) + b(W,\Gamma) \tag{1}$$

où $a(X,W,\Lambda)$ est la somme des log-vraisemblances acoustiques et $b(W,\Gamma)$ est la somme des poids de transition sur le chemin de l'état initial à l'état final du graphe. Le décodage de la parole consiste à trouver une hypothèse W_{hyp} correspondant à la séquence de mots qui maximise *g* parmi toutes les séquences de mots possibles.

Si l'on note W_{ref} la séquence de mot correcte, la performance du système de reconnaissance peut être exprimée en fonction de la différence de score entre la référence et la meilleure hypothèse. Pour un vecteur acoustique d'entrée donné, la fonction de misclassification est alors définie par:

$$d(X,\Lambda,\Gamma) = -g(X,W_{ref},\Lambda,\Gamma) + g(X,W_{hyp},\Lambda,\Gamma) \tag{2}$$

Une hypothèse incorrecte se traduit simplement par valeur positive de $d(X,\Lambda,\Gamma)$, signifiant que la séquence de mots correcte n'est pas celle qui obtient le meilleur score au sens de la fonction *g*. En intégrant $\ell(X,\Lambda,\Gamma)$ dans une fonction de perte continûment différentiable on dérive $d(X,\Lambda,\Gamma)$ par:

$$\ell(d(X,\Lambda,\Gamma)) = \frac{1}{1+\exp(-\gamma d(X,\Lambda,\Gamma)+\theta)} \tag{3}$$

où γ et θ sont des paramètres qui contrôlent respectivement la pente et le facteur de décalage de la fonction sigmoïde. En utilisant l'algorithme *Generalized Probablistic Descent* (GPD), une procédure itérative standard peut être définie, basée sur la règle suivante de mise à jour des paramètres pour les poids de transition:

$$\Gamma_{t+1} = \Gamma_t - \varepsilon \nabla \ell(X,\Lambda,\Gamma_t) \tag{4}$$

En supposant que les paramètres des modèles acoustiques sont fixés et ne sont

pas concernés par l'apprentissage, la fonction de perte ne doit être différenciée que par rapport aux poids de transition. Le gradient de (4) vaut alors:

$$\nabla \ell_i(X,\Lambda,\Gamma_t) = \frac{\partial \ell_i}{\partial d_i} \frac{\partial d_i(X,\Lambda,\Gamma_t)}{\partial \Gamma} \tag{5}$$

Le calcul de (5) fait finalement apparaître les deux termes suivants (6) et (7):

$$\frac{\partial l_i}{\partial d_i} = \gamma \ell(d_i)(1-\ell(d_i)) \tag{6}$$

$$\frac{\partial d_i(X,\Lambda,\Gamma_t)}{\partial \Gamma} = -I(W_{ref},s) + I(W_{hyp},s) \tag{7}$$

où $I(W,s)$ représente le nombre d'occurrence de la transition *s* sur le meilleur chemin de décodage pour *W*.

2.2 La règle de mise à jour de paramètres

La mise en œuvre du cadre d'apprentissage MCE implique de régler divers paramètres, en particulier les paramètres γ et ε qui représentent respectivement la pente de la fonction sigmoïde qui transfère la mesure de mis-classification dans l'intervalle [0,1], et un terme qui contrôle l'amplitude de la mise à jour les paramètres. Elle implique également, pour calculer le terme (7), de pouvoir disposer de la séquence complète d'états parcourue à la fois pour la meilleure hypothèse de reconnaissance, qui doit être mémorisée pendant le décodage, mais aussi pour la séquence de mots de référence. Il est donc nécessaire de récupérer, par alignement, le meilleur chemin complet (c.-à-d. au niveau de phone) pour la référence. Enfin, mettre en application la règle de mise à jour (4) exige de résoudre deux problèmes: 1) le choix des *n*-grams qui sont comparés dans la séquence hypothèse et dans la référence et 2) la position exacte de la transition qui sera affectée par la mise à jour des poids de transition.

Puisqu'un mot décodé correctement peut suivre un mot incorrect, signifiant que la mise à jour des paramètres se basera sur une « histoire » de mot incorrecte, nous nous concentrons sur la relation entre deux mots consécutifs plutôt qu'une certaine histoire de mot. Diverses expériences nous ont permis de réaliser que la sélection de l'arc qui est mis à jour a une grande influence sur la distribution globale des poids de transition. Dans notre implémentation, l'arc dont le poids est mis à jour est choisi aléatoirement avec probabilité uniforme parmi tous les arcs candidats possibles.

Enfin, le choix de la valeur γ détermine l'effet de la différence de score entre référence et hypothèse sur la valeur de l'ajustement des paramètres. Diverses expériences nous ont permis de montrer que le choix de $\gamma = 0.02$ est un choix raisonnable. L'amplitude de la mise à jour ε est déterminée dynamiquement pour chaque échantillon d'apprentissage, en employant une technique de *line-search*, afin de donner une mise à jour efficace des paramètres tout en préservant la stabilité de la convergence.

2.3 Implémentation

Le graphe de décodage est construit en compilant les différentes ressources impliquées dans le système de reconnaissance vocale, c'est-à-dire le dictionnaire, les modèles acoustiques et le modèle de language, sous la forme de transducteurs finis pondérés (*Weighted Finite-State Transducers*, WFST). En utilisant des algorithmes standard de construction et de manipulation de transducteurs finis, en particulier l'opération de composition, il est possible de combiner des représentations de niveaux différents[2] dans un graphe simple. La déterminisation est employée pour réduire la taille du graphe en éliminant les chemins redondants, c.-à-d. ceux qui conduisent aux mêmes séquences d'entrée et de sortie ; le graphe ainsi produit est équivalent au graphe original mais plus efficace. Un silence facultatif (modèle court de pause) est ajouté aux frontières de mot; finalement, une pénalité d'insertion de mot (Word Insertion Penalty, WIP) est ajoutée pour équilibrer la longueur des enchaînements de mot sortie.

Notre décodeur effectue la recherche de la meilleure hypothèse en employant l'algorithme de passage de tokens (*token passing*, (Young *et al.*, 1989)) dans un graphe fini. Pour accélérer la propagation des tokens sur les ε-transitions, la relation de ε-clôture entre les états du graphe est pré-calculée et stockée dans une table. Cette table est construite hors-ligne en traversant le graphe et en enregistrant les séquences de ε-transition. Pendant le décodage, cette table est utilisée pour propager les tokens depuis un état source vers tous ses successeurs (directs ou indirects), sans qu'il soit nécessaire de reparcourir le graphe.

Nous l'avons noté précédemment, notre procédure d'apprentissage nécessite une phase préalable d'alignement. Le problème de l'alignement est différent de celui décodage : il s'agit de trouver dans le graphe tous les chemins qui peuvent donner lieux à la production d'une séquence de mots W donnée.

[2] C'est-à-dire employant des alphabets d'entrées/sorties différents : transduction de séquence d'états de HMMs en séquences de phones contextuels, de séquences de phones contextuels en séquences de phonèmes, etc.

Conceptuellement, ceci revient à calculer l'intersection du langage de sortie du WFST avec *W*. Notre approche utilise le principe général de l'inversion de transducteur, opération qui consiste à échanger les symboles d'entrée et de sortie sur chaque transition. En recherchant *W* dans ce graphe inversé, les séquences phonétiques correspondantes peuvent être ainsi être calculées à l'avance : elles correspondent au langage de sortie associé à l'entrée W dans le graphe inversé. Les graphes ainsi extraits se composent généralement de quelques centaines d'états et d'arcs, et dépendent peu de la taille du graphe original.

Il est possible de distribuer les poids des transitions d'un WFST sans en changer la sémantique, c'est-à-dire sans changer la valuation globale des relations définies par le transducteur. Dans les systèmes d'ASR, la distribution locale de ces transitions est en revanche cruciale, dans la mesure où elle a un effet très direct sur l'efficacité de l'élagage qui est opéré pendant la recherche. De nombreux travaux ont ainsi mis en évidence le fait que si la probabilité est correctement redistribuée, l'efficacité de l'élagage et la vitesse de décodage peuvent être sensiblement améliorées (Mohri and Riley, 2001). Nous avons développé un algorithme original de redistribution des poids le long des transitions du graphe d'état. Cet algorithme est conceptuellement semblable à l'algorithme dit "*tropical semiring*" présenté par (Mohri, 1997) et fonctionne comme suit : le graphe est tout d'abord renversé (c.-à-d. que l'orientation de toutes les transitions est renversée) ; pour tous les états, le poids de la transition de poids maximum parmi les arcs entrants est « poussé » vers la frontière de mot la plus proche; finalement le graphe est de nouveau renversé. Cette stratégie est appliquée tout en prenant garde que des poids ne deviennent trop petits, ce qui pénaliserait trop fortement la transition correspondante. Cet algorithme a une complexité en temps comparable à l'algorithme de (Mohri and Riley, 2001), mais est plus efficace en d'espace, car il possède une complexité seulement linéaire en le nombre d'états.

3. Expériences

3.1 Protocole expérimental

Nos expériences sont effectuées sur la base de données développée à l'occasion de la campagne d'évaluation ESTER (Gravier *et al.*, 2004). Cette base de données contient des enregistrements de journaux d'information émis sur plusieurs radios francophones. Approximativement 63 heures de données manuellement

transcrites (constituant le TRAINSet) ont été employées pour estimer les paramètres des modèles acoustiques et d'un modèle de langage 3-gram (LM). L'ensemble de développement (DEVSet) et de test (TESTSet) contiennent respectivement 5.3[3] et 2.95[4] heures d'enregistrement.

Nos modèles acoustiques contiennent initialement un ensemble de 21466 HMMs, correspondant à des phones contextuels, auxquels s'ajoutent 1369 nouveaux modèles que nous avons été conduits à synthétiser lors de la procédure de construction du graphe de décodage. Chaque modèle se compose de 3 états, à l'exception du modèle de pause courte, qui représente un court silence entre mots, et ne comprend donc qu'un seul état. Cet ensemble de modèles contient un total de 6238 états distincts, chacun étant associé à une fonction de densité de probabilité à 39-dimensions, qui prend la forme d'un mélange de 32 Gaussiennes, (la matrice de covariance est supposée diagonale).

Deux modèles de langage 3-gram sont utilisés pour construire les graphes de décodage: le premier est le modèle 3-gram mentionné ci-dessus; le second est modèle 3-gram beaucoup plus grand obtenu par l'interpolation linéaire de plusieurs modèles estimés sur des archives du journal *LeMonde*, couvrant approximativement 400 Millions d'occurrences de mots. Ces deux modèles de langage sont bâtis sur des vocabulaires de 65~000 mots. Les graphes résultant contiennent respectivement 824,845 états pour 1,655,723 arcs et 6,997,044 états pour 19,676,349 arcs. Divers paramètres supplémentaires jouent un rôle dans la construction du graphe : α est employé pour équilibrer les scores de modèles acoustiques et de langage dans le log-domaine ; une pénalité d'insertion de mots (WIP) permet de réguler le « débit » du décodeur. En employant des valeurs empiriquement déterminées pour α, γ et WIP, le taux d'erreur de mots (WER) du système de base (avant apprentissage) pour chaque graphe de décodage est donné dans la table suivante:

LM	2-gram	3-gram	Perplexité	Baseline
Graph1	248739	102461	158.92	45.9
Graph2	5052090	4033834	86.42	37.9

Table 1: Le nombre de n-grams, la perplexité et le WER de baseline de graphe de décodage individuelle.

[3] Toutes les données de développement n'ont pas été utilisées : seules peuvent sont exploitables pour l'apprentissage discriminant les phrases qui ne contiennent aucun mot hors vocabulaire.

[4] Pour limiter les temps nécessaires à la réalisation des expériences, nous nous sommes limités aux données enregistrées sur 3 radios parmi celles figurant dans les radios de test.

Le décodeur utilisé dans nos expériences fonctionne respectivement à 0.7×RT et à 3×RT sur *Graph1* et *Graph2* sur un processeur 3.6Ghz Xeon[5]. Les procédures d'alignement arrivent à 0.05×RT. Des décodeurs plus rapides ont été décrits dans la littérature (voir par exemple (Saon *et al.*, 2005)), mais ces décodeurs ont sur le nôtre l'avantage de ne devoir sauvegarder et mettre à jour que l'historique complet des *hypothèses de mots*. Dans ce cas, des informations cruciales pour l'apprentissage discriminant, en particulier la séquence complète des transitions sur le long du chemin correspondant à la meilleure hypothèse de décodage, ne sont pas sauvegardées durant la recherche. Notre décodeur doit stocker et mettre à jour l'historique *complet* des quelques 20~000 hypothèses qui sont à chaque instant évaluées pendant la recherche, entraînant un surcroît d'activité très significatif: ceci explique pourquoi notre décodeur ne parvient pas à effectuer une recherche en temps réel sur de très grands graphes.

3.2 Résultats

Sélection déterministe ou aléatoire des transitions mises à jour

Dans notre cadre d'apprentissage, les hypothèses de décodage et la référence sont représentées par les séquences de phones ayant donné lieu à des séquences de mots ; en revanche, la comparaison de l'hypothèse et de la référence, qui est sous-jacente au calcul du WER et détermine donc les mises à jour des paramètres se fait au niveau des mots. Le critère MCE ne prescrit pas avec précision la transition qui doit être mise à jour quand une erreur est détectée. Ceci nous laisse plusieurs options : 1) distribuer la mise à jour tous les poids sur le long des chemins partiels qui correspondent à une erreur de décodage, ou bien 2) choisir (de manière déterministe ou aléatoire) un unique arc qui sera mis à jour. Enfin se pose la question de la mise à jour des transitions correspondant à des arcs de repli (*back-off*) dans le modèle de langage : mettre à jour ces arcs entraîne des modifications globales dans le graphe, puisque tous les chemins qui vont utiliser ces transitions sont en fait affectés par la mise à jour.

Notre implémentation traite des transitions de repli comme des transitions régulières et considère que l'ajustement des paramètres du graphe ne porte que sur un seul des arcs des chemins correspondant à des erreurs. Diverses expériences ont été effectuées en utilisant *Graph1*, à partir de la baseline (WER=35) sur le DEVSet. Deux stratégies de mise à jour déterministe (mise à

[5] Les chiffres sont obtenus en pré-calculant hors-ligne les vraisemblances acoustiques. Nous estimons qu'intégrer les calculs de ces vraisemblances augmenterait le temps d'exécution par moins de 0.7×RT pour le décodage et par approximativement 0.1×RT pour l'alignement.

jour sur le premier arc disponible et mise à jour sur le dernier arc) sont comparées avec une stratégie aléatoire (voir la Figure 1). La stratégie de mise à jour aléatoire converge non seulement plus rapidement, mais atteint également un WER inférieur, confirmant nos résultats présentés dans (Lin and Yvon, 2005). Cette stratégie est employée dans toutes les expériences suivantes.

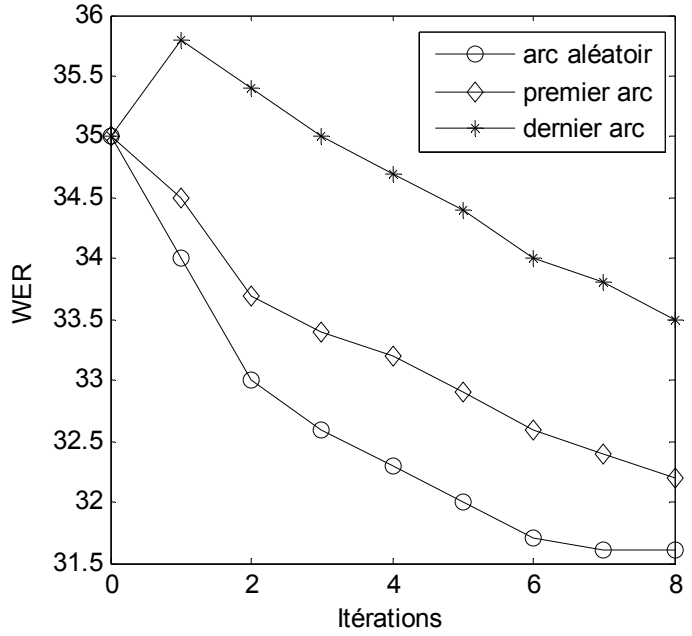

Figure 1: Performances du système en fonction du positionnement de la mise à jour.

Réduction des erreurs sur un grand graphe

Un graphe de décodage contient un grand nombre de paramètres. Certains des paramètres, notamment les poids des transitions, ne sont pas entièrement indépendants. Ceci a pour effet de rendre difficile la convergence du processus d'apprentissage vers le taux d'erreur minimum.

Cependant, un grand graphe fournit habituellement une meilleure baseline pour le DT. Par conséquent, étant donné la même quantité de données d'apprentissage, nous comparons deux graphes de taille différente: le DT est effectué en utilisant DEVSet sur *Graph1* et *Graph2*. TESTSet est employé pour évaluer l'exécution d'un graphe de décodage après l'ajustement de paramètre. Après 8 itérations, nous obtenons les résultats présentés dans la Table 2.

L'amélioration sur le DEVSet est semblable pour les deux graphes, les performances de *Graph2* fournissant une baseline bien meilleure. Sur le TESTSet, la réduction de WER est plus petite (-1% absolu), dont la moitié est obtenue après la 1ère itération.

	DEVSet	TESTSet
Graph1	35→31.6 (-3.4)	45.9→44.9 (-1)
Graph2	28.5→25.1 (-3.4)	37.9→37 (-0.9)

Table 2: La réduction de WER sur le DEVSet et sur le TESTSet, en utilisant des graphes de taille différente.

Dans des applications pratiques, un graphe peut être construit à partir d'un modèle de langage contenant un grand nombre de *n*-gram, alors que le nombre de paires de mots dans l'ensemble de développement qui est employé pour optimiser le graphe est comparativement beaucoup plus petit. Ceci signifie que le DT met à jour seulement une petite fraction des poids de transition. Dans la section suivante, afin de mieux évaluer le potentiel de cette approche, nous employons une plus grande base de donnée pour l'apprentissage discriminant.

Apprentissage avec une grande base de donnée

Ces expériences sont effectuées en utilisant *Graph2* comme pointe de départ. Une seule itération d'apprentissage est effectuée en utilisant l'intégralité des 62 heures de TRAINSet pour le DT. Cet ensemble de données contient approximativement 11 fois plus de paires de mots que DEVSet, et 7 fois plus de paires de mots différents. Les résultats sont présentés dans la Table 3. En utilisant plus de données, les résultats de décodage sur le TESTSet donnent une amélioration de 1.1 points du WER dès la 1ère itération, approximativement 3 fois plus que ce que l'on obtient en utilisant le seul DEVSet.

	Sub.	Del.	Ins.	WER
DEVSet	23.1	12.0	2.4	37.5
TRAINSet	22.8	11.2	2.8	36.8

Table 3: La réduction de WER en utilisant les données différentes.

4. Discussion et améliorations

Nos expériences ont prouvé que les méthodes d'apprentissage discriminant ont le potentiel pour réduire sensiblement le WER, même pour les tâches à grand vocabulaire. Un premier souci est la stabilité de notre procédure itérative d'optimisation: un examen plus approfondi des résultats de l'apprentissage indique que le nombre de mises à jour est graduellement réduit sur les deux graphes. Notons toutefois que même après huit itérations, le nombre de mises à jour de paramètre opérées pendant une itération reste substantiel (de l'ordre de

quelques dizaines de milliers), suggérant que les poids de transition ne stabilisent pas complètement.

Nous avons émis l'hypothèse que le DT réduit les erreurs d'apprentissage en redistribuant les poids des transitions, rapprochant les séquences de sortie des séquences de référence, et diminuant les différences de scores entre la référence et la meilleure hypothèse. Ceci est confirmé par l'analyse de la différence de scores entre la référence et la meilleure hypothèse du DEVSet : au terme de 6 itérations, la différence des scores a été diminuée pour presque tous les échantillons d'apprentissage; et 217 échantillons incorrects à l'origine (3.34% des phrases) sont complètement corrigés (la référence et l'hypothèse sont entièrement identiques).

Les résultats sur le TESTSet sont également positifs, quoique par une plus petite marge, suggérant que notre algorithme ne généralise pas bien. En fait, une étude des paires de mot qui se produisent dans le TESTSet et dans le DEVSet indique qu'environ 40% des mots du TESTSet figurent également dans le DEVSet. La situation est en fait bien pire: même si une mise à jour est opérée sur la transition $u \rightarrow v$ pendant l'apprentissage, il se peut que cette mise à jour ne soit pas exploitée pendant le test à cause de la différence dans l'historique du modèle de langage (par exemple si *wuv* apparaît dans les données d'apprentissage et *w'uv* dans les données de test).

Au final, environ 2.5% seulement des paires de mots dans l'ensemble de test sont touchées pendant l'apprentissage, montrant clairement le fait que notre procédure de mise à jour des paramètres ne peut fournir qu'une amélioration limitée sur cet ensemble de données. L'utilisation de plus de données d'apprentissage semble un remède efficace, quoique computationnellement coûteux, à cette situation. D'autres améliorations de notre procédure d'apprentissage semblent vraiment nécessaires poru rendre les mises à jour de paramètre moins locales: c'est en fait ce qui se produit avec des procédures traditionnelles d'estimation de LM: n'importe quel nouvel exemple d'*uvw* augmentera la probabilité de *w* dans le contexte de l'*uw*; il augmentera également, par une plus petite marge, la probabilité de *w* après *v*. Deux manières de faire la mise à jour moins "locale" de paramètre sont envisagées: un première consisterait à considérer les N-meilleures hypothèses plutôt que juste la meilleure (voir (Kuo *et al.*, 2002)); une deuxième serait de considérer de mettre à jour des arcs sur tous les alignements possibles de la référence, plutôt juste sur le meilleur. Parmi les autres voies d'amélioration, mentionnons également l'ajout d'un facteur de régularisation dans le critère optimisé, qui aurait pour effet de

restreindre le nombre et l'amplitude des mises à jour sur le DEVSet, ou encore la modification de la règle de mise à jour pour essayer de maximiser *la marge* entre la meilleure hypothèse et ses concurrents.

5. Conclusion et perspectives

La mise en œuvre de techniques d'apprentissage discriminant sur de grands graphes de décodage s'est avéré calculatoirement faisable et susceptible d'améliorer les performances du système. Dans ce cadre d'apprentissage, l'optimisation des paramètres est opéré sur un graphe de décodage statique dont les poids des transitions sont ajustés itérativement. Nous avons présenté les résultats d'expériences sur des tâches à grand vocabulaire qui confirment les résultats obtenus sur des systèmes de décodage à l'état de l'art pour l'anglais conversationnel (Kuo *et al.*, 2007) et nous avons discuté les avantages et les limitations de notre stratégie d'apprentissage. Deux autres améliorations sont envisagées: 1) obtenir un contrôle à grain fin de l'effet des mises à jour de paramètre. Sur un WFST « compact », la mise à jour d'un paramètre change le score de tous les chemins qui emploient cette transition, ce qui peut ai final se révéler néfaste à l'exécution globale ; 2) créer (et mettre à jour) de nouvelles transitions autant que nécessaire, afin de réduire au minimum l'effet précédent.

En conclusion, il est enfin important de signaler que nous avons jusqu'ici considéré que les paramètres acoustiques étaient fixés. Pour quelques échantillons d'apprentissage, la disparité acoustique est en fait si mauvaise que la seule mise à jour des poids de transition ne permet pas de « remonter » le score de la référence, ou le ferait au prix de changements déraisonnables des poids. De meilleures stratégies d'optimisation qui combineraient apprentissage acoustique et linguistique sont certainement nécessaires pour mieux faire face à ce genre de situation.

Chapter 1. Introduction

Speech is one of the most natural ways of communication for human beings. The task which extracts the intended message content in the signal is automatic speech recognition (ASR). Since the human speech carries not only the linguistic information but also the personal information such as the emotion and different characteristics of speakers, the high variability of speech signal makes it a great challenge to yield accurate transcripts. In the past four decades, speech recognition has converged as a scientific field in its own right, integrating various techniques and theories from linguistics, statistics, computer science and signal processing. Progress has been steady, yielding the development of systems capable of robust decoding and of dealing with the high variability of speakers and environments.

1.1 The Problem of ASR

ASR systems are generally based on Hidden Markov Models (HMMs) (Baum and Eagon, 1967) and consider the speech signal as a discrete sequence of observations in some high dimensional feature space. Small vocabulary speech recognizers simply decode the input stream of acoustic features, by searching for the most likely state sequence in the HMMs, which directly uncovers the most likely phone and word sequence. The core of large vocabulary systems is based on a similar principle, albeit on a more complex search space design, which involves many HMM acoustic units (typically contextual phones), a large dictionary and a stochastic language model. The dictionary is used to map sequences of phonetic symbols to words, taking into account all the possible pronunciation variants. Language models help discriminate between acoustically confusable words based on their history.

Difficulties in ASR

Although ASR has been implemented in a wide variety of applications, robustness in speech recognition remains difficult to achieve (Junqua and Haton, 1995). Several fundamental issues still remain to be properly addressed:

- Robustness: this covers problems related to the noise conditions, speaking styles, accents, dialects and the acoustic ambiguity.

- Adaptation: adaptation of language or acoustic models aims to improve the performance by adapting an ASR system to new conditions. The target domain may not have the same vocabulary (the out-of-vocabulary problem) or the same speakers. Furthermore, adaptation data may be much sparser than the original training data.
- Speech recognition with limited resources: recent developments in mobile devices have to deal with the fact that the hardware is not as good as that in desktop computers; the memory is limited and CPUs are less powerful.

These topics continue to attract a lot of research, especially the modeling problem. In ASR systems, the modeling problem refers to the attempts to simulate the speech process in a machine; this includes pronunciation modeling, acoustic modeling and language modeling. Although many efforts have been made on the modeling problem (Gao and Zhang, 2002; Nadas *et al.*, 1988; Rosenfeld, 2000; Sakti *et al.*, 2006), several fundamental difficulties still remain to be overcome.

Modeling Problem

(1) Acoustic Modeling

Acoustic models come from the inventory of phonetic units in a language. The task of acoustic modeling consists in modeling the characteristics of the units that are needed by the decoder to determine the spoken labels. Most of the modern ASR systems use HMM based acoustic models, which involve high-dimensional parameters. To produce a reliable set of parameters, the maximum likelihood estimation (MLE) method (Aldrich, 1997; Bahl *et al.*, 1983; Sankar and Lee, 1996) is a popular choice, which has been shown to give better performance when iterative training process is performed on a large training data. However, due to the categorical nature of language, it is difficult to collect adequate data to estimate a large set of parameters. Apart from the problem of lack of data, MLE trained models do not always seem to give the best performance (Chou, 2000; Juang *et al.*, 1997).

(2) Language Modeling

The language model is another essential component of a speech recognition system. It helps the decoder determine the correct word sequence and sort out acoustic confusions. Language modeling is based on a statistical description of language regularities from text corpus. Statistical language modeling (SLM)

based on MLE method is a straightforward and widely used modeling technique in speech recognition (Clarkson and Rosenfeld, 1997; Stolcke, 2002), which attempts to estimate a reliable probability distribution for short word sequences. However, language modeling based on MLE method suffers from the same problems as MLE based acoustic modeling. When the model tries to capture more complex dependencies, more parameters need to be estimated. In practice, even in a large text corpus, the majority of the *n*-grams occur with very low counts, which foster the needs for smoothing techniques (Chen and Goodman, 1996) such as discounting the maximum likelihood estimates (Witten and Bell, 1991) or backing-off (Kneser and Ney, 1995) to lower *n*-grams. The most popular evaluation metric for language models is perplexity. Nevertheless, lower perplexity may not always lead to higher decoding performance (Chen *et al.*, 1998; Iyer *et al.*, 1997; Printz and Olsen, 2000).

Estimation in Isolation

The three main knowledge sources used in the decoding process, namely the acoustic models, a dictionary and a language model, are usually designed and optimized in isolation. For instance, the design of phonetic dictionary is often done by hand or with the help of phonological rules. The phone sequence in the dictionary must match the actual occurrences in the database to yield the best recognition performance. It usually requires an expert to do this labor-intensive work, especially when variants of words need to be added to capture the variability. Many ASR systems give a couple of alternative pronunciations for most words and assume the pronunciations are correct for acoustic modeling.

At present, most of the research focuses on improving the training of a certain source, with the hope that it will improve the overall system performance. This does not consider the dependency between knowledge sources. For instance, parameter estimation of acoustic models depends critically on the definition of word pronunciation(s) in the dictionary. Language modeling is performed separately from other resources using different, and often much larger, corpora. These modeling approaches perform the estimation in isolation, assuming that parameters of other sources are reliable. The influence among sources is ignored, which makes it hard to reach the best performance over target data.

1.2 Objective of the Thesis

Among the many studies on ASR, this thesis focuses on the modeling problem in speech recognition. Parameter estimation in traditional modeling approach is

performed in isolation and the optimization process may not yield an optimal decoding performance. Therefore, we propose a novel training framework that combines various knowledge sources into an integrated decoding graph and apply discriminative training to update the graph parameters. In this unified training framework, we attempt to directly optimize the performance of a decoding graph, rather than indirect measures of the performance such as the likelihood.

1.3 Contribution of the Thesis

Our contribution is not only academic but also practical.

Integrated Decoding

Although the concept of performing graph search was first implemented in the HARPY system in 1970s (Lowerre, 1976), we further extend the concept by using weighted finite-state transducer (WFST) to integrate various sources into a single decoding network (Mohri *et al.*, 1996; Mohri and Riley, 1999). The WFST technique is a flexible and efficient representation of knowledge sources. Moreover, various optimization algorithms can be applied off-line prior to decoding, which improves the decoding speed and reduces the memory requirements.

Discriminative Training

Discriminative training has been shown to outperform conventional MLE method in recent studies both in acoustic modeling (Povey and Woodland, 2002; Sandness and Hetherington, 2000) and language modeling (Chen *et al.*, 2000; Kuo *et al.*, 2002). Our novel training framework applies this training approach to update graph parameters. We also propose a parameter update rule to produce sufficient parameter adjustment. Several similar studies introduce WFST technique and use discriminative training to perform parameter update (Le Roux and McDermott, 2005; McDermott *et al.*, 2007). However, their search space design basically represents the recognition lattice in the form of a WFST, which is not an integrated decoding graph. In this work, an additional search network is required to perform discriminative training.

Fast Decoding Techniques

When decoding is performed on a large and complex graph, a fast decoder is

indispensable. A recent research presents an extremely fast graph decoder that gives sub-real-time decoding speed on a large graph (Saon *et al.*, 2005). However, it is a pure decoder which does not keep enough information of the transition paths for discriminative training. We thus extend their concept and develop several fast-decoding techniques to reduce the overhead of history keeping. Moreover, our sub-graph extraction technique aiming at fast alignment is a general-purpose method. It can be directly used, for instance, to perform acoustic modeling.

1.4 Thesis Organization

This thesis is organized as follows.

We begin with a general overview of typical ASR systems in Chapter 2. The involved knowledge sources and the prevailing estimation techniques are introduced with a critique. Here, we also cover several implementations of search strategy in speech recognition, together with performance issues. Based on the discussions on modeling problems and decoding efficiency, we propose a novel discriminative training framework at the end of this chapter.

Chapter 3 elaborates the details of discriminative training on static decoding graph. At first, we compare discriminative training and conventional MLE from a modeling point of view. We then present the common discriminative training criteria in ASR tasks and introduce an error rate based criterion and the training framework. A further discussion is made on the choice of the loss function, the optimization method and parameter selection. The second part of this chapter deals with the construction of an integrated decoding graph whose design greatly affects the decoding efficiency and accuracy. We describe our graph construction procedure and weight pushing algorithm. In addition, we also present several fast graph decoding techniques that are needed to make discriminative training of practical use.

Chapter 4 presents the experimental results. The database and the experimental setup are presented in detail. Experiments are carried out from various directions to show the improvements of discriminative training on large decoding graph and the performance of different parameter updating schemes. To further investigate the effect of discriminative training, an analysis is made from the reduction of confusion pairs, the number of graph updates, score difference, the coverage of parameters and the comparison of transition paths, where we also point out the potential improvements.

In Chapter 5, the conclusion of the thesis is presented, giving a perspective of discriminative training on decoding graph.

The appendices present the derivation of MCE criterion, an example of graph representation of language model and the published papers.

Chapter 2. Speech Recognition: An Overview

Speech recognition is the process which inputs speech data and converts it into a sequence of words. The goal of speech recognition is to obtain the decoded words as accurately as possible, which requires the ability to find out correct patterns in the speech. Modern automatic speech recognition (ASR) systems implement this process by introducing the concept of pattern-matching. Under this view, recognition process is to produce the most possible pattern in the speech signal.

In the following, we review the history and more recent developments of ASR in Section 2.1 . In Section 2.2, three main knowledge sources and popular modeling approaches are investigated in detail. In Section 2.3, various search strategies in ASR are presented with discussions on the performance issues. Finally, we come up with a novel training framework aiming at producing a reliable set of parameters on an integrated network.

2.1 History and Developments

2.1.1 Bayes Decision Theory

When ASR systems moved from simple pattern-matching to large vocabulary decoding tasks, various statistical methods and learning theories have been proposed, which give a strong foundation to the probabilistic framework. The statistical approaches have their root in Bayes decision theory (Duda and Hart, 1973). The aim of the decision (classification rule) is to assign an object to a class such that the expected loss is minimized (Chien *et al.*, 2006). From this viewpoint, given a word sequence $W = \{w_1, w_2, ..., w_n\}$ and an observation vector X, an intuition of the speech recognition problem can be expressed as:

$$W^* = \arg\max_{W} P(W \mid X) \tag{2.1}$$

where the recognizer's task is to find the most likely word sequence W^* among all possible word sequences. A direct implementation of this idea is inapplicable. By applying Bayes' theorem to the conditional probability, (2.1) can be written as the following:

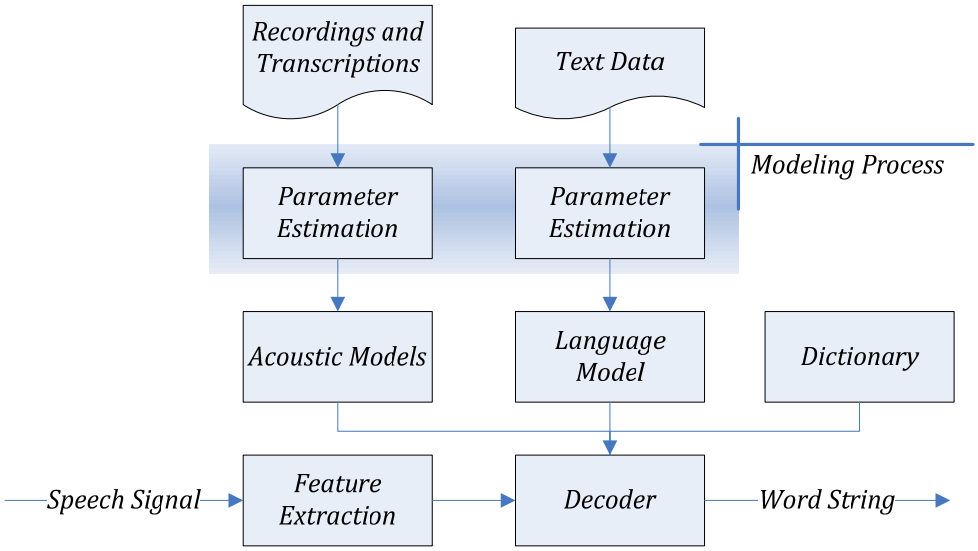

Figure 2.1: Probabilistic speech recognition framework.

$$W^* = \arg\max_{W} \frac{P(X|W)P(W)}{P(X)} = \arg\max_{W} P(X|W)P(W) \qquad (2.2)$$

where $P(X|W)$ and $P(W)$ are respectively factored as acoustic model and language model (Jelinek, 1997).

With the help of Bayes decision theory, speech recognition has been successfully transformed from a pattern-matching task to a statistical decision problem. Figure 2.1 displays the standard speech recognition architecture, and outlines the fact that recognition performance critically depends on the estimation of model parameters. A good estimation procedure should not only produce a reliable set of parameters but also adjust parameters to higher recognition accuracy.

The modeling problem is the key in speech recognition and has attracted a lot of research. During the 1990's, theoretical advances in pattern recognition further transformed the traditional training framework into complex learning problem, and better performance can be achieved by reducing the decoding errors (Juang and Katagiri, 1992; Juang and Rabiner, 2005).

2.1.2 The Statistical Paradigm

From Simple Patterns to Various Knowledge Sources

The first ASR system was developed in the 1930's in Bell Labs at AT&T. The researchers developed a machine for speech analysis and synthesis. Since then,

the development of systems using speech as a medium of interaction has attracted many researchers. Early ASR systems considered speech as a sequence of phonetic elements such that each element corresponds to a pattern representing a digitized articulation of some human sound(s). A speaker dependent speech recognizer was built in the 1950's for isolated digit recognition, and achieved about 50% efficiency (Davis *et al.*, 1952).

In the 1970's, the HARPY system (Lowerre, 1976; Lowerre and Reddy, 1990) was shown to give a reasonable performance by doing recognition on a graph network with some syntactical rules. The search space design, represented by a finite state network, could efficiently reduce the computation. In addition, the syntactical rules helped the recognizer make use of the knowledge not only from template matching of speech parameters but also from the relationships between words in the language. Such systems, which take speech parameters and syntactical rules as input and perform pattern-matching algorithm on a search space to obtain the best hypothesis, already implemented the typical architecture of most modern decoders.

Statistical Modeling of Knowledge Sources

(1) Acoustic Models

One essential improvement in speech recognition systems was obtained when Hidden Markov Models (HMMs) were introduced for acoustic modeling. Although the theory of HMM was proposed in 1960's (Baum and Eagon, 1967), this technique was only extended to mixture densities and widely applied to ASR in the 1980's. Among the various studies, the Baum-Welch algorithm (Baum, 1972) provided an efficient estimation procedure of the model parameters, which only requires speech transcripts. Using complex probability density functions, the speech signal was represented in a statistical form and the system could work in a speaker-independent manner. This means that one can build a system using less data for more speakers. By combining pattern recognition approaches, the use of HMM based methodology has successfully established a statistical framework for the speech recognition problem, and has significantly pushed ASR developments from simple pattern matching to large vocabulary, continuous speech and speaker-independent systems. Till present, the HMM based methodology is the most popular approaches for acoustic modeling.

(2) Language Model

In the last few decades, another major issue in ASR systems has been the design of syntactical rules which are expected to help correct the word sequence(s).

These syntactical rules, when represented in a statistical framework, define a language model. In the 1970's, a speech recognition system that incorporated a statistical language model was developed at IBM (Jelinek *et al.*, 1975). The language model predicts the subsequent word based on the already decoded word sequence. A strong statistical representation of the word relationship could guide the decoder to select the best hypothesis while searching for word strings. In current developments, language models play a critical role and constitute an essential component of ASR systems, especially for large vocabulary tasks. Even if they represent a very crude approximation of the syntax of natural languages, language models are very difficult to improve upon in terms of performance, and hard to be efficiently integrated in the decoding algorithms.

After significant efforts, the research of speech recognition has been established in a statistical paradigm, under which most current research is dedicated to enhance the performance and to compensate the weakness of model parameters estimation, particularly as far as acoustic modeling and language modeling are concerned. The flexibility of probabilistic foundation allows researchers to investigate various methods to better fit the model to the training data.

2.1.3 Search

The search is an indispensable component of ASR systems. Many search techniques have been developed for the decoding problem (Mohri *et al.*, 2000b; Ney and Ortmanns, 1997; Viterbi, 1967). These techniques aim at quickly decoding the message content in the speech signal. In recent years, the search space often incorporates large and complex knowledge sources. A good search technique is not only computationally effective but should also make it possible to output more than just the best scored sequence (for instance, a word lattice and the pronunciations that were used).

One-Pass and Multi-Pass Search Strategies

The search strategies are generally implemented as one-pass or multi-pass search. In the one-pass search (Odell *et al.*, 1994), the decoder attempts to find the most likely hypothesis by searching the whole search space, using all integrated knowledge sources at a time. The multi-pass search tries to reduce to the complexity of search space in the first pass, using a word graph (Ney and Aubert, 1994) or a tree trellis (Soong and Huang, 1991) to represent the candidates for further processing. Less computational cost approaches are applied at other stages.

Dynamic Programming Technique

One commonly known method for the detection of temporal, repeated speech units is known as dynamic time warping (DTW). DTW is based on dynamic programming and is used to compute the best possible alignment between a test and a reference pattern. The sequence with the highest score (or the lowest distance) from the input pattern is the decoded result. Since DTW search allows the patterns to be stretched and compressed, this method gave substantial improvements in early speech recognition systems using simple speech patterns (such as digits), small vocabulary size and limited number of speakers. Since 1970's, numerous variants of DTW were proposed, including the more subtle search method: the Viterbi algorithm.

The Viterbi algorithm (Viterbi, 1967) is a search algorithm based on the principle of dynamic programming. It was proposed by Andrew Viterbi for error correction coding in communications and has been widely used in many other applications. In speech recognition, the algorithm searches the most likely sequence by holding most probable hypotheses and eliminating other hypotheses at each time frame. Viterbi search is a computationally demanding technique. The search is usually performed with approximation criteria such as beam pruning and score threshold to improve the hypotheses. The use of such heuristics makes the search suboptimal, but greatly reduces the computation cost.

A Search*

As the search space is getting more complex and large, several alternative computationally efficient methods have been developed, either to search the decoded result heuristically or to significantly reduce the computation cost. In 1968, the general purpose A* search was developed (Hart *et al*., 1968). It visits from initial node to the terminal in a graph. The partial hypotheses are represented by paths. During the search, it expands new branches to the leaves according to a heuristic estimate so as to expand only the most promising paths as partial hypotheses.

WFST Decoding

In recent years, research on graph representation has opened the possibility to further improve decoding efficiency in terms of computation time and memory space. These improvements advance ASR to continuous and spontaneous

applications that require very large vocabulary, complex language and acoustic models. The use of weighted finite-state transducer (WFST) for graph representation unveils another interesting development in speech recognition (Mohri *et al*., 1996). A weighted transducer is a finite-state machine which carries a) the mapping information of the input and output symbols, and b) the associated weights. The weights can be the costs or probabilities. When knowledge sources in ASR are represented in the form of transducers, more general algorithms such as composition, determinization and other graph manipulation approaches can be applied to yield more efficient decoding, without deteriorating the performance (Pereira *et al*., 1994).

2.1.4 Current Applications

With ever growing CPU speed and memory storage, speech recognition has been implemented in a bunch of applications and has been applied successfully to many languages. The potential of ASR lies in the fact that it will make human machine interaction much easier and natural.

In recent years, when the use of mobile devices such as mobile phone or PDA becomes more popular and the devices get more powerful, large keyboard and mouse would no longer be suitable for such small-size devices. Smart and small input interface must be used to fulfill this need. Typical equipment is a stylus that is used to click the menu on the device and browse the windows, in a manner similar to what users do in desktop computers. However, the use of stylus could be tedious, especially when users are typing a short text, searching a telephone number or a name in the personnel address book. Several products have already integrated speech interface to deal with voice-command applications. Some commercial products are also available for desktop computers. For example, the IBM's ViaVoice [6] is a successful commercial product which provides voice-enabled, conversational access to computer, and free expression for dictation. In addition, the car with speech recognition capability[7], the indexation of broadcast news and the telephone based service are all speech recognition applications. Recently released Windows Vista [8] also integrates speech recognition to make the user interaction with the PC more natural.

[6] IBM ViaVoice, http://www-306.ibm.com/software/voice/viavoice/
[7] Honda cars equipped with Satellite-Linked Navigation System:
http://automobiles.honda.com/
http://domino.watson.ibm.com/comm/pr.nsf/pages/news.20040901_speech.html
[8] Windows Vista, http://www.microsoft.com/windowsvista/default.aspx

2.2 Statistical Models and Resources

2.2.1 Dictionary

The Role of Dictionaries in Speech Recognition

Current paradigm of speech recognition incorporates a dictionary, the acoustic models, and a language model to transcribe human speech. One fundamental issue is the definition of the voice units in the human speech. These units must represent what is actually pronounced and must be able to cover the variants.

In conventional ASR systems, the dictionary contains a list of mappings between a word and the corresponding phonetic units. With these mappings, the words occurring in the transcriptions can be represented as a sequence of phones. The training process of acoustic models starts from these phone sequences and their corresponding recordings to perform parameter estimation. It is thus obvious that the performance of acoustic modeling depends heavily on the dictionary design.

The dictionary is also used during decoding. When a speech signal is decoded into a sequence of phones, these phones may correspond to a set of distinct words which have the same pronunciations (for instance, *abîme*, *abîment* and *abîmes* are all pronounced /a b i m/) and a word may contain a couple of variants. For instance, the word alpes can be pronounced as the following:

/a l p @ z/
/a l p z/
/a l p/

The dictionary is used to translate the phones into the correct word(s).

Issues in Dictionary Design

The word-to-phone mapping in the dictionary must be correct to train reliable models. Besides, the coverage of variants must be included. For recognition with a small vocabulary, such as searching a name or a telephone number in a personal address book, the speech units can be words or phones. Even if variants are taken into consideration, dictionary design remains simple and is easy to implement. However, recognition of conversational speech is much more difficult. People speak in natural way with personal styles and various accents. Topics range from daily life to specific domain. The number of words and variants

increase greatly as compared to what is needed in simple applications. Moreover, the inter-word effects such as coarticulation[9] and word contraction (for instance, *tu as* is pronounced *t'as*) are commonly found in conversational speech. Even if an effort is made to include all the variations and spontaneous effects, the size of the dictionary would become too large to maintain.

Rule-Based and Data-Driven Approach

One approach to handle this problem is to include phonetic variants for words. If this is done by hand, it requires an expert to do such a labor-intensive work. When a lot of new words are to be added, hand tuning is less efficient. An alternative is to introduce phonological rules to the dictionary (Imai *et al*., 1995). However, a few rules may not cover the variants in the data set while too many rules result in too many possibilities.

Another possibility is to learn the pronunciations or variants from data (Bacchiani and Ostendorf, 1999; Schultz and Waibel, 1998; Singh *et al*., 2002; Sloboda and Waibel, 1996). These data-driven approaches attempt to choose the pronunciations of a word that frequently appear in the database rather than the "correct" pronunciations. Due to the high variability of human pronunciations, accurate dictionary design remains a difficult task. Most ASR systems use a fixed set of mappings or rules to represent the words in terms of phones. The set of word-to-phone mappings is used as a dictionary for decoding and for acoustic training.

2.2.2 Acoustic Models

Hidden Markov Model

State-of-the-art speech recognition systems are often based on HMMs which characterize the speech signal as statistical patterns by a double-layered stochastic process (Baum and Eagon, 1967). Each model is a finite set of states and each state is associated with a probability distribution. Transitions connecting states are weighted transitions which carry transition probabilities. According to the nature of probability distribution, HMMs can be discrete or continuous. Most popular approaches introduce continuous HMM (CHMM) to train the models. Formal definition of HMM can be given as a triple (A, B, π),

[9] The beginning of a phone is affected by the preceding phone, and the end is modified by the succeeding one.

respectively representing transition probabilities, output distributions and initial state probabilities. Let N and o_t be the number of states in the model and the observation vectors at time t respectively. An HMM needs to satisfy the following constrains.

- Initial state distribution $\pi = \{\pi_i\}$: the probability at state q_1 when t=1, $\pi_i = P[q_1 = s_i],\ 1 \le i \le N$
- State transition probabilities $A = \{a_{ij}\}$:
  $a_{ij} \ge 0,\ 1 \le i, j \le N$
  The sum of transition probabilities satisfies the stochastic constraint:

  $$\sum_{j=1}^{N} a_{ij} = 1,\ 1 \le i \le N$$

- Probability distribution on a state $B = b_j\{(o_t)\}$: for the continuous density models, the probability of observation o_t at state j is given by assuming a mixture of Gaussian models:

  $$b_j(o_t) = \sum_{m=1}^{M} c_{jm} \mathcal{N}(\mu_{jm}, \Sigma_{jm}, o_t) \tag{2.3}$$

  where M is the number of mixture components in state j and c_{jm} is the mixture weight. The probability distribution is represented by a Gaussian mixture density:

  $$\mathcal{N}(\mu, \Sigma, o) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}(o-\mu)' \Sigma^{-1} (o-\mu)} \tag{2.4}$$

  The μ and Σ are mean vectors and covariance matrix of the mixture component. For the discrete density models, the probability distribution is expressed:

  $$b_j(o_t) = P_j[v(o_t)] \tag{2.5}$$

  where $v(o_t)$ is the vector of output symbols given input vector o_t and $P_j[v]$ is the probability at state j of generating the symbol vector v.

Types of Basic Units

While HMMs have been widely used to train acoustic models from an inventory of phones, the performance of phone sequence decoding is still less than ideal for spontaneous and conversational speech, where human coarticulation effects occur frequently. Individual phone recognition is no longer enough for accurate decoding. Models that do not take into account the coarticulation effects of are called context-independent (CI) models.

To improve decoding accuracy, the coarticulation effect must be handled. One can define various CI models to capture the influence between adjacent sounds, either by phonological rules, data-driven methods or mix of the two. Besides, recent studies have proposed several approaches to span the CI models from the neighboring context (Woodland *et al*., 1995). Models of this type are called context-dependent (CD) models. These CD models can be obtained from the right context (right-context dependent models), the left context (left-context dependent models) or by creating a new model taking the CI model in the middle, given its left and right context, yielding the so-called "tri-phones" that are commonly used in most ASR systems.

To expand CI to CD models, a choice can be made for the CD phones at the word boundary to determine the cross-word or internal-word models to be used. Generally, silence or short pause can be considered as special phones and are not used for context expansion.

A common issue of using CD phones for speech recognition is that a very large number of models need to be estimated. For a phone inventory of size n, the complexity of possible CD phone models would be $O(n^3)$, which makes it difficult to collect adequate training data for each CD phones. Typically, the combination of CD phones is called logical models. Several CD phones may occur rarely and some of them do not exist in human speech, such as some combination of consonants. Popular approach of CD model training is usually performed by state-tying (Young *et al*., 1994) or mixture-tying (Digalakis *et al*., 1996), where the same training samples are shared among different probability distributions. Once the training process is finished, physical models are created, and also a list of mappings from logical models to physical models.

Statistical Acoustic Modeling

An HMM is a probabilistic model of random variables. Under the model, there are three basic problems (Rabiner, 1989):

- Evaluation problem: the calculation of the likelihood of an observation sequence.
- Recognition problem: the issue of searching the most likely state sequence given the observation.
- Estimation problem: the estimation of model parameters to best match the observation.

Statistical acoustic modeling aims to develop the techniques dealing with the estimation problem. Several popular estimation methods are presented below.

(1) Maximum Likelihood Estimation

In the literature, the most commonly used criterion for estimating the parameters of an HMM is maximum-likelihood (ML). This criterion attempts to adjust the means, variances, mixture weights and state transition probabilities such that the posterior probability of a model can be maximized given evidences under the form of a sequence of observations. A popular and computationally efficient algorithm, the Baum-Welch algorithm, is often applied to ML estimation to find the best parameter settings. By iteratively performing expectation-maximization (EM) procedure, the probability of a model is guaranteed to increase and to converge to local maximum.

However, several researchers have investigated and have pointed out the problems of ML estimation with Baum-Welch algorithm. Basically, the ML estimation with Baum-Welch algorithm recasts the HMM parameters learning problem in a statistical framework and focuses on the estimation of model parameters in order to maximize the likelihood. Though the method has improved recognition accuracy considerably, a higher likelihood does not always imply a better recognition performance, especially when there is poor discrimination across the models.

(2) Discriminative Training

In contrast to ML, an alternative approach uses maximum mutual information (MMI) criterion (Bahl *et al*., 1986) for acoustic modeling. This discriminative criterion attempts to maximize the mutual information of word sequence between hypothesis and reference, by taking a couple of competing hypotheses and trying to reduce the probability of incorrect ones. In several ASR tasks, MMI has been successfully applied in acoustic modeling (Chow, 1990; Normandin, 1991). However, MMI estimation usually suffers from the high computational cost for large vocabulary ASR tasks. Besides, MMI estimation yields higher error reduction on the training set than the reduction on the test set, which is known as the generalization problem. The problem requires additional regularization terms to be integrated in the objective function in order to guide more improvements on the test set (Woodland and Povey, 2002).

One notable drawback of MMI is that it still does not directly integrate the system performance (such as word error rate) into the optimization function. Even if several iterations have been performed, the solution will not necessarily

reflect an improved performance and may risk the over-fitting on the training data. A recent study proposed minimum word error (MWE) and minimum phone error (MPE) criteria for acoustic modeling (Povey and Woodland, 2002). These two criteria combine the training errors into objective function to increase the decoding accuracy both at the word and the phone level. In Section 3.2 (p. 40), these discriminative training criteria will be presented in more detail.

(3) Other Modeling Techniques

Artificial neural networks (ANN) have been shown to be a powerful method for pattern-recognition (Bishop, 1995). However, the time variable makes it difficult to directly handle ASR in ANN. Several studies developed various ANN architectures for time sequence classification (Lippmann, 1989). Recent research further combined the ANN and HMM for large ASR tasks. This architecture of hybrid ANN/HMM models not only has the power of classification but also has the ability to handle temporal patterns. In recent years, the hybrid approaches have been demonstrated to give competitive performance with typical HMM based systems (Johansen, 1996; Ynoguti *et al*., 1998).

2.2.3 Language Modeling

In human speech, it is inevitable that there exists word sequence having similar or identical pronunciations. Even if well-trained acoustic models are used, the decoder may not be able to decode the word strings accurately. To further distinguish the acoustically ambiguous utterances, another knowledge source, the language model, is used to help the decoder find out the most likely word strings. The issue of how natural language regularities can be captured in a statistical framework, and the way to estimate the parameters of a model, has attracted many efforts in language modeling.

Statistical Language Modeling

Statistical language modeling (SLM) was proposed in the 1980's (Bahl *et al*., 1983), which gives a probabilistic description of the word order in natural language. *N*-gram models are the most popular language model for representing the statistical characteristics. An *n*-gram predicts the probability of word wi based on the history of previous *n*-1 words, under the Markov assumption that only the closest *n*-1 words are relevant:

$$P(w_1^{L_w}) = \prod_{i=1}^{L_w} P(w_i \mid w_1^{i-1}) \approx \prod_{i=1}^{L_w} P(w_i \mid w_{i-(n-1)}^{i-1}) \qquad (2.6)$$

where L_w denotes the length of word sequence and n is the order of context.

Larger n could capture more complex language regularities. However, the number of n-grams would increase exponentially with n. Generally, the tri-gram (n=3) language model is a common choice for ASR tasks.

One straightforward approach to estimate the probabilities of a model is to compute the relative occurrence of a word pair over the whole corpora, which is an example of the maximum likelihood (ML) technique,

$$P_{ML}(w_i \mid w_{i-(n-1)}^{i-1}) = \frac{P(w_{i-(n-1)}^{i})}{P(w_{i-(n-1)}^{i-1})} = \frac{c(w_{i-(n-1)}^{i})/N}{c(w_{i-(n-1)}^{i-1})/N} = \frac{c(w_{i-(n-1)}^{i})}{c(w_{i-(n-1)}^{i-1})} \tag{2.7}$$

where $c(.)$ is the number of occurrence of an n-gram and N is the total number of occurrences. However, ML estimates suffer from the sparse distribution of n-grams. Since most of the n-grams either never occur or occur with very low counts, direct use of ML estimation from n-gram occurrences would be greatly biased due to the sparsity. Besides, when more words have been used in the language model, more data needs to be collected to yield representative language regularities. In practice, more data is not a solution to the data sparsity, even if data is collected from the web (Zhu and Rosenfeld, 2001).

Smoothing

To deal with the sparse estimation problem, an enormous number of techniques have been proposed to reliably estimate the n-gram probability so as to avoid zero counts and unreliable estimates for unseen events (an occurrence of an n-gram) or when there is insufficient data. The process, which aims at making probability distributions of n-grams in the training data more uniform, is called smoothing.

One standard approach introducing discounted probability and lower-order models to smooth the probability estimates is described below (Kneser and Ney, 1995):

$$P(w_i \mid w_{i-(n-1)}^{i-1}) = \begin{cases} P^*(w_i \mid w_{i-(n-1)}^{i-1}) & c(w_{i-(n-1)}^{i}) > 0 \\ \beta \times P(w_i \mid w_{i-(n-2)}^{i-1}) & c(w_{i-(n-1)}^{i}) = 0 \end{cases} \tag{2.8}$$

where P^* can be obtained by multiplying a discount ratio to (2.7) and β is considered as a normalization constant to the lower-order estimate:

$$\beta = \frac{1 - \sum_x P^*(x \mid w_{i-(n-1)}^{i-1})}{1 - \sum_x P^*(x \mid w_{i-(n-2)}^{i-1})} \tag{2.9}$$

where $x \in (w_{i-(n-1)} w_{i-(n-2)} \ldots w_{i-1} x)$. The discount ratio can be calculated by using various approaches, which are presented in the following.

(1) Good-Turing Discounting

One straightforward smoothing technique is to modify the original counts by redistributing the probability mass such that language model would be less biased by the frequently-occurred or rarely-occurred *n*-grams. An early implementation of this idea is Good-Turing discounting (Good, 1953), which modifies the number of occurrence *r* of an *n*-gram as follows:

$$c^* = (c+1)\frac{E(n_{c+1})}{E(n_c)} \tag{2.10}$$

where n_c is the number of different *n*-grams that appears *c* times. The probability estimate of the *n*-gram is:

$$P_{Good-Turing}(w_i \mid w_{i-(n-1)}^{i-1}) = \frac{c^*}{N} \tag{2.11}$$

It is clear that the Good-Turing estimate can not be used for *n*-grams with zero occurrences ($E(n_c) = 0$). In (2.10), $E(n)$ is a function to estimate different *n*-grams that happen exactly *c* times.

(2) Jelinek-Mercer Smoothing

Unlike Good-Turing smoothing, which yields a probability distribution from modified counts, the Jelinek-Mercer smoothing (Jelinek and Mercer, 1980) estimates the probability by combining the higher-order maximum likelihood models P_{ML} and lower-order smoothed models P_{JM} in a linearly interpolated form:

$$P_{JM}(w_i \mid w_{i-(n-1)}^{i-1}) = \lambda_{w_{i-(n-1)}^{i-1}} P_{ML}(w_i \mid w_{i-(n-1)}^{i-1}) + (1-\lambda_{w_{i-(n-1)}^{i-1}}) P_{JM}(w_i \mid w_{i-(n-2)}^{i-1}) \tag{2.12}$$

Where $0 \le \lambda \le 1$ and the uniform distribution on the 0^{th} order can be used as a base case to end the recursive interpolation.

To maximize the probability distribution, the λs need to be carefully estimated. Different methods for finding λ yield a variety of smoothing techniques. The deleted-interpolation method (Jelinek and Mercer, 1980) partitions the training data into several parts and computes the λ by circulating different parts. An alternative is performed by dividing the training data into retained part and held-out part. The held-out part is introduced to calculate the λ and is never used in P_{ML} estimation. Therefore, when held-out estimation is performed, the n_c is obtained from the retained data, rather than the whole training set.

(3) Witten-Bell Smoothing

The Witten-Bell smoothing (Witten and Bell, 1991) was first developed for text compression. The key concept (the method C in the reference) is to re-estimate the unseen *n*-grams from the seen *n*-grams that occur at least once. It can be considered as a linear interpolation of models of different orders to form a

mixture model similar to Jelinek-Mercer smoothing. The interpolation ratio λ can be generally defined as:

$$\lambda_{w_{i-(n-1)}^{i-1}} = 1 - \frac{CNT(w_{i-(n-1)}^{i-1})}{CNT(w_{i-(n-1)}^{i-1}) + \omega \times \sum_{w_i} c(w_{i-(n-1)}^{i})} \tag{2.13}$$

where the hyper-parameter ω can be learned from a development set and the original Witten-Bell smoothing keeps $\omega = 1$. The $CNT(w_{i-(n-1)}^{i-1})$ is the number of unique words following the history $w_{i-(n-1)}w_{i-(n-2)}...w_{i-1}$, which is defined as:

$$CNT(w_i) = |\{w_i : c(w_{i-(n-1)}^{i-1} w_i) > 0\}| \tag{2.14}$$

In the smoothing techniques presented so far, the Good-Turing smoothing is basically a ML estimate, which modifies the non-zero occurrence to produce a uniform distribution. It is often used in combination with other techniques. The Jelinek-Mercer smoothing yields probability estimates by interpolating models of different orders associated with an interpolation ration (weight). The Witten-Bell smoothing also can be viewed as an instance of interpolation-based method. In addition, several popular smoothing techniques perform probability estimate by backing-off to lower-order model. Although lower-order distributions are used both in interpolated model and back-off model, the key difference is in determining the probability of *n*-grams with nonzero counts: the interpolated models use the information from lower-order distributions while back-off models do not. In the following, we will present two popular back-off methods: the Katz smoothing and the Kneser-Ney smoothing.

(4) Katz Smoothing

Katz smoothing (Katz, 1987) basically extends the Good-Turing smoothing by combining models of different orders. The Katz *n*-gram model can be defined recursively in terms of lower (*n*-1)-gram model with a count threshold k_c for discounting (Katz suggests k_c=5) as follows:

$$P_{Katz}(w_i \mid w_{i-(n-1)}^{i-1}) = \begin{cases} \dfrac{c(w_{i-(n-1)}^{i})}{c(w_{i-(n-1)}^{i-1})} & \text{if } c > k_c \\ d_c \times \dfrac{c(w_{i-(n-1)}^{i})}{c(w_{i-(n-1)}^{i-1})} & \text{if } 1 \le c \le k_c \\ \beta(w_{i-(n-1)}^{i-1}) P_{Katz}(w_i \mid w_{i-(n-2)}^{i-1}) & \text{if } c = 0 \end{cases} \tag{2.15}$$

where the discount ratio is calculated as follows:

$$d_c = \frac{\frac{c^*}{c} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}} \tag{2.16}$$

and if d_c is determined, the normalization factor $\beta(w_{i-(n-1)}^{i-1})$ can be chosen as:

$$\beta(w_{i-(n-1)}^{i-1}) = \frac{1 - \sum_{w_i:c>0} P_{Katz}(w_i \mid w_{i-(n-1)}^{i-1})}{1 - \sum_{w_i:c>0} P_{Katz}(w_i \mid w_{i-(n-2)}^{i-1})} \tag{2.17}$$

so that the total number of counts is unchanged by Katz smoothing. That is,

$$\sum_{w_i} c_{katz}(w_{i-1}^{i}) = \sum_{w_i} c(w_{i-1}^{i})$$

Note that in Katz smoothing, the large counts $c > k$ are taken to be reliable and are not discounted. The *n*-grams with lower counts are discounted to compensate, using the discount ratio derived from Good-Turing smoothing. If an *n*-gram with zero counts is observed, the probability is estimated by backing-off to lower-order model with a normalization factor. Generally, the Katz smoothing is easy to implement and has represented the state-of-the-art in smoothing techniques for many years (Goodman, 2000).

(5) Kneser-Ney Smoothing

Another back-off method, whose variant has been shown to consistently outperform other widely-used smoothing techniques (Chen and Goodman, 1996), is Kneser-Ney smoothing (Kneser and Ney, 1995). Different from other back-off methods such as Katz smoothing that estimates the lower-order probability from the smoothed lower-order distribution, the Knesr-Ney smoothing further proposed a singleton distribution to compute the lower-order distribution only from the *n*-grams that are observed just once. In fact, the lower-order distribution becomes an important factor only when a few or no counts are present in the higher-order distribution. The intuition of Kneser-Ney smoothing can be formulated as follows:

$$P_{Kneser-Ney}(w_i \mid w_{i-(n-1)}^{i-1}) = \begin{cases} \dfrac{\max\{c(w_{i-(n-1)}^{i}) - D, 0\}}{\sum_{w_i} c(w_{i-(n-1)}^{i-1})} & \text{if } c > 0 \\ \delta(w_{i-(n-1)}^{i-1}) P_{Kneser-Ney}(w_i \mid w_{i-(n-2)}^{i-1}) & \text{if } c = 0 \end{cases} \tag{2.18}$$

where D is a discounting parameter and δ is chosen to make the distribution sum to 1. An extensive comparison shows that the Kneser-Ney smoothing works very well (Chen and Goodman, 1996). However, this back-off version always performs discounting even when it has effectively lowered the probability

(Goodman, 2004). A variant was motivated in (Chen and Goodman, 1996), which represents the back-off smoothing in an interpolated form and is known as modified Kneser-Ney smoothing:

$$P_{Kneser-Ney}(w_i \mid w_{i-(n-1)}^{i-1}) = \frac{\max\{c(w_{i-(n-1)}^{i}) - D, 0\}}{\sum_{w_i} c(w_{i-(n-1)}^{i-1})} + \lambda P_{Kneser-Ney}(w_i \mid w_{i-(n-2)}^{i-1}) \quad (2.19)$$

where

$$\lambda = D \times \frac{CNT(w_i)}{\sum_{w_i} c(w_{i-(n-1)}^{i-1})} \quad (2.20)$$

and $CNT(w_i)$ is defined in (2.14). The discounting parameter D is left open which results in some variants.

Overall, the Kneser-Ney smoothing and its variant calculate the lower-order probability in a novel manner to produce a powerful estimate for the events with very low counts. Several recent studies have demonstrated its superiority in language modeling (Chen and Rosenfeld, 2000; Goodman, 2004).

Other Statistical Modeling Techniques

Apart from the popular *n*-gram models and smoothing techniques, there are also various approaches developed in language modeling, such as exponential models and adaptation models. In fact, the data distribution in natural language is often unreliable due to the unnecessary fragmentation of contexts (the sparse data problem). One approach, based on maximum entropy principle, introduces an exponential model to avoid the data fragmentation (Pietra *et al*., 1992). However, this approach suffers from the slow training procedure and there is no explicit control on parameter variances.

Moreover, natural languages are highly heterogeneous, covering various topics and categories. In many real ASR tasks, where domain-specific data may not be available, several approaches have been proposed to improve language modeling by adaptation (Clarkson and Robinson, 1997; Mahajan *et al*., 1999). In practice, adaptation can be considered as an approach to sharpen the language model. It can be performed by building language models for each topic or combining different topic-specific language models through interpolation (Iyer and Ostendorf, 1999), where the source may come from in-domain or cross-domain data.

Generally, adaptation is a process to update current language model by the information captured from new incoming data. This design allows the model to be more close to a specific task domain. Therefore, significant perplexity[10] and word error rate reduction have been observed in these studies (Bellegarda, 2004). One interesting research investigated the possibility to further improve the performance by combining various modeling techniques together (Goodman, 2000). The results show that a reduction of the perplexity by almost a half can be achieved, and the smoothing technique is an important factor to *n*-gram modeling.

Estimation of Language Model Parameters

Intrinsically speaking, even for the most popular *n*-gram language model, the statistical form is just a set of word pairs and their associated probabilities, which is not the nature of language and very crude syntactic dependencies can be captured by such a language model. In addition, although the use of language model in many large vocabulary recognition systems has resulted in observable improvements, unfortunately, maximum likelihood estimation (MLE) for language modeling attempts to maximize the likelihood without considering the actual acoustic confusions. Moreover, the estimation of model parameters is usually performed in isolation and the likelihood improvement does not always reflect better recognition accuracy (Chen *et al*., 1998; Printz and Olsen, 2000).

Recent developments have introduced discriminative training on estimating language model parameters. Several discriminative criteria have been demonstrated to give significant improvements, such as minimum classification error (MCE) (Chen *et al*., 2000; Juang *et al*., 1997; Kuo *et al*., 2002; Paciorek and Rosenfeld, 2000), minimum sample risk (MSR) (Gao *et al*., 2005) and reranking techniques with the perceptron algorithm (Roark *et al*., 2004). Among the many studies, MCE is a popular choice and is shown to yield significant improvements. Unlike typical language modeling approach which maximizes the likelihood, MCE criterion aims to reduce the decoding errors by increasing the separation between training samples. The key of this training framework is to define an objective function which takes into account the language model parameters and the recognition performance as function variables. Moreover, an optimization method is applied to search the function minimum such that decoding errors can

[10] The perplexity is the most widely-used evaluation metric for language models. Nevertheless, there have been many counter-examples in the literature showing that lower perplexity does not always lead to a lower word error rate (Azzopardi *et al*., 2003; Chen *et al*., 1998).

be reduced. Training procedure is performed iteratively till convergence is reached. These discriminative training criteria will be presented in Section 3.2 (p. 40).

Graph Representation of Language Model

In the recent past, weighted finite-state transducer (WFST) technique has attracted a lot of interests in speech recognition. A WFST is an extension of finite state machine. One special feature of WFST is that each arc, which goes from one state to another (or a loop), carries three messages: the input label, the output label and the transition weight. This property facilitates the use of WFST in ASR, particularly the mapping between two different information sources. For instance, a WFST can be used to encode the mapping from a word to its phone sequence(s) in the dictionary. Language model that is compiled as a WFST represents the transitions from one word history to another weighted by the corresponding *n*-gram probabilities. When knowledge sources are represented by WFSTs, graph composition can be applied to further combine different WFSTs in a single search space (graph). By performing various optimization techniques such as determinization and minimization (Mohri, 1997) to eliminate redundant arcs and states on the graph, decoding time and memory requirements can thus be greatly reduced.

Amongst WFST operations, the determinization is particularly critical; it is the process of producing a deterministic graph which is equivalent to a non-deterministic one[11], such that the deterministic graph contains at most one path matching any input sequence. This irredundancy reduces the time and space requirements to process an input sequence (Mohri, 1997; Mohri *et al*., 2000b) during decoding.

The efficient representation of WFST allows knowledge sources to be represented in a flexible manner for more sophisticated WFST operations. Recent studies using WFST in ASR have demonstrated a significant improvement in decoding efficiency than traditional speech recognition systems (Caseiro and Trancoso, 2002; Kanthak *et al*., 2002; Saon *et al*., 2005). More technical details on graph construction procedure and WFST operations will be presented in Section 3.6.3 (p. 64).

[11] In fact, not all WFSTs can be directly determinized (Mohri, 1997). For many ASR tasks, however, the WFST representation of knowledge source is determinisible.

2.3 Search Strategy

In HMM-based speech recognition systems, one fundamental issue is the decoding problem. Given a sequence of observations and a set of trained HMMs, the decoding process attempts to find out the best state sequence for the observations. Since searching all possible state sequences to discover the best one is extremely computationally-intensive, one alternative solution for large vocabulary ASR is to incorporate various approximation techniques which make the search suboptimal.

One prevalent approach in ASR is to find the most likely state sequence using the Viterbi algorithm (Viterbi, 1967), which performs the search in an HMM state-connected network. Due to the graph nature, finding the most likely sequence is solved as a shortest-path problem in an acyclic graph. Several dynamic programming (Bellman, 1952) techniques were investigated to address (Beulen *et al.*, 1999; Dijkstra, 1959) this issue. The Viterbi algorithm is an application of dynamic programming techniques, which only requires to compute the score of the best path reaching state at each time frame. Thus Viterbi algorithm is a time-synchronized search method. Since at each time frame it holds a list of candidates and propagates the candidates to next time frame, even for a small set of models, the book keeping induces a significant computation overhead.

Search using Viterbi algorithm usually incorporates pruning techniques by keeping only the top-N candidates at each time frame during decoding (or histogram pruning that adaptively changes the number of active hypotheses (Steinbiss *et al.*, 1994)). Viterbi search incorporating the pruning technique is known as Viterbi beam search. This approach significantly reduces computation cost and has been widely used in many large vocabulary ASR systems. However, as more and more data becomes available for large vocabulary ASR systems, where thousands of acoustic models may be used and the language model may contain tens of millions of *n*-grams, an efficient search strategy is indispensable for fast decoding. The following sections introduce three popular search strategies and discuss their performance.

2.3.1 Dynamic Style Beam Search

Dynamic programming search techniques have been successfully applied to handle speech recognition tasks since their introduction in the 1970's (Jelinek,

1976). Although the approximation makes the search suboptimal, it greatly reduces the computational cost and makes the search technique applicable to complex ASR tasks. However, as more word alternatives are needed to be held during decoding process, it not only produces more acoustic confusions but also increases the computation costs.

Moreover, the search space is a connected network. A bunch of hypotheses may be produced from a state. If there are N active states at a given time frame and each state involves M successors, the number of possible hypotheses would be $N \times M$, yielding an intractable computation.

Reduction of Search Effort

To keep the search efficient, the dynamic style beam search (DSBS) uses several techniques during decoding, namely an efficient representation of the lexicon, the beam search and the integration of language model.

In large vocabulary ASR tasks, many words begin with the same leading phone(s). Merging identical phones would greatly reduce the search effort without loss of decoding accuracy. This can be performed by representing the words in a word graph or a tree structure (Soong and Huang, 1991). For a large list of hypotheses, one can eliminate the less probable hypotheses and hold the promising ones. This technique is called beam search. Moreover, one can further incorporate a language model during decoding. The use of language model is to help the decoder determine the best word strings among a list of hypotheses. As a result, hypotheses with lower scores can be pruned from the list as soon as possible.

Word Graph Construction and Search

To implement this idea, various approaches have been proposed which extend dynamic programming. One efficient approach is the word graph algorithm (Beulen *et al.*, 1999; Ney and Aubert, 1994; Ney and Ortmanns, 1997; Oerder and Ney, 1993; Ortmanns *et al.*, 1996c), which is performed by introducing the following two quantities:

- $Q_\tau(t,s)$, the score of the best path at time t in state s and the starting time of the tree is τ.
- $H(w,t)$, at ending time t, the probability of generating acoustic observation vectors $X=\{x_1,x_2,...,x_t\}$ and a word sequence which ends with word w.

Based on these two quantities, the algorithm selects the best predecessor for each word pair with ending time *t* to expand a new tree for time (*t+1*). The decision is performed by a recombination process and the rest of the paths are not considered any further. For all hypotheses in the beam, the word strings are organized as a tree, and word labels are distributed over the arcs with corresponding starting and ending time of the word. Since common word labels are merged in the form of a tree, this hypothesis representation is more memory efficient than keeping all possible word strings in the beam. Meanwhile, once the tree is built, nodes with identical time information are merged. Only one arc is kept if several arcs associated for the same word carry the same word labels. A more flexible word graph can thus be constructed to cover all hypotheses in this representation but with less complexity. The complete algorithm of word graph construction and its implementation is elaborated in (Ney and Aubert, 1994; Ortmanns *et al*., 1997b).

A simple example, using trigram language model to build the word graph, is shown in Figure 2.2. The solid circle and empty circle represent respectively the word history and the word boundary. For each word history, a separate lexical tree is generated starting from the word history as depicted in doted triangles. Solid lines are transitions between acoustic models, and dotted lines between two states are transitions from a word end to the root of lexical copy. The dotted lines are the place where language model is used. Before processing new word hypotheses from the lexical tree, language model probabilities are incorporated into the scores. Taking Figure 2.2 for instance, given word history (*xy*) and (*zy*), two hypotheses reaching the end of word *x*, producing the word history (*yx*) which is the root of a tree lexicon in upper triangle. To expand word hypotheses from word history (*yx*), the language model probability is applied to determine the best score that will be propagated to the root of lexical tree.

Optimization Techniques

The word graph approach combines word hypotheses in the form of a graph. In this representation, word hypotheses can be easily produced from the small graph, without actually keeping a complete list of word strings. When the approach employs a pruning technique using language model probability, scores of hypotheses can be compared in advance to determine the best hypothesis which is allowed to perform state expansion to succeeding words. In addition to the efficient representation, various optimization techniques are combined with the word graph search method to improve decoding efficiency and memory requirements.

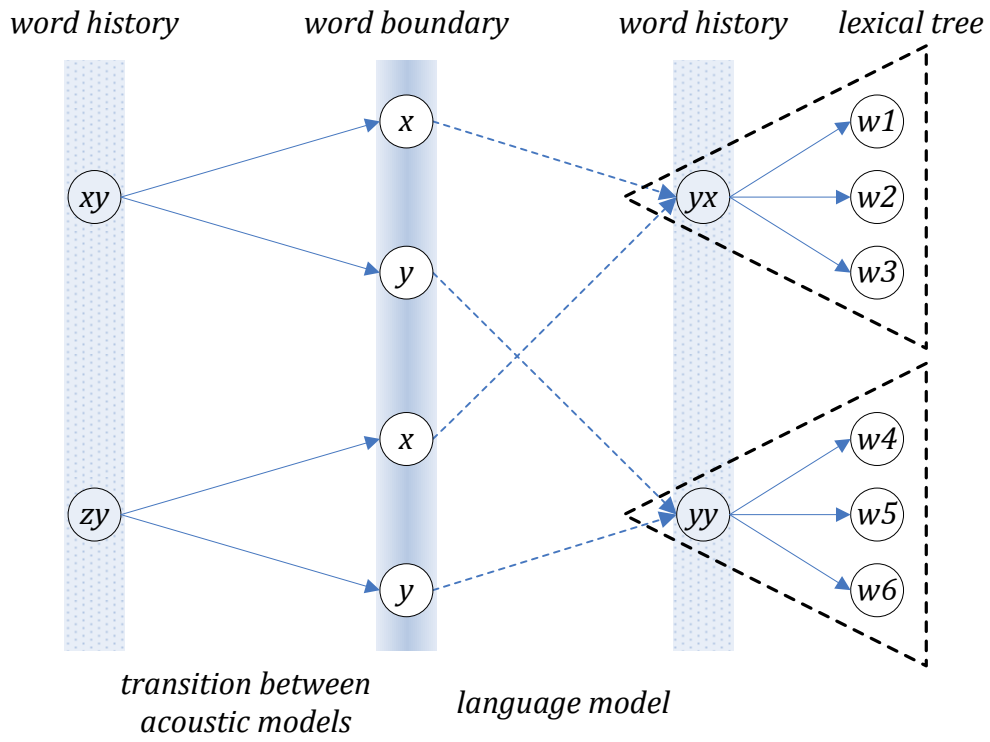

Figure 2.2: Word graph expansion and search. When tokens reach the root of a lexical tree, language model probabilities are taken into account to determine the best hypothesis.

One popular solution is histogram pruning (Haeb-Umbach and Ney, 1994; Steinbiss *et al.*, 1994) which limits the number of possible hypotheses to a given threshold. If the number of active states is greater than the threshold, a selection process is performed to trim the hypotheses with relative low scores.

It can be readily noticed that the allocated tree lexicons would occupy a lot of memory space and produce numerous word hypotheses for large vocabulary ASR tasks, particularly when a complex high order language model is used. In this case, determining the best score for propagation would become less efficient. Another possibility is the use of language model look-ahead (Ortmanns *et al.*, 1996a; Ortmanns *et al.*, 1997a). Unlike language model pruning mentioned above in expanding tree hypotheses, the basic concept of language model look-ahead is to make use of language model probabilities as soon as possible in the decoding process. Before tree expansion from the word boundary, language model probability is calculated and accumulated into the score of the best hypothesis. The use of language model look-ahead probability increases the score difference between hypotheses and makes pruning more efficient.

Language model look-ahead is a general technique. In modern search space design, weight pushing, which re-distributes language model probability on a weighted finite-state network, is an extension of this concept. More details regarding weight pushing over finite state network will be given in Section 3.6.5 (p. 71).

2.3.2 A* Search

In graph-searching problem, the A* algorithm (Hart *et al.*, 1968) is a common choice for finding the shortest path. Given an initial state and a goal state, the algorithm performs the search from the initial state and generates successor states under a certain condition. This process is repeated until one of the successors reaches the goal state and satisfies the condition. The returned path is the solution to the problem. A key component here is a heuristic function which is employed to help estimate the distance to reach the goal state. The choice of a heuristic function is arbitrary. A good heuristic is the key for finding a good solution quickly and accurately. In speech recognition, the decoding process in a state-connected network can be considered as a graph searching problem where the goal of decoder is to find out the most likely sequence from the network. If the A* algorithm is implemented as the search method, the returned path is the sequence with highest score.

The Heuristic Function

In a path-finding problem, the A* algorithm tries to find the shortest path (or lowest cost) under the decision:

$$f'(n) = g(n) + h'(n)$$

where n is the current state. The function $g(n)$ corresponds to the sum of costs over traversed paths from the initial point to the current state. Since the best solution has not yet been found, the cost from current position to goal state is a guess, which is estimated by $h'(n)$ according to a heuristic function. The sum of $g(n)$ and $h'(n)$ is an estimated lowest cost up to current state. The algorithm then generates successors. If the successors are generated without limit, memory allocation would be a potential problem. For searching in a larger graph where several transitions may reach the same leaf, memory space allocation can be alleviated by keeping the best score at each state for further successor expansion.

The A* algorithm usually introduces an OPEN list and a CLOSED list. The OPEN list keeps the states that have not yet been expanded. Each time a state is popped

from the list. When state expansion is performed, the state is placed in the CLOSED list. The use of OPEN and CLOSED lists allows us to choose the best state before searching according to the cost which is the sum $f(n)$. For instance, if a successor of state *n* is already in the CLOSED or OPEN list, and its expansion has higher cost than $f(n)$, the successor of state *n* can be discarded since a better expansion has been found with lower estimate. The main structure of A* algorithm is a loop which repeatedly pops the state with lowest cost from the OPEN list to generate its successors. If the current state is a goal state, the loop terminates and returns a solution. The pseudo-code of A* algorithm can be found in (Nilsson, 1997).

Performance Issues

The performance of A* search depends critically on the choice of a heuristic function. At each state, the heuristic function helps the algorithm make an estimation to determine how far it is from the current state to the goal state, and to reduce the state expansion. Thus, a good heuristic function will give an optimal solution at high search efficiency while a bad heuristic function may result in a sub-optimal path or may yield an exhausted exploration of the search space. The performance of a heuristic function is defined in terms of admissibility and computational efficiency.

(1) Admissibility

If a search algorithm is guaranteed to return an optimal solution, the algorithm is said to be admissible. For the A* algorithm, it is admissible only if the heuristic function never over-estimates the cost. That is, the estimated cost from current state to the goal state must be less than or equal to the actual cost. Otherwise, the over-estimated cost could make the algorithm choose a sub-optimal solution. To ensure that the paths with lowest cost are always found, the global admissibility of A* algorithm using heuristic function $h(n)$ is expressed as below:

$$h(n) \le h^*(n) \tag{2.21}$$

where $h(n)$ is the estimated cost and $h^*(n)$ is the actual cost from state *n* to goal state. Although (2.21) ensures that the lowest cost of traversing paths, it is not easy to find such a heuristic function and to prove its global admissibility for general path searching problems. The choice of heuristic function may vary depending on the search problem.

One property of heuristic function is the monotonicity, meaning that the estimated cost of a state n_1 to its neighbor n_2 is always less than the actual cost.

$$h(n_1) - h(n_2) \leq h^*(n_1) - h^*(n_2)$$

The property ensures the score difference between two consecutive states satisfies the inequality. Thus, the monotonic is also locally optimistic, which implies global optimal of the heuristic function but not vice versa.

(2) Computational Efficiency

In addition to the admissibility of a heuristic function, the search efficiency is an important consideration for A* algorithm. As one can imagine, when the graph is getting larger and more complex, popping states from a large list to determine the best score would be a CPU intensive task and large number of generated successors still make the list difficult to handle. In practice, one of the choices may be made according to the need, a) either an optimal solution should be obtained or b) a good approximation is satisfying.

The Stack Decoder

The first ASR decoder based on A* search was proposed in (Jelinek, 1969; Jelinek *et al*., 1975), which is a variant of A* search and is known as a stack decoder. This approach has many appealing properties, especially for the matching of acoustic and language model during decoding. In many ASR search implementations (Aydin *et al*., 2005; Kenny *et al*., 1992; Odell *et al*., 1994; Paul, 1992), the stack decoder has been modified for various tasks.

A stack decoder considers the speech decoding as a problem of graph searching where the words are represented as states and language model probabilities are distributed over state transitions. From the initial state to the final state, the goal of decoder is to find out the most likely state sequence. Due to the attractive control strategy which focuses on the search without managing different information on the network, A* algorithm allows acoustic models, language model and a variety of sources to be integrated in a single search space.

A stack decoder usually requires two passes to complete speech decoding:

(1)Generates multiple hypotheses
(2)Performs accurate search on the generated hypotheses

The first pass also enables the decoder to perform N-best sentences search. With a good heuristic function, the generation of hypotheses is fast. However, there are some drawbacks when performing recognition on the second pass. In large vocabulary continuous speech systems, where word boundaries are difficult to

locate, there may exist a high number of hypotheses, which will increase the computation on the second pass. Besides, the heuristic function must be carefully chosen so that the recognition accuracy would not be deteriorated. From a technical point of view, the implementation of a stack decoder requires extra work to maintain the list of hypotheses in an efficient data structure for fast access, such as a sorted list or a tree. In a large search network, the decoding speed or accuracy may not be as competitive as standard Viterbi algorithm, particularly when finite state decoding techniques are introduced to optimize the graph.

2.3.3 Decoding on Static Graphs

Integrated Search Network

When the three main knowledge sources, namely the acoustic models, pronunciation lexicon and language model, are available, one possibility to perform the search is to combine all the sources in a single network. During decoding, parameters from various sources are accessed at the same time. Applying search constraints such as beam search during decoding or performing finite state machine operations beforehand to reduce computation and network complexity, decoding efficiency can be further improved. In the 1970's, the concept of integrated search network was implemented in the HARPY system by taking the advantage of finite state network (Juang and Rabiner, 2005; Lowerre and Reddy, 1990).

In the integrated search network, decoding thus becomes a simple search task. However, it is obvious that the search efficiency relies heavily on the network design. As knowledge sources are getting more complex, from context-independent to context-dependent phones, tens of thousands of vocabulary words and millions of *n*-grams in the language model, optimization methods on integrated finite state network become the most demanding task in large vocabulary ASR.

Two main strengths of using an integrated search network are:

- Determinization: each hypothesis is evaluated once.
- Reduce housekeeping overhead during decoding.

With these two features, one can perform network optimization methods on the whole graph to produce an equivalent but a smaller one. Recent developments of

finite state machine operations further extend the idea of network optimization to ASR tasks, where various knowledge sources can be combined in a single network.

Weighted Transducer for Speech Components

A finite state machine (FSM) is a finite-state graph which consists of a set of states and transitions, represented by nodes and arcs respectively. This foundation gives a general representation for mathematical model in natural language processing (Maurice, 1987; Mohri, 1996). By extending the use of FSM, if a weight is distributed over the arc, the transition between states becomes a weighted mapping from one word to another. This can be considered as a probability distribution over word strings. Further extension of FSM design was investigated such that various information sources can be incorporated in a common representation (Mohri, 1996; Pereira *et al.*, 1994).

The generalization of classical FSM takes the form of weighted finite-state transducers (WFST) T. A WFST over a weight set K is given as a set of states Q, an input symbol set Σ, output symbol set Ω, a set of transitions E, an initial state $I \in Q$, a set of final states $F \subseteq Q$, and some initial weight λ and final weight function ρ.

$$T = (\Sigma, \Omega, Q, E, I, F, \lambda, \rho) \tag{2.22}$$

With this design, the WFST is able to cope with complex information sources in a simple representation. In particular, the mapping of an input and output symbol pair with associated weights opens the possibility of using WFST for speech components.

Typical ASR systems extract the message content for the speech signal by decoding acoustic observations into a sequence of basic units (often contextual phones). The phone sequence is then mapped into word(s) according to the dictionary. To reduce acoustic confusions, language model probability is used to determine the most probable word strings. The recognition cascade from observations to words is shown in Figure 2.3.

If we consider the recognition cascade in a reverse order, we get the following mapping for each FSM:

- G: word sequence $\rightarrow$ words represented by different numbers.
- L: word $\rightarrow$ corresponding pronunciation sequence(s).
- C: context-independent (CI) phone $\rightarrow$ context-dependent (CD) phone.
- H: CD phone $\rightarrow$ associated HMM.

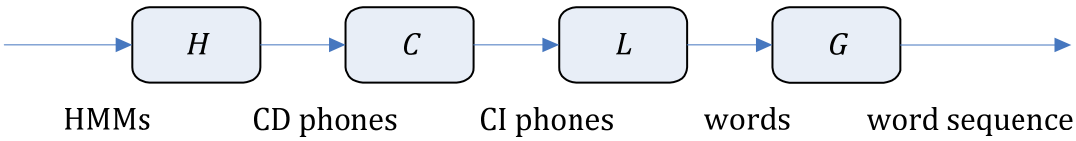

Figure 2.3: Recognition cascade from HMMs to word sequence.

From G to H, these machines perform the mapping from high-level word sequence to low-level HMMs. The resulting graph is a finite-state network combining all the information sources. The decoder then only needs to perform the search of the most likely sequence from the initial state to a final state.

With the use of WFST to represent the various speech components, different sources can be easily integrated in a single graph for further optimizations. When speech components are in the form of an FSM, composition and determinization are commonly used for graph construction. The first operation composes two FSTs into one FST. The latter is an optimization technique which eliminates redundant arcs to yield an equivalent but more efficient graph. For most ASR systems whose search space is constructed as a WFST, graph construction procedure basically follows the cascade shown in Figure 2.3. FST composition is performed from right to left (Mohri *et al.*, 2000a):

$$H \circ C \circ L \circ G \tag{2.23}$$

Depending on the problem design, various modifications may be introduced for a specific task. More details about integrated decoding graph construction procedure are presented in Section 3.6 (p. 58).

Graph Performance

Although WFST gives a "natural" representation for various sources, and allows them to be integrated in a decoding graph, there are still essential issues to deal with in order to decode the graph with high efficiency and accuracy: (1) graph optimization and (2) weight pushing.

(1) Graph Optimization

During each FST composition step, the temporal graph often consists of states with a bunch of transitions to successors. Some of the transitions carry the same input, output symbols and weights. If these transitions are not eliminated before

decoding, the decoder would have to keep more candidates in the hypothesis list. In addition, when the decoder is going to perform state transitions, one source of overhead is that redundant transitions make the decoder repeatedly generating new hypotheses from the same source with the same acoustic and language model scores. As illustrated in Figure 2.4, scores of hypotheses over transitions $\overrightarrow{01}$ and $\overrightarrow{02}$ are the same. For a large beam size, this would make the decoder much less efficient both in terms of time and memory space.

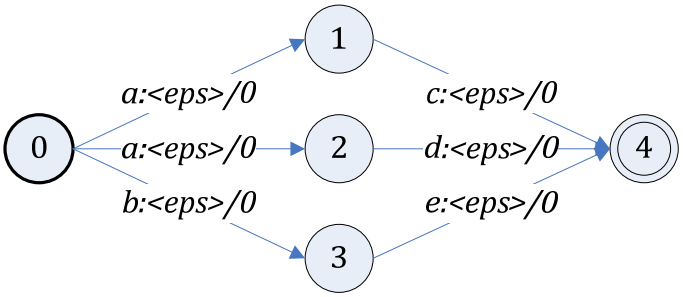

Figure 2.4: A non-deterministic FST. Each arc carries input symbol, output symbol and transition weight.

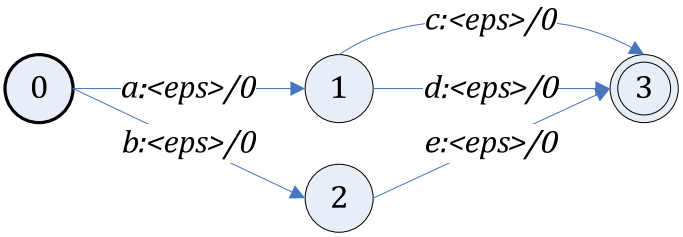

Figure 2.5: A deterministic FST where redundant arcs are eliminated. States are also renumbered.

One important technique to deal with this issue in transducer operation is determinization (Mohri, 1997). For a weighted finite-state network, determinization eliminates the redundant arcs to make an equivalent and deterministic graph, meaning that at each state, there exists at most one transition which is labeled with any element of the input alphabet.

Figure 2.5 shows a deterministic transducer determinized from the one on Figure 2.4; two transitions with identical input symbol *a* are merged. At each state, the graph contains at most one transition with any given input symbol. It is clear that deterministic transducers are more efficient than non-deterministic ones, yielding a unique choice to determine the transition without keeping track of different paths (Mohri, 1997; Mohri and Riley, 1998; Mohri and Riley, 1999).

(2) Weight Pushing

For ASR using WFST as a decoding graph, the weights distributed on the arcs of G come from the language model probabilities, HMM state transition tables and word variants. When G is further composed with the graph L, there exist transitions in which the input symbols are the same. The resulting non-deterministic graph is determinized to reduce the complexity. Despite the efficiency improvement, one issue that needs to be carefully handled is how the weights should be distributed, particularly when determinization is performed to eliminate the redundant arcs.

The weights of the WFST can be distributed in many ways that still result in an equivalent WFST. When the graph is composed at the phone level, a word transition is turned into a sequence of phone transitions, and the weight distribution could be even more complex. In many ASR tasks, studies have shown that language model look-ahead method could help decoding speed-up by *looking at* language model probability as soon as possible to perform better beam pruning (Kanthak *et al.*, 2000; Lai *et al.*, 2002; Ortmanns *et al.*, 1996a; Ortmanns *et al.*, 1996b). Therefore, the general principle of WFST weight pushing is to push the weights towards the initial states as much as possible. This weight distribution yields an equivalent but more suitable transducer for pruning and speech recognition.

In practice, the implementation of a weight pushing algorithm must carefully examine a number of additional issues. For instance, if a word insertion penalty is to be added, where should the penalty be used? If it is added to the machine G, it is obvious that the whole probability distribution is shifted by the penalty. Another consideration is that most of the ASR systems incorporate an optional silence (or a short pause) at the end of a word. However, the optional silence is a special word which is not integrated in the language model. In a combined graph, if all phones are treated as regular ones, some transition weights may be distributed over the transition of the optional silence, which may not be reasonable in general ASR applications.

2.4 Summary and Discussion

In this chapter, we have reviewed the speech recognition framework, fundamental problems and various developments in ASR. The Bayes decision theory gives a strong foundation for formulating ASR task in a statistical framework, the performance of which critically depends on a reliable set of parameters.

Error Rate Based Training Method

Most of the studies focus on the modeling problem so as to improve the reliability of model parameters. The MLE is a common estimation approach both in acoustic modeling and language modeling. This approach would give an optimal solution if infinite data is used. However, due to the data sparsity of natural language and the training criterion which attempts to maximize the likelihood, training improvement may not reflect better decoding accuracy. Discriminative training is an alternative to MLE, aiming at increasing the separation between training samples. The MCE is an error rate based criterion which combines training errors into an objective function such that decoding errors can be reduced if the function's minimum is found.

Integrated Parameter Estimation

Moreover, one issue in conventional modeling approach is that the parameter estimation for a sentence is performed in isolation, assuming that parameters of other sources have been reliably estimated, thus ignoring the interdependency between different sources. Recent research on FSM design opens the possibility to integrate various sources into a single search space. The integration of knowledge sources motivates our attempt to build a unified framework for speech recognition. Under such a framework, various optimization approaches can be applied to make the decoding efficient and to update the graph parameters. If the graph is formulated in an objective function, any reduction of the function loss will also enhance the graph quality.

Discriminative Training on Integrated Decoding Graph

Therefore, we propose a novel training framework which introduces discriminative training on integrated decoding graph. With an error rate based criterion and an optimization method, we attempt to reduce the decoding errors by updating graph parameters such that the performance of decoding graph can be directly optimized.

Chapter 3 further investigates the finite-state graph and the various discriminative training criteria. The problem formulation, discriminative training procedure and the implementation of training task are also presented.

Chapter 4 presents the experimental results, together with a discussion of the strengths and shortcomings of this approach.

Chapter 3. Discriminative Training on Static Decoding Graphs

The speech recognition problem has been established in a statistical framework for more than two decades. One essential issue for accurate decoding is the estimation of the various model parameters. For most of the ASR tasks, maximum likelihood estimation (MLE) is a popular choice for parameter estimation. This approach attempts to find the value of parameters from the training samples such that the likelihood of the data is maximized. However, MLE relies on the availability of large training samples; furthermore, the improvement in training does not always reflect higher decoding performance (Chou, 2000; Juang *et al.*, 1997; Nadas *et al.*, 1988). In the following sections, we will extend this discussion on MLE and investigate discriminative training techniques for parameter estimation.

3.1 Discriminative Training

Review from Maximum Likelihood Estimation

The principle of MLE states that the desired probability distribution is the one that makes the observed data most likely. It is performed by adjusting the set of parameters so as to maximize the likelihood function. From a mathematical point of view, the likelihood is a function of the training data:

$$L(x_1, x_2, ..., x_n \mid \theta_1, \theta_2, ...\theta_k) = \prod_{i=1}^{n} f(x_i; \theta_1, \theta_2, ..., \theta_k) \tag{2.24}$$

where $f(x;\theta)$ is a probability density function (PDF), $x_1, x_2, ..., x_n$ and $\theta_1, \theta_2, ...\theta_k$ are respectively the observations and the parameters to be estimated. In the *log* domain, (2.24) can be expressed as:

$$\Lambda = \ln L = \sum_{i=1}^{n} \ln f(x_i; \theta_1, \theta_2, ..., \theta_k) \tag{2.25}$$

The MLE estimate is obtained by maximizing Λ:

$$\frac{\partial(\Lambda)}{\partial \theta_j} = 0, \quad j = 1, 2, ..., k \tag{2.26}$$

where $\ln L$ is assumed differentiable and the shape of $\ln L$ is a concave so that the global maximum exists. This optimal property gives the MLE a strong foundation for finding the best set of parameters. One strength of MLE is that the minimum variance estimates can be achieved if unlimited training samples are provided. However, this assumption has the drawback that the estimates could be highly biased for small samples, especially for large vocabulary ASR tasks where collecting enough data is difficult. Moreover, the goal of MLE is to maximize the likelihood rather than to directly minimize decoding errors.

Optimizing Separation of Training Samples

Since the objective of MLE is not strongly related to the decoding accuracy, this led researchers to explore other techniques that could produce reliable parameters, such as discriminative training. In contrast with MLE, discriminative training aims at optimizing the separation of training samples. It is performed by formulating an objective function that, in some ways, penalizes parameters that are liable to confuse correct and incorrect answers.

In recent decades, various discriminative training criteria such as maximum mutual information (MMI) (Bahl *et al.*, 1986; Normandin, 1995) and minimum phone error (MPE) (Povey and Woodland, 2002) have been successfully developed for producing reliable parameters in acoustic modeling. Several discriminative techniques also have been demonstrated to give significant improvements in language modeling, such as minimum classification error (MCE) (Chen *et al.*, 2000; Juang *et al.*, 1997; Kuo *et al.*, 2002; Paciorek and Rosenfeld, 2000), minimum sample risk (MSR) (Gao *et al.*, 2005), and the reranking techniques with the perceptron algorithm (Roark *et al.*, 2004).

In the following section, we will present the technical details of the training criteria mentioned above, and discuss their strengths and weakness.

3.2 Training Criteria

Discriminative training is performed by formulating the parameters in an objective function, where a wide variety of function choices are possible. Generally, the objective function should convert the function's improvements into higher decoding performance, and to allow for efficient optimization methods. Among the various discriminative training techniques, the MMI and MCE are the most popular ones. We will first present these two criteria and extend the idea to other discriminative techniques.

3.2.1 Maximum Mutual Information

MMI training is performed by maximizing the mutual information between the training samples and word sequences. Suppose A and W are a set of training samples $\{A_1, A_2, ..., A_R\}$ and the corresponding word sequences $\{W_1, W_2, ..., W_R\}$, the objective function of MMI is the mutual information between the two events, given by:

$$\mathcal{F}_{MMI}(\theta) = \sum_{r=1}^{R} \log \frac{P_\theta(A_r, W_r)}{P_\theta(A_r) P_\theta(W_r)} = \sum_{r=1}^{R} \left(\log P_\theta(A_r \mid W_r) - \log P_\theta(A_r) \right) \qquad (2.27)$$

The goal is to find a parameter vector θ that maximizes $\mathcal{F}_{MMI}(\theta)$. Assume that language model $P_\theta(W) = P(W)$ is given, the function's maximum can be obtained by taking partial derivative of $\mathcal{F}_{MMI}(\theta)$ with respect to θ, yielding

$$\frac{\partial \mathcal{F}_{MMI}(\theta)}{\partial \theta_i} = \sum_{r=1}^{R} \left(\frac{\frac{\partial P_\theta(A_r \mid W_r)}{\partial \theta_i}}{P_\theta(A_r \mid W_r)} - \frac{\sum_{\widehat{W}} \frac{P_\theta(A_r \mid \widehat{W})}{\partial \theta_i} P(\widehat{W})}{P_\theta(A_r)} \right) \qquad (2.28)$$

where $P_\theta(A_r)$ is expanded by:

$$P_\theta(A_r) = \sum_{\widehat{W}} P_\theta(A_r \mid \widehat{W}) P_\theta(\widehat{W}) \qquad (2.29)$$

MMI estimates implicitly take into account all possible word sequences.

The objective of MMI is to increase the first term of (2.27), which corresponds to (2.25) of MLE. However, MMI differs from MLE in that the parameter estimation also tries to decrease the probability of incorrect word sequences (by subtracting the second term in (2.27)) in the meanwhile, for all possible competing word hypotheses as shown in (2.29).

Optimization Method

To estimate the model parameters for MMI training, the prevalent technique comes from the extended Baum-Welch algorithm (EBW) (Gopalakrishnan *et al.*, 1991). It is an extension of the Baum-Eagon inequality for optimizing rational objective functions and is applied only to discrete HMMs. Further extension to continuous HMMs with Gaussian densities was proposed by (Normandin, 1991), where the parameter update equations are performed more efficiently than with the traditional gradient descent algorithm.

In addition to EBW, there are various approaches that are proposed aiming at faster convergence or reliable estimation. In (Zheng *et al.*, 2001), a new set of equations is derived based on quasi-Newton algorithm. The training speed and decoding accuracy are both improved. In (Schluter *et al.*, 1997), the generalized probabilistic descent (GPD) method (presented in the next section) was implemented to make a comparison with EBW in MMI training. The result shows the strong similarities on Gaussian densities between GPD and EBW, and that both methods give competitive performance.

Discussion

Many studies in the literature use MMI training for acoustic modeling. In fact, it is also possible to apply MMI criterion in language modeling, such as (Ohler *et al.*, 1999) and (Warnke *et al.*, 1999), where the latter further incorporates MMI in optimizing the interpolation parameters of language model (a smoothing technique presented in Section 2.2.3). Both in acoustic and language modeling, the MMI is shown to give significant word error rate reduction in comparison to the standard MLE criterion.

3.2.2 Minimum Classification Error

In contrast to MMI, MCE aims at minimizing the decoding errors on the training data. Using simple zero-one cost function to measure the error rate is straightforward; however, the piecewise-constant property violates the constraint that the objective function should be continuously differentiable such that a simple numerical search methods can be applied to reach the function's minimum. Therefore, most of the recent MCE applications introduce embedded smoothing (Juang *et al.*, 1997) for a loss function. The use of MCE criterion in discriminative training requires the following functions to be defined:

- Discriminant function
- Misclassification function
- Optimization function

The first step is to define a discriminant function that can return a value suitable for the classification problem. In a speech recognition task, the score, which is a sum of acoustic *log* likelihood and language model *log* probability, is a reasonable choice. Once a discriminant function is selected, the second step introduces a misclassification measure combining discriminant and anti-discriminant criteria in a functional form. For instance, if the discriminant function returns the score

of reference, the anti-discriminant function will represent the score of competitive hypotheses. Thus, a misclassification function can be considered as a decision process taking into account the correct and incorrect hypotheses. Then, the third step applies an optimization method on the misclassification measure to obtain a set of parameters such that the error rate is minimized.

In many ASR tasks, given a word string *W*, a set of acoustic model Λ, a set of transition weights Γ and an observation sequence *X*, the discriminant function is defined as the following:

$$g(X,W,\Lambda,\Gamma) = a(X,W,\Lambda) + b(W,\Gamma) \tag{2.30}$$

where $g(X,W,\Lambda,\Gamma)$ is the score combining acoustic *log* likelihoods $a(X,W,\Lambda)$ and language model *log* probabilities $b(W,\Gamma)$.

The misclassification measure is taken of the form:

$$d(X,\Lambda,\Gamma) = -g(X,W_{ref},\Lambda,\Gamma) + G(X,\{W_{hyp}\},\Lambda,\Gamma) \tag{2.31}$$

and $G(X,\{W_{hyp}\},\Lambda,\Gamma)$ is given by the average score of N-best competing hypotheses and a positive number η:

$$G(X,\{W_{hyp}\},\Lambda,\Gamma) = \log\left(\frac{1}{N}\sum_{r=1}^{N}\exp[g(X,W_{hyp},\Lambda,\Gamma)\eta]\right)^{\frac{1}{\eta}} \tag{2.32}$$

The misclassification measure can be considered as a continuous function of decision rule. $d(X,\Lambda,\Gamma) \le 0$ means a correct decoding and a positive $d(X,\Lambda,\Gamma)$ implies that an error occurs and that the parameters need to be adjusted.

As mentioned above, the objective function is a smoothed loss function that translates the misclassification measure into zero-one domain for gradient optimization. The sigmoid function is an obvious candidate, formulated by:

$$\mathcal{F}_{MCE}(d) = \frac{1}{1+e^{(-\gamma d+\theta)}} \tag{2.33}$$

where γ and θ control the slope and the shift of the sigmoid function respectively. Once the loss function has been formulated, the goal is to find a set of parameters that minimize the loss. In the following, we will investigate the choice of loss function and the optimization method in MCE training.

Property of Sigmoid Function

MCE training, as expressed in (2.33), introduces a sigmoid function as the loss function to define the misclassification measure for error minimization. Parameter update is performed only when the score difference $d(X,\Lambda,\Gamma)$ between the reference and the hypothesis is positive. However, as the

score difference gets larger, the slope of sigmoid function tends to 0 and no parameter adjustment is made. To perform an effective training, an upper bound for the score difference is used. Thus, the usable score difference lies in a certain interval of the sigmoid function.

In addition, the score difference is scaled by γ (θ is usually kept 0) and the derivation of the sigmoid function is $\gamma \cdot \mathcal{F}(d) \cdot (1 - \mathcal{F}(d)) \cdot I$ (I is the difference of number of transition weights which is not affected by γ. Derivation is presented in Section 3.3 and Appendix A), which shows that the slope parameter γ controls the value of the gradient of $\mathcal{F}(d)$. Sigmoid functions with different γ are displayed on Figure 3.1. In Section 3.5.3, we will present our parameter selection in more detail.

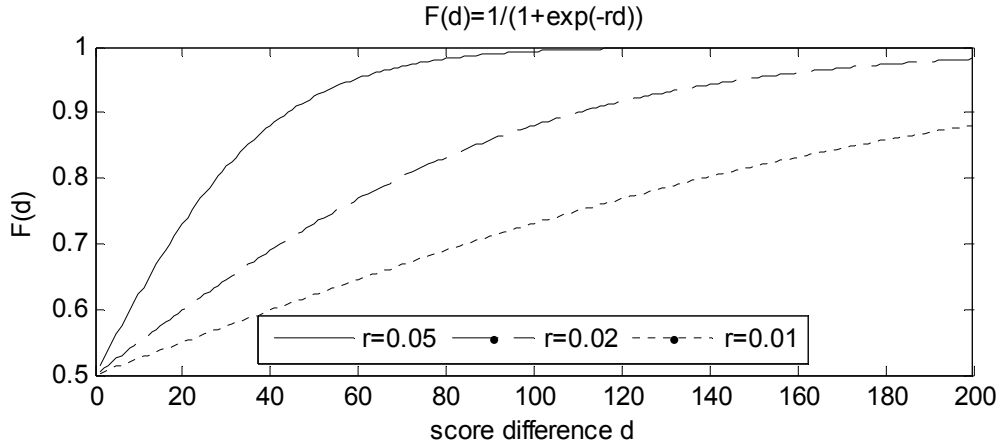

Figure 3.1: Sigmoid functions with different γ.

Choice of a Good Loss Function

For a learning system, the learning behavior is to find the function's global minimum. An ideal choice for the loss function is a convex function. A convex function has the property that the absolute minimum exists and can be reached from any starting point on a function curve or surface. Well-known convex functions are exponential functions and quadratic functions defined on the domain of real numbers. This property makes it straightforward to apply various convex optimization methods. However, the functions encountered in practice may not be convex, such as the sigmoid function. Minimizing such non-convex functions runs the risk to reach a local, rather than a global minimum.

Although various investigations on the performance impact of using different loss functions have been made in the literature (Rosasco *et al.*, 2004; Zhou, 2002), the choice of a good loss function varies widely depending on the data distribution and on the problem design. In the discriminative training framework,

the training performance relies heavily on the loss function and optimization method. A recent study explores several loss functions and optimization methods (Altun *et al.*, 2003). These experiments suggest that a better performance can be obtained by extending the feature space in the model, rather than using a different loss function. These findings have been confirmed in (McDermott and Hazen, 2004).

GPD Optimization Method

The generalized probabilistic descent (GPD) algorithm is a powerful optimization method which is often applied on MCE criterion for minimizing classification error. The GPD algorithm basically performs standard gradient descent method to optimize the parameters (Schluter *et al.*, 2001) but extends the calculation of the gradient for each training sample, which makes it a form of stochastic approximation.

Extension to Stochastic Approximation

Standard gradient descent is an attractive approach due to its simplicity and generality. Assume v, ε and $F(\cdot)$ are a vector of variables, a learning rate (step-size) and a differentiable function respectively. The parameter update on v is expressed as:

$$v_{t+1} = v_t - \varepsilon \nabla F_D(v_t) \tag{2.34}$$

where t is an iteration number and D indicates the whole training data. With a good starting point and a carefully chosen learning rate, gradient descent often converges to a local minimum[12]. However, the update amount of v is the sum of gradients of all training samples. The whole model is updated once by this adjustment at each iteration. Therefore, standard gradient descent is a *batch* training. For discriminative training on a large graph, several transitions are never or rarely traversed, and confusions frequently occur at some positions. It may not be reasonable to update the involved parameters by the scaled gradient $\varepsilon \nabla F_D(v_t)$.

A variant is thus proposed to update the model for each individual training sample. It is known as stochastic gradient descent or stochastic approximation (Widrow and Hoff, 1960). This *on-line* training performs sample-by-sample

[12] Local minimum is guaranteed under stochastic approximation conditions. It requires learning rate selection over time to produce sufficient parameter change. However, satisfying these conditions usually results in slow convergence.

parameter update, and usually converges faster than standard gradient descent (Darken and Moody, 1992; Le Roux and McDermott, 2005; Nekrylova, 1975). The parameter update of on-line training is written as:

$$v_{t+1} = v_t - \varepsilon \nabla F_d(v_t) \tag{2.35}$$

where *d* is a training sample in the whole training set *D*.

Property and Comparison

The main difference between (2.34) and (2.35) is that, at each iteration, standard gradient descent updates the model once while stochastic gradient adjusts the model parameters for each training sample. With a small enough learning rate, stochastic gradient can closely approximate the standard gradient. However, it still runs the risk of being trapped to local minimum, as standard gradient descent does. Global minimum of stochastic gradient descent is not guaranteed. Additional optimization and convergence properties of the stochastic gradient descent are investigated in the literature (Blum, 1954; Spall, 2003). In ASR tasks, the theoretical survey of GPD in error rate minimization has been explained in (Juang *et al.*, 1997).

Despite GPD on MCE yielding significant improvements in many ASR tasks, a common drawback of GPD is the over-fitting effect. This should be carefully handled, especially when relatively few data is available to train a complex model. Several methods have been proposed to prevent the over-fitting such as extending parameter space, cross-validation, early stopping or modified MCE (Biem, 2006; Shimodaira *et al.*, 1998). In general, providing sufficient training data is needed to avoid the over-fitting problem.

Other Optimization Methods

GPD optimization method for error rate minimization in MCE training is not the only choice. Other optimization methods are also possible, such as Broyden-Fletcher-Goldfarb-Shanno (BFGS) method, quasi-Newton's method, conjugate gradient method (Le Roux and McDermott, 2005; McDermott and Katagiri, 1994). In recent work, one has compared the performance of different optimization methods on MCE framework with a large data set (Le Roux and McDermott, 2005). Although results show that MCE can be effectively optimized by various methods, parameter selection is needed to reach better performance. More recently, G. Zweig and his coworkers have used Quickprop, a fast variant of the well known backpropagation algorithm, to update the parameters of a static decoding graph (Kuo *et al.*, 2007).

Discussion

Although MCE incorporates error rate into the objective function, a comparison between MMI and MCE shows that both can be represented in a unifying formula and optimization methods (Schluter *et al.*, 2001; Schluter and Macherey, 1998), where significant improvements in acoustic modeling were observed over ML for both criteria.

3.2.3 Other Training Approaches

In addition to MMI and MCE, several training approaches were conceived recently. These approaches either extend the error rate based idea or introduce a novel problem design.

(1) Minimum Phone Error

One recently proposed training approach uses the minimum phone error (MPE) criterion (Povey and Woodland, 2002) that aims at minimizing the decoded errors at the phone level, which can be expressed as:

$$\mathcal{F}_{MPE}(\lambda) = \sum_{r}^{R} \frac{\sum_s p_\lambda(O_r \mid s)^k P(s) PhoneAcc(s)}{\sum_s p_\lambda(O_r \mid s)^k P(s)} \tag{2.36}$$

where $\{O_1, O_2, ..., O_R\}$ is a sequence of observations with corresponding transcriptions $\{s_r\}$, $P(s)$ is language model probability, λ is a set of HMM parameters, k is a scaling factor of acoustic score and $PhoneAcc(s)$ represents the number of correctly decoded phones in hypothesis s.

It is clear that MPE criterion allows the estimates of decoding errors to be used directly in the training process. Maximizing $\mathcal{F}_{MPE}(\lambda)$ is equivalent to increase the decoding accuracy. It can be considered as minimizing the errors by explicitly modeling the phone accuracy while MCE typically focus on string accuracy (McDermott *et al.*, 2007).

(2) Reranking with Perceptron Algorithm

In recent years, reranking techniques have resulted in significant improvements in various discriminative training tasks (Collins and Duffy, 2002; Collins and Koo, 2000; Och, 2003; Shen *et al.*, 2004). It is performed by extracting new features from the N-best list, and then by using these new features to rerank the N-best list. One popular algorithm employed to perform this reranking is the perceptron algorithm (Rosenblatt, 1958).

This algorithm is simple and is widely used in machine learning community. The idea behind is to incrementally modify the parameters such that the misclassified instances move closer to the correct side of the decision boundary. In ASR tasks, a general perceptron algorithm is proposed in (Collins, 2002). For a set of training samples (x_i, y_i), $i = 1,2,...,n$, the learning function is given by:

$$\mathcal{F}_{Perceptron}(x_i) = z_i = \arg\max_{z \in \mathrm{GEN}(x_i)} \Phi(x_i, z) \cdot \nu \tag{2.37}$$

where $\mathrm{GEN}(x_i)$ is a decoder that produces a set of candidates from an input sample x_i, Φ maps a candidate to a feature vector and ν is a parameter vector. Using the training samples as evidence to adjust the parameters, if $z_i \neq y_i$, the parameter vector is updated by the following rule:

$$\nu = \nu + \Phi(x_i, y_i) - \Phi(x_i, z_i) \tag{2.38}$$

In ASR tasks, the feature of $\Phi(x, y)$ can be defined as the number of times a decoded word (or an *n*-gram, consecutive words) appears in the reference y. The goal of the decoder is to output the best path that maximizes $\mathcal{F}(x)$. It can be seen that the parameter update of perceptron algorithm is performed by feature selection, in an attempt to make the hypothesis and the reference word strings getting closer.

(3) Minimum Sample Risk

Most existent discriminative training methods introduce a loss function in the problem design and attempt to reach the minimum error by optimizing the loss function. However, the loss function is an approximation (or translation) of the decoding errors. An alternative is to directly minimize the error rate on the training samples, a technique called minimum sample risk (MSR) (Gao *et al.*, 2005).

Rather than using a loss function, the MSR employs a simple heuristic training algorithm to minimize the training errors. It is performed by selecting a subset of the most effective features and then iteratively optimizing the model parameters. In ASR tasks, the features can be the occurrence of *n*-grams in a word string *W* or their log probabilities. These features are mapped to a set of real values, in the form of a vector *f*. Each feature has a corresponding parameter. The product of the feature and parameter vector λ can be defined as the score of a word string.

$$score(W, \lambda) = \lambda \cdot f \tag{2.39}$$

Maximizing the score is similar to the perceptron algorithm presented above. MSR further formulates the problem as an error minimization task; both feature selection and parameter adjustment are performed based on the following function:

$$\mathcal{F}_{MSR}(\lambda) = \arg\min_{\lambda} \sum_{i=1}^{N} Er(W_i^R, W_i^*(x_i, \lambda)) \qquad (2.40)$$

where $Er(\cdot)$ is an error function that measures the number of errors by comparing the reference transcript W_i^R with the most likely word string $W^* = \arg\max Score(W, \lambda)$ generated from a training sample (x_i, W_i^R), for $i = 1,2,...,n$. The total number of training errors is defined to be the sample risk. The goal of (2.40) is to find a set of parameters λ that minimizes the risk, by taking one feature at a time, reaching its minimum in that direction and repeating the minimization through all features until the error reduction converges.

It is noticeable that the MSR differs from other discriminative training methods in that the effectiveness of each feature is estimated, instead of being combined in the objective function, where the high correlation of features may result in only a slight improvement (Finette *et al.*, 1983; Gao *et al.*, 2005).

3.3 Problem Formulation

In this thesis, the static decoding graph is constructed from a language model and other sources. Our goal is to enhance the graph quality by updating some of the parameters. With a suitable discriminative training criterion, we hope that a reduction of training errors will yield an improvement of the recognition accuracy on the test set.

We follow the notations and problem design of our previous work (Lin and Yvon, 2005). Assume *G* is an integrated finite-state graph based on the principles described above (additional construction details are given in Section 3.6, p. 58). The graph *G* contains two kinds of parameters: state transition weights and acoustic model parameters, which are associated with HMMs states Gaussian densities. Given a word string *W*, a set of acoustic model Λ, a set of transition weights Γ and an observation sequence *X*, the conditional *log* likelihood of *X* is approximated as the score of the best path in *G* for input *X* and output *W*. This score combines the individual acoustic *log* likelihoods and transition weights along the decoded path, as shown in the following equation:

$$g(X,W,\Lambda,\Gamma) = a(X,W,\Lambda) + b(W,\Gamma) \tag{3.1}$$

where $a(X,W,\Lambda)$ is the sum of acoustic *log* likelihoods and $b(W,\Gamma)$ is the sum of transition weights along the path from the starting state to the final state. Speech decoding consists of finding a word hypothesis W_{hyp} which maximizes g over all possible word sequences. If W_{ref} is the correct word sequence (reference string), the performance of the recognizer can be expressed as a function of the score difference between the reference and the best hypothesis. For a given input vector, the misclassification function thus is defined as:

$$d(X,\Lambda,\Gamma) = -g(X,W_{ref},\Lambda,\Gamma) + g(X,W_{hyp},\Lambda,\Gamma) \tag{3.2}$$

An erroneous recognition hypothesis thus simply translates into a positive value of $d(X,\Lambda,\Gamma)$, meaning that the correct word sequence is not the top ranking one according to g. Once the misclassification function has been formulated, the next step is to define a loss function, $\ell(X,\Lambda,\Gamma)$, which aims at minimizing error rate by integrating the misclassification measure $d(X,\Lambda,\Gamma)$:

$$\ell(X,\Lambda,\Gamma) = \ell(d(X,\Lambda,\Gamma)) = \frac{1}{1+\exp(-\gamma d(X,\Lambda,\Gamma)+\theta)} \tag{3.3}$$

where γ and θ are the parameters which control respectively the slope and the shift factor of the sigmoid function. Using generalized probabilistic descent (GPD) algorithm (Katagiri *et al.*, 1990), a standard iterative procedure can then be defined, based on the following parameter update rule for the set of transition weights:

$$\Gamma_{t+1} = \Gamma_t - \varepsilon \cdot \nabla \ell(X,\Lambda,\Gamma_t) \tag{3.4}$$

meaning that new transition weight Γ_{t+1} is updated by subtracting a scaled gradient from original weight Γ_t. If the acoustic model parameters are fixed, the loss function needs to be differentiated only with respect to the transition weights. Given that $b(W,\Gamma)$ is the sum of transition weights and $a(X,W,\Lambda)$ is considered as a constant, the derivation of (3.4) yields:

$$\nabla \ell_i(X,\Lambda,\Gamma_t) = \frac{\partial \ell_i}{\partial d_i}\frac{\partial d_i(X,\Lambda,\Gamma_t)}{\partial \Gamma} \tag{3.5}$$

Consider Γ as a transition weight vector, taking partial derivatives of $d_i(X,\Lambda,\Gamma)$ with respect to Γ, (3.5) finally yields:

$$\frac{\partial \ell_i}{\partial d_i} = \gamma \ell(d_i)(1-\ell(d_i)) \tag{3.6}$$

$$\frac{\partial d_i(X,\Lambda,\Gamma_t)}{\partial \Gamma} = -I(W_{ref},s) + I(W_{hyp},s) \tag{3.7}$$

where $I(W,s)$ represents the number of transition weight *s* on the best decoding path for *W*. This procedure can be generalized to N-best hypotheses, rather than the single best one (Kuo *et al.*, 2002). Details of the derivation are provided in Appendix A (p. 116).

3.4 Training Procedure

Based on the problem formulation and the MCE training framework discussed in the previous sections, the training procedure is represented as a flowchart displayed on Figure 3.2. The procedure consists of three phases: 1) graph construction 2) discriminative training and 3) testing, which are presented in the following.

Initially, the graph is built by the graph construction procedure presented in Section 3.6. Three knowledge sources, namely the acoustic models, the language model and the dictionary, are combined to produce a single search space, the *graph 0*, for further parameter update.

Then, all training samples are used to update the parameters of *graph 0* by discriminative training and produce *graph 1.* In the discriminative training phase, the *graph 1* is taken as input to produce *graph 2* in the next training iteration. One has to note that the graph parameters are updated sequentially. The updating includes three steps: decoding, alignment and transition weight adjustment on the word pairs. At each iteration, decoding, alignment and parameter updating are recorded in a log file for further analysis.

In the testing phase, the updated graph is used for decoding on the test set to evaluate the performance improvement incurred by the training procedure.

The training loop is repeated until the desired number of training iterations is reached.

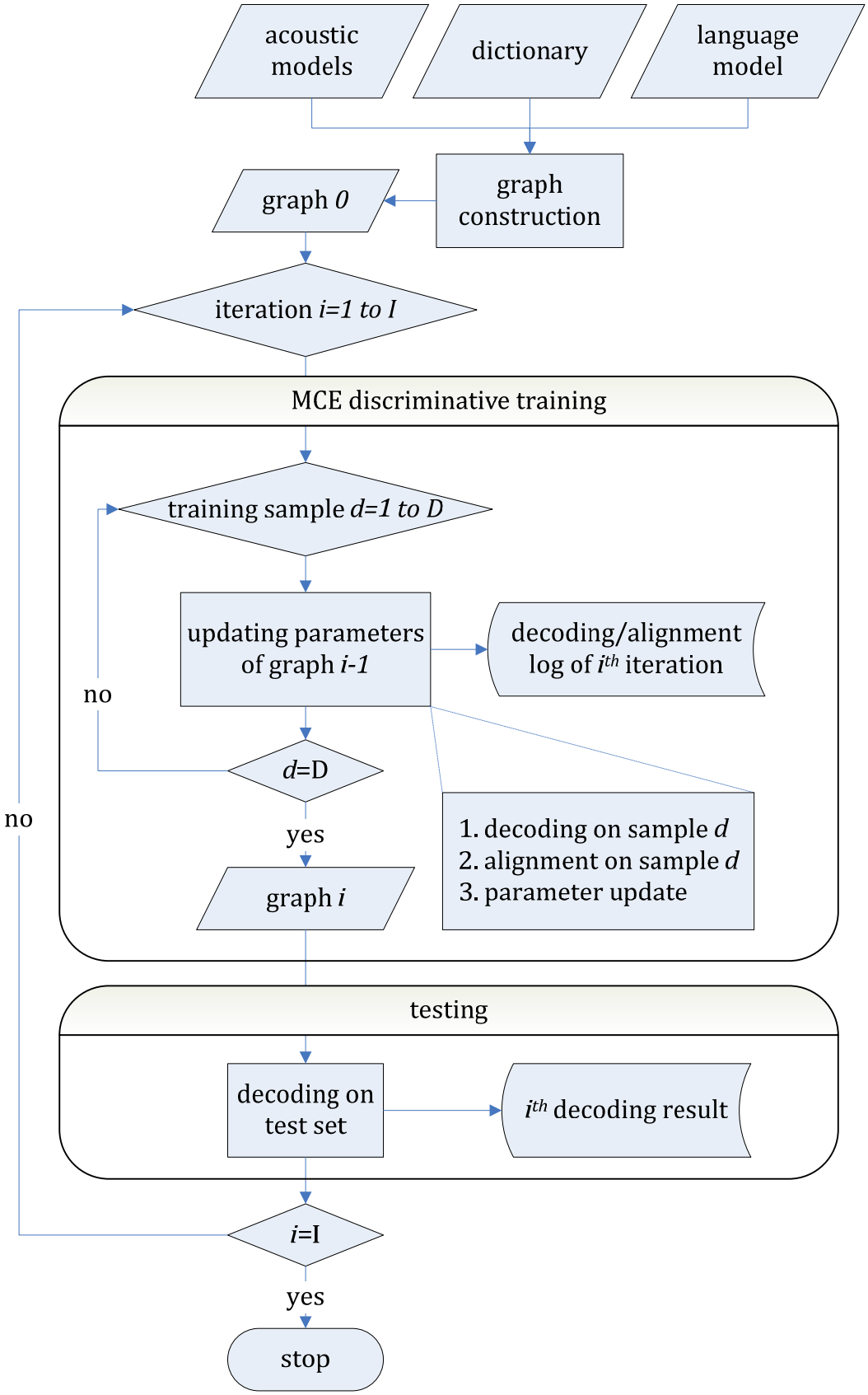

Figure 3.2: Flowchart of MCE discriminative training on integrated decoding graph. The I and D are training iterations and size of data set respectively.

3.5 Parameter Update Rule

In MCE training, the parameters that have to be carefully selected are γ and ε, representing the slope of sigmoid function that transfers a misclassification measure to the 0-1 domain, and a scale factor to update the parameters. Moreover, training is performed on a weighted finite-state graph. A word is represented by a sequence of phones and redundant arcs are determinized to reduce the graph complexity. Therefore, the following two issues remain critical:

- The choice of a word pair from the reference and hypothesis word strings.
- The update position, whose choice is dependent on the distribution of transition weights.

A general parameter update rule for MCE training on integrated decoding graph is given by (3.4). Additional implementation details will be discussed from Section 3.5.1 to Section 3.5.3.

3.5.1 Choice of a Word Pair

When decoding or alignment is performed on a graph, the result is a sequence of arcs starting from the initial state to the terminal state, yielding the corresponding word string. Intuitively, if the reference and the hypothesis word strings are identical, the hypothesis is considered as a correct decoding. Even though the transition arcs may be different, the difference occurs in the transition going through an optional silence or through the back-off state. Based on this discussion, when decoding and alignment produce word strings and the corresponding transition paths, parameter update is determined by the comparison of word strings. If the reference and hypothesis word strings are the same, no parameter update is performed. Otherwise, parameter update is performed on the mismatched words. If the probability of a mismatched word has to be changed, only one arc along the corresponding path is selected for actual transition weight adjustment.

When updating the transition weights of mismatched words, one notable issue is that a correctly decoded word may follow an incorrect word, meaning that parameter update is based on an incorrect word history. Although we want to fix the errors in the word history, decreasing this weight may be harmful to the correct words (Kuo *et al.*, 2002). We thus focus on the relationship between two

consecutive words rather than considering a certain word history. Assume that, for instance, the reference and hypothesis word strings and are the following:

hypothesis: <s> franck mathevon </s>
reference: <s> de franck mathevon </s>

The set of word pairs are obtained by extracting consecutive words:

"<s> franck"
"franck mathevon"
"mathevon </s>"
"<s> de"
"de franck"

Given the set of consecutive word pairs, parameter update is performed according to the following rule:

- If a word pair exists both in the reference and in the hypothesis, with identical occurrence counts, no update is performed.
- If a word pair exists both in the reference and the hypothesis with different occurrence counts, parameter update is performed on this word pair.
- If a word pair exists only in the reference or only in the hypothesis, parameter update is performed on this word pair.

Since a word pair involves a sequence of arcs, which may be shared with other pairs, the choice of the updated arc has a great influence on the overall transition weight distribution. In the next section, we will investigate this issue to propose a suitable parameter update.

3.5.2 Update Position

In our implementation of MCE training, a word pair W_1W_2 is defined as a sequence of arcs. The first arc outputs W_1 and the last arc outputs W_2. As illustrated in Figure 3.3, W_1W_2 is represented by the following transitions:

$$W_1W_2 = \{a_2, a_3, a_4, a_5\} \tag{3.8}$$

Along this sequence, arc a_2 is the first arc of W_1W_2 and is shared with the word pair W_1W_3. If parameter update is always performed on the first arc, the adjustment on arc a_2 also influences the weight distribution on W_1W_3. This

dependency makes different word pairs less distinguishable and results in unstable performance improvement, especially when language model look-ahead technique is applied to increase pruning. This dependency also occurs when updating the last arc. This makes the choice of the update position problematic, particularly for word pairs containing words with only one phone such as the preposition *à* and the determiner *l'*. Due to the high influence of transition weights, a deterministic arc selection scheme would result in local instability. In our implementation, the arc for parameter update is randomly determined: one possible arc is chosen by random sampling with uniform probability amongst all possible candidates.

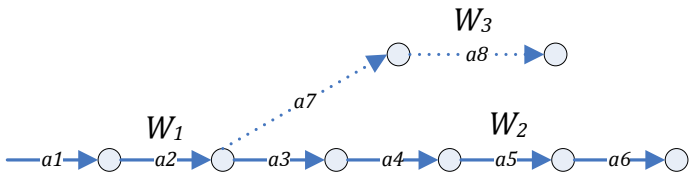

Figure 3.3: An example of decoded paths represented by solid lines. The dotted lines are possible paths in the graph. $\{a_1, a_2, ..., a_8\}$ *are weighted transitions and* $\{W_1, W_2, W_3\}$ *are output word labels.*

One has to note that the arc sequence may contain the back-off arc. Following the derivation in Section 3.3, the back-off arc is not treated as a special case but as a regular transition. Parameter update on the back-off arc is performed as on the other arcs.

3.5.3 Parameter Selection

As discussed in Section 3.2.2, parameters γ and ε have to be carefully selected. In the following, we will address the issue of parameter selection in terms of the distribution of transition weight adjustment.

γ : *The Slope and Gradient Control*

(3.3) shows how a loss function transfers the misclassification measure, where the slope is controlled by γ. Parameter update is then performed by taking the gradient of the loss function. The value of γ is determined by relating the effect on the parameter adjustment[13] with respect to the score difference.

[13] The actual transition weight is calculated by (3.4). However, the term on the right side of (3.5) is a difference of transition weight occurrence and is not affected by the value γ. The discussion thus focuses on the value given by (3.6).

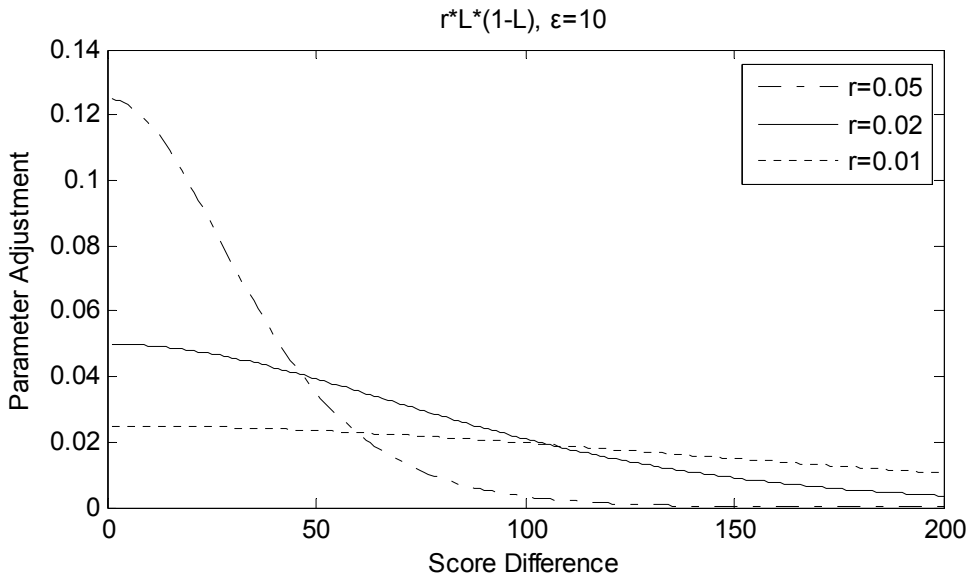

Figure 3.4: Parameter adjustment with respect to score difference in (0,200) where 200 is the upper bound. L is the loss function and the learning rate ε=10.

Figure 3.4 shows that different slope gives distinct update distribution. For $\gamma = 0.05$, the distribution prefers the sentence whose score difference is small, meaning that word pairs with small score difference will obtain more parameter adjustment than those of large score difference. This is reasonable since small score difference indicates that the hypothesis is close to the reference, meaning that either the decoded word string is almost correct or that the errors can be easily fixed. Giving more parameter adjustment will quickly correct the errors. Contrarily, parameter update for score difference greater than 120 almost does nothing. For $\gamma = 0.01$, the training process gives similar parameter change for all sentences, no matter whether the word strings are correctly decoded or not. A reasonable value for this parameter is the following: the slope control $\gamma = 0.02$ with a learning rate $\varepsilon = 10$, giving minimum 0.0035 and maximum 0.05 parameter adjustments in the *log* scale.

ε : The Learning Rate

The learning rate is a factor which scales the gradient of the loss to determine the actual update amount of a transition weight. Generally, low learning rates yield more stable convergence rate while at the cost of longer training time. Using higher learning rate to speed up convergence must be carefully designed to prevent "skipping" local minimum. Due to the sample-by-sample nature of MCE training using the GPD optimization method, one possibility to give an effective parameter update while keeping stable convergence is to use a dynamic learning rate (Magoulas *et al.*, 1999; Yu *et al.*, 1995).

In the MCE training framework, the line-search technique (Mor'e and Thuente, 1994) is applied to determine the update for each training sample, under Armijo condition (Armijo, 1966; Driessen *et al.*, 1998; Luenberger, 1989):

$$f(\Gamma_k + \varepsilon_k p_k) \le f(\Gamma_k) + \mu\varepsilon_k \nabla f(\Gamma_k) p_k \tag{3.9}$$

where $\mu \in (0,1)$, $f(\cdot)$ is the loss function. The parameter is updated by:

$$\Gamma_{k,t+1} = \Gamma_{k,t} + \varepsilon_k p_k \tag{3.10}$$

where Γ_k is the parameter of a training sample *k*, ε is the learning rate (or the step-size of individual sample) and *p* is the descent direction. The computation of the learning rate and of the descent direction may vary depending on the method. In our implementation, the descent direction *p* for training sample *k* is expressed as:

$$p_k = -\nabla f(\Gamma_k) \tag{3.11}$$

and ε is calculated by the following procedure:

until $f(\Gamma_k + \varepsilon^{(j)} p_k) \le f(\Gamma_k) + \mu\varepsilon^{(j)} \nabla f(\Gamma_k) p_k$ {
$\varepsilon^{(j+1)} = \tau\varepsilon^{(j)}$
$j = j + 1$
}
$\varepsilon_k = \varepsilon^{(j)}$

Figure 3.5: Search for the best learning rate.

where $\tau \in (0,1)$ and *j* is an iteration index. At each iteration, a new ε is used to verify whether the condition still holds or not. In practice, satisfying the stop condition may require a lot of iterations, most of which will produce negligible difference on the learning rate. Therefore, a tolerance threshold is often used as a stop condition to limit the computation. With the definition of p_k in (3.11) and the procedure for finding the learning rate, the parameter update rule using the line-search method can be expressed as:

$$\Gamma_{k,t+1} = \Gamma_{k,t} - \varepsilon_k \nabla f(\Gamma_k) \tag{3.12}$$

It is similar to (3.4), except that the learning rate is determined dynamically for each training sample.

3.6 Graph Construction

Fast decoding on weighted finite-state graph critically depends on the details of graph construction, which greatly affect the search efficiency. The graph is built by compiling knowledge sources from typical ASR systems as finite state machines (FSM), then by integrating these sources into one search space. Three main knowledge sources in speech recognition are the **dictionary**, the **acoustic models** and the **language model**. The use of FSM in ASR tasks, the design principle and the implementation details of this finite-state graph are presented in the following sections.

3.6.1 Weighted Finite-State Transducers

Weighted Finite-State Transducers in ASR

Speech decoding involves the process of mapping from a pronunciation sequence to word(s), and from word(s) to high level linguistic representations, which can be viewed as a recognition cascade presented in Section 2.3.3 (p. 33). Many ASR studies generally follow this design but further combine several large knowledge sources to produce higher decoding accuracy. Although more information can be captured, the size of sources becomes the main limitation of fast decoding. This motivates the use of an efficient representation of these models in large vocabulary ASR tasks.

One recent development introduces weighed finite-state transducers (WFST) in ASR tasks (Mohri, 1997; Mohri *et al.*, 2000b). A WFST is a type of finite state machine, whose state transitions carry the input, output symbols and arbitrary weights. It is clear that a transition sequence from the initial state to the final state of a WFST represents a weighted path and the mapping from inputs to the corresponding outputs. A special case of WFST is the weighted finite-state acceptor (WFSA). It can be considered as a WFST without output symbols. The state transitions of a WFSA specify the symbol and a weight, which makes it a proper choice for representing language models.

Both for transducers and acceptors, the epsilon transitions are often introduced to represent an empty string or a delay. Generally, the use of epsilon transitions is critical to the efficiency of finite-state graph. We will investigate this issue later.

Operations on Transducers

Once the various knowledge sources are represented as WFSTs for ASR tasks, two main issues are 1) how the mapping between different information sources can be represented, and 2) the possibility to reduce the transducer to a minimal size. In the following, we will present several transducer operations, together with a discussion of these critical issues.

(1) Composition

The composition operation performs the combination of two transducers $R:\Sigma^* \to \Gamma^*$ and $S:\Gamma^* \to \Delta^*$ to produce a single transducer $T = R \circ S : \sum^* \to \Delta^*$. (The Σ^* and Γ^* are strings over the set of alphabets Σ and Γ). Assume that the corresponding sets of states are $\{r\}$, $\{s\}$ and $\{t\}$ respectively, the relation of symbol mapping represented by the transducer composition is expressed as the following (Mohri *et al.*, 2000a; Pereira and Riley, 1997):

$$(R \circ S)(r,t) = \sum_{s \in \Gamma^*} R(r,s) \times S(s,t) \tag{3.13}$$

where the weight of transition $r \to t$ is a product of weights w1 and w2 over transitions $r \to s$ and $s \to t$ respectively. Obviously, the time and space complexity of this operation is quadratic with the combined WFSTs. It can be seen that if R represents the mapping sequence from A to B, and S represents the mapping from B to C, the composition produces a transducer representing the mapping from A to C. When constructing transducers from various sources, the composition allows different mappings to be combined in a single transducer. For instance, the composition of the transducers constructed from a dictionary and a word string yields a transducer whose output is identical to the word string while the connections between states are phone-level transitions.

A WFST can be considered as a generalization of WFSA. When the composition is performed on a WFSA and a WFST, the acceptor is converted to a transducer with identical input and output symbols. In addition, the epsilon transitions of WFST represent the empty mappings. When the composition is performed, the epsilon transitions should be carefully dealt with, so that the resulting transducer would not contain redundant paths. We will come back to this issue at the end of this section.

(2) Determinization

In many real applications, the constructed transducers are not deterministic. A transducer is deterministic if and only if the transitions leaving a state carry

different input symbols and if there is no input epsilon transitions (Mohri *et al.*, 2000a). Determinization is an important finite-state transducer optimization technique that has several special features (Mohri *et al.*, 2000b; Mohri and Riley, 1997).

- It aims at reducing the transitions of a state to a minimum, such that there are no two transitions sharing the same input symbols.
- It is applied directly to the whole transducer, rather than the redundancy reduction on partial lexical trees such as the word graph approach (Ortmanns *et al.*, 1996c).
- The deterministic transducer is equivalent to the non-deterministic one, meaning that the functionality is not changed after determinization.

Once the determinization is performed on a transducer, one critical issue is how the weights should be distributed when eliminating redundant transitions. For instance, merging arcs with different weights should preserve the sum of weights along the path. In (Mohri, 1997), a weighted determinization algorithm was proposed, which generalized the classical determinization on automata. Although not all weighted transducers can be determinized (Mohri, 1997), for many ASR tasks whose transducer representation is acyclic, the transducer is determinizable. The details of weight distribution over the transducer will be discussed later.

Epsilon transitions can also be a source of problems, particularly for the representation of *n*-gram language models; where epsilon transitions are used to reach the back-off state. Determinization of such finite-state graphs would result in an exponential blow-up of the graph size due to the subset construction (Aho *et al.*, 1986). As a consequence, the epsilon is treated as a regular symbol. This trick allows to make a compromise between the graph size and the determinism. The resulting graph still satisfies the definition of determinism (Mohri, 1997; Mohri and Riley, 1999) in that further composition will not produce multiple matches. The reason why we do not remove the epsilon transitions to the back-off state will be explained at the end of this section.

With the determinization, the time efficiency of a transducer can be greatly improved by reducing the alternatives of a state. Although the worst case complexity of determinization is exponential (Mohri, 1997), this rarely occurs in usual ASR tasks, since the initial transducers generally contain a lot of redundancy (Mohri, 1997).

(3) Minimization

Once a deterministic transducer has been constructed, one can further reduce its size by minimization (Mohri, 1997; Mohri, 2000; Mohri and Riley, 1997). The determinization only produces a transducer in which there is at most one transition with given input symbol. However, there may be several states in the deterministic transducer such that the set of strings from these states to the terminal state are identical. These states are said to be equivalent (Mohri, 2000), meaning that they can be further merged by the minimization procedure without changing the transducer's behavior. This procedure requires to move the output symbols or to redistribute the weights of a deterministic transducer to yield the one having smaller number of states. If there exists no other deterministic transducers having fewer states, this transducer is minimal (Mohri, 2000).

The minimization of weighted deterministic transducers is performed by extending the classical minimization algorithm, which considers the weighted transducer as an unweighted acceptor by taking the input, output symbol and the weight of a transition as a single label (Mohri, 2000; Mohri and Riley, 1997). Before applying the classical minimization algorithm, one important step is to redistribute the weights along the path (Mohri, 1997). This does not affect the transducer's behavior but it makes it more likely to produce a transducer with a smaller size. It is obvious that arbitrary weight distribution may yield several weighted transducers which are all minimal, meaning that the minimal weighted transducer is not unique. In ASR tasks, however, how the weights are distributed along the path has a critical impact on the performance. We will investigate this issue again.

For the case of weighted transducers, the determinization increases the time efficiency by reducing the alternative paths leaving a state. The minimization further reduces the size by merging equivalent states. The complexity of minimization is $O(|Q|+|E|)$ in acyclic transducers and $O(|E|\log|Q|)$ in the general case (Mohri, 1997), where $|Q|$ and $|E|$ respectively represent the number of states and the number of transitions.

(4) Weight Distribution

As noted previously, the weight distribution is not clearly specified in the determinization and minimization procedures. However, for many ASR systems which critically rely on the weights (language model probability, HMM state transition probability or word variants) to improve the pruning, the decoding performance could be improved if the weights are properly distributed (Ortmanns *et al.*, 1997a).

The motivation of weight pushing on transducers follows this intuition and pushes the weights towards the initial state as much as possible to improve pruning. Two general weight pushing techniques were proposed in (Mohri and Riley, 2001), called *log semiring* and *tropical semiring* algorithms. A semiring is an algebraic structure $(\mathbb{R},\oplus,\otimes)$ over a set of real numbers, which is similar to a ring without additive inverse (no negative elements). For instance, $(\mathbb{N},+,\times)$ is a semiring over natural numbers, with ordinary addition (identity 0) and multiplication (identity 1). In the *log* domain, the semiring can thus be expressed as below:

$$(\mathbb{R}_+ \cup \{\infty\},\oplus,+) \tag{3.14}$$

where + is an ordinary addition of two log probabilities $x, y \in \mathbb{R}_+ \cup \{\infty\}$. For the log semiring, the $\oplus$ is given as:

$$x \oplus y = -\log(e^{-x} + e^{-y}) \tag{3.15}$$

In ASR tasks where Viterbi approximation (finding the path with minimum weights) is applied, the $\oplus$ for tropical semiring is defined by:

$$x \oplus y = \min(x, y) \tag{3.16}$$

When weight pushing is performed, both the log weight pushing algorithms attempt to push the weights towards the initial state. The difference is that the log semiring algorithm redistributes the weight of each transition such that outgoing probabilities of a state sum to 1 while the tropical semiring algorithm keeps the minimum over the transitions. Although tropical semiring algorithm outperforms log semiring algorithm in (Mohri and Riley, 2001), another investigation shows an opposite result when language model probability is not pushed beyond word boundaries (Kanthak *et al.*, 2002).

One has to note that both semiring approaches yield equivalent transducers in the minimization step (Mohri and Riley, 2001). They only differ in the way weight is redistributed on the path.

Epsilon Removal

As mentioned previously, the epsilon transitions are often introduced in the transducers, representing the empty string or a delay. The process to remove the epsilon transitions from a transducer to produce an equivalent transducer without epsilon transitions is called epsilon-removal. One generic epsilon-removal algorithm was proposed in (Mohri, 2002), which performs efficient removal and makes it an independent part of transducer optimization.

In practice, the transducer representation of *n*-gram language models uses epsilon transitions connecting to the back-off state with back-off probabilities

(Allauzen *et al.*, 2003). In this connected network, these epsilon transitions can be freely accessed during decoding. However, removing these transitions is not desirable. For a 65k words vocabulary, removing epsilon transitions to the back-off state would increase the number of states needed to represent the back-off mechanism from $(65\text{k} + 65\text{k}+1)$ to $(65\text{k} \times 65\text{k})$. This would result in an intractable transducer.

3.6.2 Principle

In (2.23) (p. 35), a general composition rule $H \circ C \circ L \circ G$ is proposed, which allows all resources to be integrated into one single decoding graph. However, depending on the problem design, various WFST operations and modifications may be introduced to produce a more efficient decoding graph. In this thesis, the goal of graph construction is to build an integrated search space which combines knowledge sources from the dictionary, the tied-state tri-phone acoustic models and an *n*-gram back-off language model. Three additional considerations are taken into account in our graph construction procedure:

- Optional silence: at the end of a word, a short pause is added to allow for an optional silence after each word.
- Within-word acoustic models: in this decoding graph, tri-phone models are used on the transitions within a word. Transitions which cross a word boundary use context-independent models.
- Word insertion penalty: a word insertion penalty is used to penalize or to reward the decoding hypothesis when a word label is produced.

The finite-state operations and knowledge sources used in the graph construction procedure are defined as below:

Notation	Description
$\circ$	composition of two source graphs into one graph
det	determinization of a graph
G	FSA representation of the language model
L'	unambiguous dictionary FST
C'	inverse of determinized CI to CD FST
H	FST mapping context-dependent phone to HMM
wip	add word insertion penalty
π	removal of epsilon transitions <*eps*>
pFST	graph weight or label pushing
sp	incorporation of a short pause

Table 3.1: WFSTs and operations for graph construction.

Taking into account these three considerations, the graph construction procedure can be expressed in the following manner:

$$\text{sp}(\text{pFST}(\det(H \circ (\det(C' \circ (\det(\text{pFST}(\text{wip}(\pi(L' \circ G)))))))))) \quad (3.17)$$

Three aspects of the implementation of (3.17) will be explained below: (1) WFST representation of knowledge sources (2) graph operations and (3) weight pushing.

3.6.3 WFST Representation

Language Model

Language models are constructed by estimating the probability between words. The *n*-gram models are the most popular in ASR tasks. Each *n*-gram history represents a state, and the language model probabilities are distributed over the arcs connecting different states, admitting the natural representation by WFSAs (Allauzen *et al.*, 2003). By combining all the transitions between words and word histories together, a connected network can thus be constructed, starting from the state *<s>* and ending at the terminal state *</s>* as shown in Figure 3.6. The source 3-gram language model and its finite-state representation are presented in Appendix B (p. 120).

In Figure 3.6, state 0 represents the history *<s>*, which is the beginning of the graph. State 12 is the terminal state, represented by a double circle. Overall, there are three kinds of states.

- The back-off state: the state 1 in the figure.
- The state representing a 1-word history: the states such as 2, 3, 4, 5 and 6 represent the words (or unigram history in the language model). This kind of state involves a transition from current state to the back-off state and a transition from the back-off state to current state, and carries back-off probability and unigram probability respectively.
- The state representing a word history of length 2: the states such as 7, 8, 9, 10 and 11 are such states. The 3-gram *log* probabilities are distributed over the transitions between these states. In addition, these states also have transitions to lower order *n*-grams with back-off probability, such as the transition from state 7 to state 3.

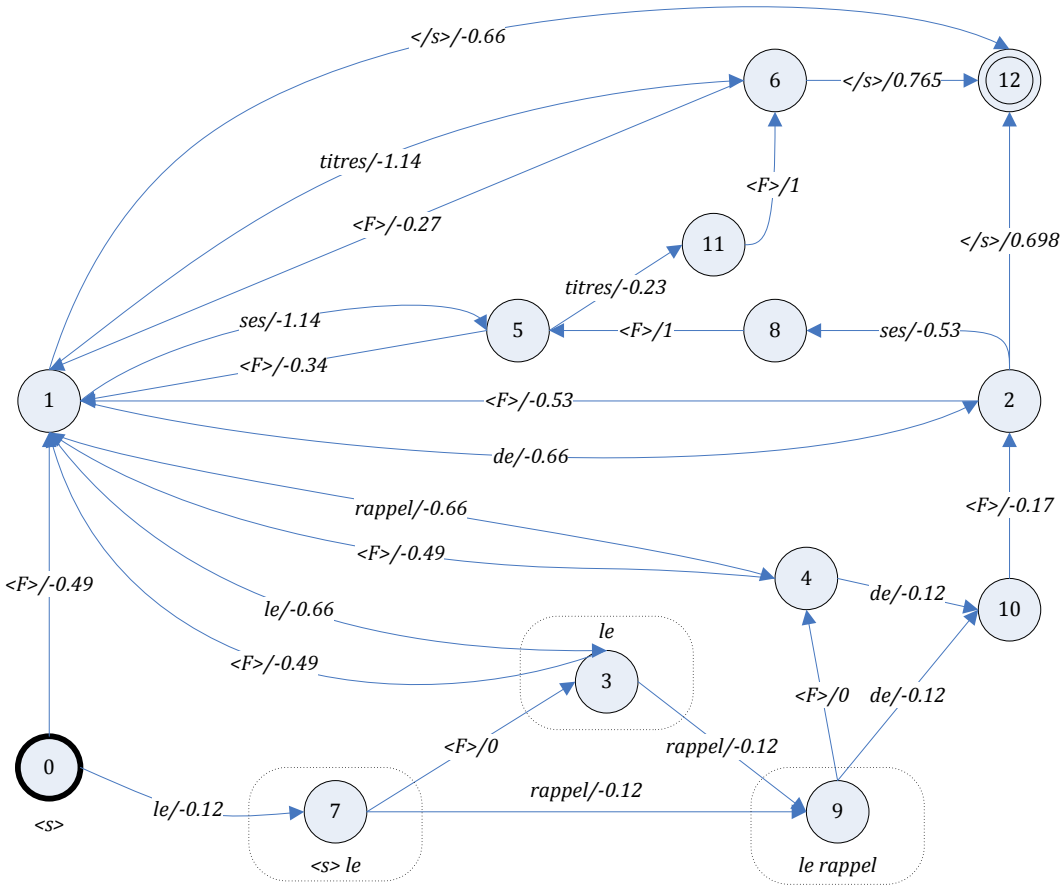

Figure 3.6: A 3-gram language model in the form of a weighted finite-state acceptor. The dotted rectangles are examples of 1-gram (state 3) and 2-gram histories (state 7 and 9). Initial state is 0 and final state is 12.

As mentioned in Section 3.6.1, the epsilons are no longer delays for determinization on transducers; the label from a state to the back-off state is represented by *<F>* so as to be treated as a regular symbol instead of the reserved label *<eps>*.

Once the language model is constructed as a WFSA *G*, we attempt to expand the graph at the lower levels by composition and determinization, such that more alternatives can be reduced to yield a more efficient graph. To achieve this, we use the dictionary as another source for composition.

Dictionary

A pronunciation dictionary is a list of mappings between phone sequence(s) and words. In a large scale dictionary, there often exists identical pronunciation that

corresponds to different words (homophones). This ambiguity makes the resulting graph $L \circ G$ non-deterministic after composing the dictionary L and the language model G. Thus, auxiliary symbols are introduced to distinguish the pronunciations (Mohri *et al.*, 2000a; Mohri and Riley, 1999), yielding an unambiguous dictionary L'.

In our application, the dictionary contains 118857 entries, including 65001 words and their pronunciation variants. The first step towards representing the dictionary as a finite-state graph is to disambiguate all phone sequences by introducing auxiliary symbols. These auxiliary symbols ensure that each phone sequence is unique in the dictionary. In French, the ambiguity frequently occurs in the conjugation, and among singular and plural nouns, as shown in the following examples. The words having the same pronunciation are distinguished by adding #n at the end of phone sequence, where n is an arbitrary index.

mange	m	a~	Z	#0
manges	m	a~	Z	#1
mangent	m	a~	Z	#2
lait	l	E	#0	
laits	l	E	#1	

Figure 3.7: Disambiguation of pronunciation sequence by introducing auxiliary symbols #n.

After disambiguating the dictionary, the end of a phone sequence is either a phone (except the short pause "#" and long silence "##") or an auxiliary symbol. Then, the second step is performed on the pronunciations ending with a phone. In this case, an auxiliary symbol is added at the end of phone sequence. Since the pronunciation ending with a phone implies that the phone sequence is unique in the dictionary, adding an auxiliary symbol still keeps the dictionary unambiguous. This second step is not theoretically motivated, but stems from practical considerations. With the use of auxiliary symbol, any phone sequence of a word is preceded and followed by auxiliary symbols which play the role of word boundary markers and help keeping track of transition weight distributions between words or variants.

An example of pronunciation dictionary transducer is displayed in Figure 3.8, using the words listed in Figure 3.7. The word label is placed in the beginning of the phone sequence to optimize transducer composition. In our application, the weights are not used in the pronunciation dictionary.

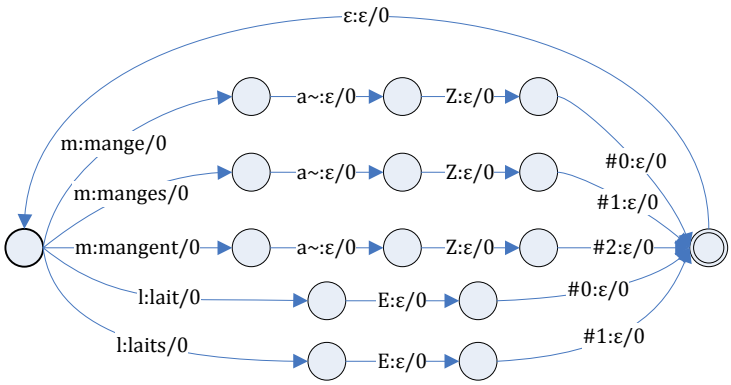

Figure 3.8: Transducer of the pronunciation dictionary.

Acoustic Models

Acoustic models are not directly integrated in the graph, especially when the acoustic models contain a large set of tri-phones. The incorporation of acoustic models is realized by two FSTs: 1) the FST C' representing the mapping from context-independent (CI) to context-dependent (CD) phones and 2) the FST which maps CD phones to tri-phones. In this graph construction, the CD phones refer to logical phones and many of them may link to the same tri-phones. The tri-phones are physical models that actually exist in the acoustic model set.

(1) CICD FST

To convert a given CI phone transition to the corresponding CD phone transition, one has to look at its preceding and following CI phones to produce the right CD phone (for context of length 2) (Chen, 2003). Conceptually, if CI phones are available, the set of all CD phones can be obtained by taking any three of them as a combination, representing the left context, middle phone and right context. Any consecutive three CI phones that can be found in the dictionary is an element in the set of CD phones.

Therefore, for a machine containing all CI phones and CD phones, if it takes a CI phone as input and outputs a CD phone by looking its preceding and succeeding CI phones, the machine can be used as a mapping to convert a CI-phone graph to a CD-phone based network. This machine is called CICD FST. This FST can be build by all possible CI phone combinations, except *<eps>* which indicates null or epsilon transition and is used only as the left and the right context. Figure 3.9 shows a part of the transitions in the CICD FST.

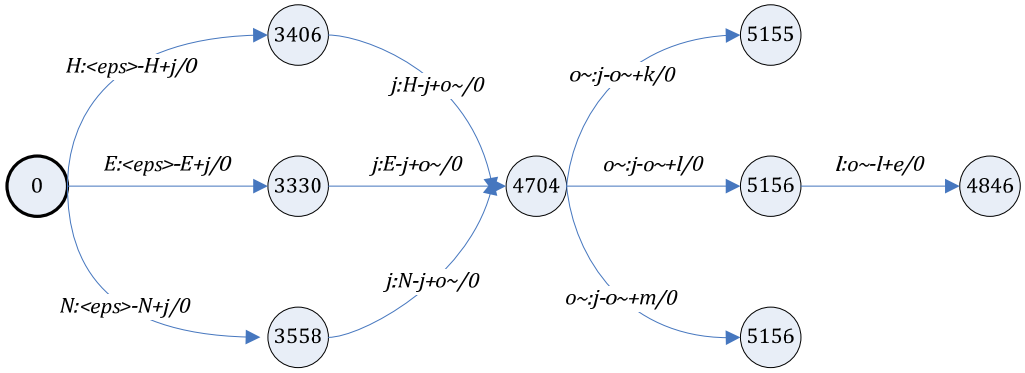

Figure 3.9: Graph representation of CI to CD phone mapping. The arc carries the CI phone input, the CD phone output and a null transition weight.

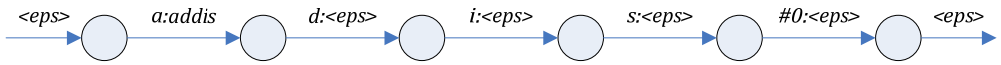

Figure 3.10: CI phone representation of "addis".

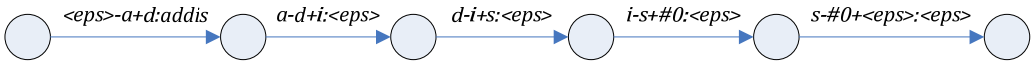

Figure 3.11: CD phone representation of "addis".

When the CICD FST is built, by composing CICD FST with CI graph, CI phone transitions are converted to CD phone transitions, as illustrated in Figure 3.10 and Figure 3.11.

(2) Logical Model to Physical Model

When the disambiguated dictionary L' is composed with the language model G, the graph $L' \circ G$ carries CI phones over the arcs. By further composing $L' \circ G$ with the CICD machine, the input labels become CD phones. For small vocabulary ASR task, each CD phone is an individual HMM whose parameters are estimated from the training data. However, if numerous CD phones are used, it is difficult to collect adequate data to perform reliable parameter estimation for each individual unit/model. Some of the models thus share the same acoustic states or data according to a decision tree. If several unseen new models are created, they can be quickly synthesized by state tying without starting a new training process. Therefore, the CD phones are often called logical models which logically represent the context relationships. The models containing actual states and mixtures for likelihood computation are physical models.

The mapping from logical models to physical models can created by *HHEd* (Young *et al*., 2002) with a decision tree. If a logical model name cannot be found in the set of physical models, the logical model is synthesized from the decision tree. Otherwise, the logical model is itself a physical model. By scanning all the logical models, this decision yields the list of logical-physical model mappings.

In our graph construction procedure, the auxiliary symbols are used only for disambiguating the pronunciation sequences. When a CI phone graph is converted to a CD phone graph, several CD phones may be created, which differ only by the auxiliary symbol and result in poor determinization. In fact, these auxiliary symbols are no longer used after disambiguating the dictionary. Removing these symbols on the CI-phone network would yield a better determinization of the CD-phone graph. This can be realized by replacing transitions containing auxiliary symbols with epsilon transitions *<eps>* and by performing epsilon removal. One has to make sure that there are no other *<eps>* transitions before replacing auxiliary symbols, especially the transitions to back-off state of the language model.

Another approach is to build an augmented CICD FST which maps auxiliary symbols to auxiliary CD phones by a self-loop. Additional distribution name must be added to the machine *H* that represents the mapping from CD phones to physical models. In this implementation, auxiliary symbols are not removed on CI-phone graph and are treated as regular ones. The use of auxiliary symbols in graph construction guarantees the determinizability of graph after each composition (Mohri and Riley, 1999).

Our approach follows the concept of keeping auxiliary symbols for determinizability and further incorporates a mapping rule. When the CD phones on the graph are to be converted to HMMs, we 1) perform a simple matching according to the rule and 2) map the CD phones to HMMs. The advantage of using this rule is that it allows a flexible model mapping for various purposes, such as the removal of auxiliary symbols, the use of within-word models or cross-word models at the word boundary.

Figure 3.12 shows an example of our CD-phone processing rule. If the left or the right context is an auxiliary symbol, the auxiliary symbol part is truncated. For instance, the transition *i-s+#0* in Figure 3.11 will become *i-s* by applying the mapping rule. If the middle context of CD phone is an auxiliary symbol such as the last transition *s-#0+<eps>* in Figure 3.11, it is replaced by an epsilon transition. Otherwise, the CD phone is not changed. By applying this rule, CD

phones are converted to logical models, and then to the corresponding physical models. Since several CD phones may be converted to the same physical model, graph determinization can be performed again to eliminate the redundant paths, making the search space smaller.

Given a CD phone $p_1 - p_2 + p_3$ and a set of auxiliary symbols *AUX*:

if $(p_1$ *or* $p_3) \in AUX$
    truncate *AUX* symbol
if $p_2 \in AUX$
    $p_1 - p_2 + p_3 \leftarrow < eps >$
else
    $p_1 - p_2 + p_3$ unchanged

Figure 3.12: The CD-phone processing rule for a graph using within-word models.

3.6.4 Graph Operations

Composition and Determinization

When the knowledge sources are represented as FST graphs, two main operations are used to combine them: graph composition and determinization. The first combines two different sources into a single network and the second reduces the graph complexity by eliminating redundant arcs. In our graph construction procedure, the AT&T's tools[14] for finite state machine operations, the standard *fsmcompose* and *fsmdeterminize*, are used to accomplish the task. As mentioned previously, homophones in the pronunciation dictionary have to be disambiguated to make the composition result determinizable.

Adding the Word Insertion Penalty

One commonly used parameter for better decoding performance is the word insertion penalty (WIP). In typical ASR systems, the WIP can be selected independent of other sources. In an integrated graph, however, the use of WIP affects greatly the transition weight distribution, which is critical for good performance. In practice, the WIP can be added on the graph when graph construction is finished. This has the benefit to allow the training of WIP, but the added cost is not re-distributed by the weight pushing procedure. Direct shift of probability distribution may deteriorate the performance. Thus, our graph construction procedure adds the WIP on the graph $L' \circ G$ prior to any transition

[14] AT&T FSM Library™, http://www.research.att.com/~fsmtools/fsm/

weight adjustment (refer to (3.17)). The resulting graph is a network whose probability distribution is shifted by WIP.

Integration of Optional Silence

The use of an optional silence is integrated by a small program. It searches the graph for arcs that produce a word label and then creates two additional arcs for the optional silence. The transition weight is placed over the first arc for better pruning and the word label is left on the second arc as the mark of a word boundary.

Moreover, the integration of optional silence is performed off-line by sequentially scanning the graph transitions. An example of this procedure is displayed in Figure 3.13 and Figure 3.14. In our graph construction procedure, the optional silence is a one-state acoustic model # which represents the short pause between words, instead of the three-state model ## for long silence.

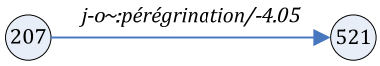

Figure 3.13: A transition that outputs a word label.

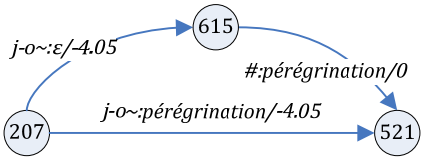

Figure 3.14: Integration of an optional silence.

3.6.5 Weight Pushing

Language Model Look-Ahead

In typical ASR systems, it has been demonstrated that if the probability is properly redistributed, it would improve the pruning and performance (Ortmanns *et al.*, 1997a). This technique is named language model look-ahead. The idea is to incorporate the language model knowledge as soon as possible during the decoding so as to improve pruning efficiency. In "dynamic style" beam search (cf. Section 2.3.1), language model look-ahead is implemented as follows. Remember that in this type of search, a separate lexical tree copy is needed for each language model history *h*: upon reaching a leaf node for word *w*, the

language model probability $P(w|h)$ is retrieved. Language model look-ahead requires to propagate backwards the probabilities closer to the root node. This operation needs to be done for each history, yielding a problem when working with large language models. In an integrated static decoding graph, we thus use weight pushing techniques. It is performed off-line and directly on the whole graph.

Weight Pushing on the Graph

In a weighted finite-state graph, the distribution of transition weights along the paths is a critical issue. One may leave transition weights in the beginning of a phone sequence to benefit from an early knowledge of the language model probabilities. However, this scheme will result in a less efficient graph due to the more redundant arcs[15] that are not determinized. On the other hand, when the transition weights are pushed towards the end, it gives more possibility of producing a smaller graph. In contrast, the transition weights are almost useless if they are not pushed backward to the beginning of the phone sequence. The goal of weight pushing is to produce an equivalent graph with reasonable transition weight distribution such that decoding is performed at higher speed and accuracy.

Implementation

In our experiments, the word boundary is the important information for analyzing the transition weight distribution between words and the distribution among pronunciation variants. In addition, our fast alignment techniques (explained in Section 3.7.4) require the word boundary information to extract sub-graphs from a large WFST. Therefore, we proposed a weight pushing approach which is conceptually similar to the *tropical semiring* algorithm (cf. Section 3.6.1) but performs a simpler task. When the language model is composed with the dictionary, both the transition weights and output labels (words) are pushed forward for better determinization. At the end of graph construction, when the graph carrying physical models has been determinized, the transition weights are pushed backward as much as possible (refer to (3.17), the pFST operations). Word labels are left at the end of a pronunciation sequence,

[15] When a large transducer can not be directly determinized, the determinization is performed by graph encoding and decoding. Two arcs are considered identical when the input, output labels and transition weights are all the same. Therefore, the word labels are pushed forward for better determinization. (http://www.research.att.com/~fsmtools/fsm/man4/fsmintro.1.html)

which will help the sub-graph extraction procedure (presented in Section 3.7.4). Note that transition weights are redistributed within two word boundaries. The pseudo-code of our weight pushing algorithm is shown in Figure 3.15.

```
int graphNodeNum;// number of nodes to perform pushing
int option;// pushing forward or backward
function pushWeight(graph){
    nodeNum=collectNode(graph, graphNodeNum);
    while(nodeNum>0){
        for(int i=0;i<graphNodeNum;i++){
            if(graph[i].pushCondition is true)
                entonnoirPushing(graph, i, option);
        }
        nodeNum=collectNode(graph, graphNodeNum);
    }
}
```

Weight pushing algorithm: main program.

```
function entonnoirPushing(graph, node i, option){
    float sourceProb;
    sourceProb=getMaxInProb(graph, node i);
    toGate(graph, node i, option);
    reDistributeWeight(graph, node i, sourceProb);
    if(option==0)// pushing forward, also moving word labels
        forwardLabel(graph, node i);
    graph[node i].pushCondition=false;
}
```

Sub-routine of weight pushing.

```
function reDistributeWeight(graph, node i, sourceProb){
    int sNode, sBranch, update=0;
    for(int j=0;j<successorNumber of node i;j++){
        obtainSuccessorInfo(sNode, sBranch, j);
        graph[sNode].weight[sBranch]+= sourceProb;
        update=1;
    }
    // if redistributed, update predecessors' probabilities
    if(update==1){
        for(int j=0;j<graph[node i].inDegree;j++)
            updatePredecessorProb(node i, sourceProb, j);
    }
}
```

Sub-routine of weight pushing.

Figure 3.15: Weight pushing algorithm.

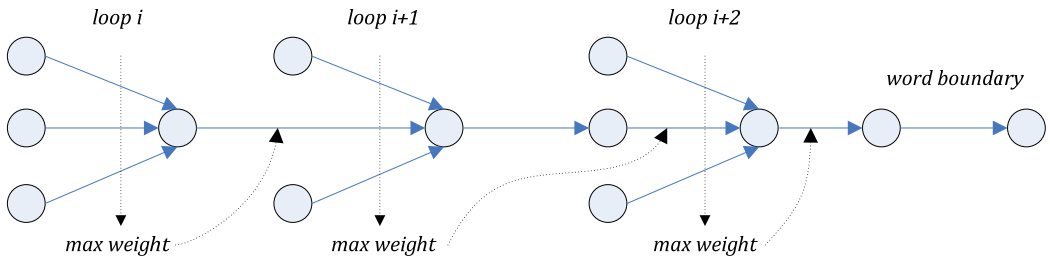

Figure 3.16: Weight pushing in the graph. Maximum transition weight is pushed towards the word boundary. The i is an iteration index of the while-loop in the pushWeight function.

At first, the main program computes the number of nodes whose transition weights are to be pushed. If one of the incoming transition weights is 0, the node will not be selected by the **collectNode(graph, graphNodeNum)**. The **entonnoirPushing(graph, i, option)** performs the weight propagation over the transitions. It calls three sub-routines to complete the task. The **getMaxInProb(graph, node)** returns the maximum transition weight among the incoming arcs of **node**. The function **toGate(graph, node, option)** traverses recursively the successors starting from **node**, until a word boundary or the successor whose in-degree greater than one is encountered. Finally, the **reDistributeWeight(graph, node i, sourceProb)** updates the weights.

Figure 3.16 illustrates how this algorithm performs weight pushing. Each pushing process gets the maximum weight from the incoming weights and pushes the weight as much as possible.

Complexity

Our weight pushing algorithm has $O(N)$ space complexity and $O(NMB)$ time complexity, where N, M and B respectively represent the number of graph nodes, the number of graph nodes that satisfy the pushing conditions and the average branching factor. In general, M is much smaller than N since most of the nodes carry an incoming transition weight 0. These nodes are not selected to perform weight pushing. B is also small in a deterministic graph. This algorithm gives competitive time complexity with respect to (Mohri and Riley, 2001) but is much more space efficient.

Weight Distribution Issues

Weight pushing is a critical step of the graph construction procedure, especially

for the phone level graph. A word often represents a couple of phone sequences when a language model graph is composed with a dictionary. How the language model probability is distributed along the sequence is an issue to investigate.

In the above algorithm, the maximum transition weight is pushed as much as possible until either the node whose in-degree is more than one or the word boundary is reached. For pushing weight forward (the graph $L' \circ G$), the node whose in-degree is greater than one is a word boundary. For pushing weight backward, the same algorithm is used but is performed on a reversed graph. This is an implementation technique. If the graph is reversed, it is simple to find the successors and push the maximum weight towards the word boundary.

Besides, in a reversed graph, the node whose in-degree is greater than one is the place where paths are distinguished. Pushing weights to this place could improve pruning. However, a critical issue when pushing weights backward is that a couple of pronunciations are unique or slightly determinized on the first leading arcs(s). When pushing weights backward, it is obvious that transition weights would have more possibility to be pushed towards the word boundary. During the decoding process, the token that reaches this word boundary may receive exact language model probability instead of a re-distributed transition weight, resulting in more penalty or reward than other competitive candidates.

3.7 Implementation: Design for Fast Decoding

Discriminative training requires the parameter updating process to be iteratively performed with a set of training samples. For large vocabulary ASR tasks, especially on a large database, a fast decoder is needed to make the training practical. Therefore, we develop several implementation techniques aiming at a fast decoding. In the following sections, we review the computation bottlenecks of discriminative training and present our own solutions.

3.7.1 Computation Bottleneck

Discriminative training using the MCE criterion involves 1) decoding a training sample, 2) alignment of the sample and then 3) performing parameter update. The three processes were illustrated on Figure 3.2. Updating parameters is a simple task. It adjusts transition weight on a certain arc. However, fast decoding and alignment can not go with ease, especially on a complex search space.

Decoding consists in finding the most likely word sequence in a large graph. Searching all possible paths to select the best candidate is not tractable. However, in a connected network, different transitions may reach the same state and branch to a lot of successors. If the decision can be made in advance before branching, it would significantly eliminate useless candidates and reduce the time for comparing hypothesis scores. This motivates our design of a look-up table, which helps determine the best score without actually traversing the graph (Saon *et al.*, 2005). This is explained in Section 3.7.2.

Alignment is the process of finding the best state sequence given a word string. If the paths that produce the word string can be pre-computed, alignment will not have to do a lot of searching. Furthermore, if the paths can be extracted, also in the form of a graph, alignment is performed in a quite small search space and the speed of alignment will be extremely fast. Thus, we propose a general sub-graph extraction approach. It can be used not only for doing alignment in discriminative training but also for other alignment tasks.

3.7.2 Look-up Table

General Token Passing

Our decoder performs the search by using the general token passing (Young *et al.*, 1989) over a finite-state network. A token can be viewed as an object holding the information such as the accumulated log probability (score) at a state i up to current time frame t and the path history of the token. At each time frame, the tokens are propagated through the network, producing a bunch of successors. To make the search efficient, the following steps are performed for all tokens.

- Update token's information: the state position, the score and the history.
- Determine the best token reaching a state.

With the above procedure, the tokens with the best score are used for the next time frame. In many ASR tasks, a pruning method is often used after the procedure, to further reduce the number of tokens so that only the top-N tokens are used at each time frame.

Our decoder implementation extends this concept over WFST. Tokens are associated with the weighted transitions. For the tokens that are at the last acoustic state of a model, graph searching amounts to propagating the tokens to

the next acoustic states. However, epsilon transitions are bothersome to determine the best token at the states since tokens will not stay over epsilon transitions. As displayed on Figure 3.17, the tokenB and tokenC directly reach state 6. The selection of the best token is easy. But tokenA still reaches state 6 through an epsilon transition $\overrightarrow{56}$ at the same time. To make the selection of best token more efficient, we use a lookup table.

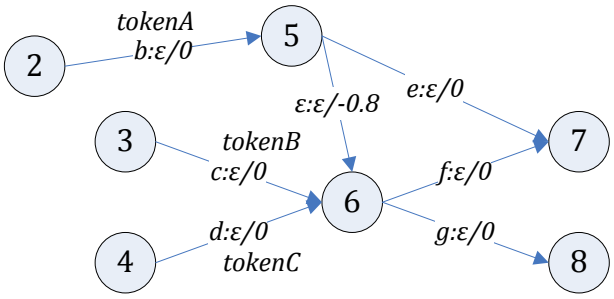

Figure 3.17: Decoding hypothesis redirection for merging. Tokens are over the transitions, rather than on the states.

Implementation

The lookup table holds the epsilon-closure relationships. An epsilon closure is a set of states reachable from a set of starting states through epsilon transition(s):

$$\varepsilon - closure(Q') = \{q \mid q' \in Q', q' \to q \text{ is } \varepsilon\text{-reachable}\}$$

The relationship of epsilon closure represents the branch index of epsilon transitions for each state and the sequence of branch indexes to reach the next non-epsilon transitions. It is built by traversing the graph, saving the epsilon paths information in a table. It is performed off-line together with the graph construction procedure. With the look-up table, a token can be easily re-directed to its destinations without having to search the graph. Figure 3.18 shows the procedure for token redirection using a lookup table.

```
for i=1 to number of tokens {
    for j=1 to number of destinations of token i {
        destinationState=redirectToken(lookup table, j, token i);
        selectTheBestToken(destinationState);
    }
}
```

Figure 3.18: Redirection procedure for tokens that are going to leave the arc.

In brief, the redirection procedure firstly determines the number of destination states of a token. For instance, tokenA in Figure 3.17 ends at state 5 with two destination states 6 and 7. The function **redirectToken(lookup table, j, token i)** performs the redirection of tokens over the transition to the destination state. Thus, a copy of tokenA is propagated over $\overrightarrow{57}$ (a non-epsilon transition) and another copy is moved over $\overrightarrow{56}$, penalized by -0.8. TokenB and TokenC end at state 6 which does not have any outgoing epsilon transition. The function **selectTheBestToken(destinationState)** is directly performed without token re-direction. It keeps the state accessed by the token with the highest score before branching to its successors.

Notice that the redirection is not performed by actual graph search. The redirection path is kept in the look-up table. The use of look-up table not only helps determine the best token in an efficient way but also skillfully prevents from extraneous token branching.

Our experiments are performed on a graph of more than 7M states and 20M arcs. The decoder keeps 20k tokens at each time frame. During decoding, the tokens which are going to cross a word boundary are stored in an array. Generally, the number of such tokens is less than 3000. The above mentioned token redirection procedure is performed only for these tokens. Our analysis observed that this procedure takes less than 5% of the total decoding time but significantly trims useless candidates and greatly accelerates the graph search.

Pruning Techniques

Token-passing approaches often use pruning techniques to retain the most promising tokens for following path expansions. Two popular pruning criteria are:

- Global beam pruning: the tokens t' are eliminated if $S(t') < k \cdot S_{best}(t)$, where $S(t)$ is the score of token t and k is a coefficient to determine the beam size.
- Histogram pruning: this pruning method keeps a certain number of active tokens at each time frame.

The first pruning criterion requires that the pruning factor k should be carefully estimated, so that the beam size would not be too tight to reach a high decoding accuracy. Our decoder thus uses the histogram pruning to limit the maximum number of tokens to 20k, and eliminates the rest ones.

3.7.3 Pseudo-Sorting

In many ASR tasks, the exact order of the top-N tokens is not necessary during decoding. Our decoder implements a pseudo-sorting approach to efficiently obtain the N-best tokens, given in the following algorithm:

```
void getTopN(tokenArray, int low, int up, int topN){
    int splitPosition, pkey;
    if(low<up){
        pkey=(low+up)/2;
        swap(tokenArray, low, pkey);
        splitPosition=partition(tokenArray, low, up);
        if(splitPosition+1>topN)
            getTopN(tokenArray, low, splitPosition, topN);
        if(splitPosition<topN-1)
            getTopN(tokenArray, splitPosition+1, up, topN);
    }
}
```

Figure 3.19: Pseudo-Sorting algorithm for getting the top-N tokens.

At first, the algorithm selects the middle element as a pivot and swaps the element with the leading element. Once the pivot has been placed in the beginning of the array, the **partition(tokenArray, low, up)** examines the rest of array elements by comparing the value with the pivot. The result is a partially sorted array which is "filtered" by the pivot. The returned position indicates that the elements in the left are smaller than the pivot and those greater than the pivot are in the right.

Basically, this approach is performed by moving top-N tokens towards the beginning of the beam instead of doing a complete sorting to make all the tokens in a descending or ascending order. Even for the top-N tokens that have been moved, they are not in a sorted order. Our pseudo-sorting approach performs data swapping to collect the required number of tokens. Therefore, although it was devised from the Quick Sort algorithm (Hoare, 1961), it runs faster than the typical one.

3.7.4 Sub-Graph Extraction

Small Graph for Fast Alignment

The alignment is different from decoding in that the search procedure should

traverse the graph given a certain word string. That is, the possible paths for aligning a training sample are limited. It is not necessary to search the whole graph and to prune the incorrect paths for each training sample. This suggests that if these paths can be computed in advance, the graph searching work would be significantly reduced. In other words, alignment can be performed in a very fast way on a quite small graph in which the relevant paths have been extracted from the large one.

Principle of Graph Inversion

Our approach is motivated from the general principle of graph inversion. This operation exchanges the input and output symbols of state transitions, so that for each transition:

$$a : b / weight \xrightarrow{inversion} b : a / weight$$

where the input symbols are either words or empty strings after graph inversion. By searching the desired word strings, the relevant paths can thus be selected in advance.

Implementation

To extract the sub-graph for fast alignment, a complete graph search is necessary to find all possible paths from the starting node to the terminal node. All matched paths are saved in a file corresponding to each training sentence. This process is performed off-line. Since the graph contains back-off transitions, the paths from one word to another may be multiple. In addition, a word may appear more than once in the sentence while the path reaching that word may be different. For instance, "*de publier **des** contenus protégés par **des** droits d'auteur*" contains two occurrences of ***des*** with different left and right contexts. Therefore, to produce a correct word sequence, a string comparison is needed during alignment. Assume that the training set contains the following word sequence:

<s> le vin français retrouve un peu de couleur à l' étranger </s>

Extraction is performed on the large graph, starting from state 0, searching all possible paths until a word boundary of *<s>* is reached. A string matching program is executed to compare the word labels for each traversed path. Then, from the states that produce *<s>*, the extraction continues traversing the graph for the next word *le*. This procedure is repeated from *<s>* to *le*, from *le* to *vin*,

from *vin* to *français*, until *</s>* is found. An extraction example is shown from Figure 3.20 to Figure 3.22. Figure 3.20 illustrates the possible paths from *vin* to the next possible words in the large graph. If a path produces the desired word label *français*, all arcs along that path are selected. Figure 3.21 shows examples of valid paths and incorrect ones, which are respectively represented as solid lines and as doted lines. The procedure continues for all words in the word sequence. Finally, the states and transitions that have been selected are saved in a file, as shown in Figure 3.22.

Discussion

An implementation concern that is important and needs to be paid attention to for sub-graph extraction is data consistency. One has to keep in mind that the state indices of the small graph have been re-numbered, and the relative transition arcs have been changed. When the score and transition paths are obtained from the small graph for doing alignment, one has to know their corresponding positions on the large graph in order to update the transition weights at the right place.

Moreover, several graph parameters may have been updated by training sample *i*. When doing alignment for sample *i+1*, where some transition weights in the large graph may have been adjusted by previous training samples, transition weights on the sub-graph have to be synchronized with the large graph before the alignment starts, such that the alignment is always using the weights which are updated by the training procedure.

All the mapping information, from sub-graph to large graph, for transition weight synchronization and index re-numbering are obtained during the graph extraction procedure. When the small graph is extracted, the extraction procedure also outputs a table, which lists the mapping position from small graph to the large one, including states, arcs and branch indices.

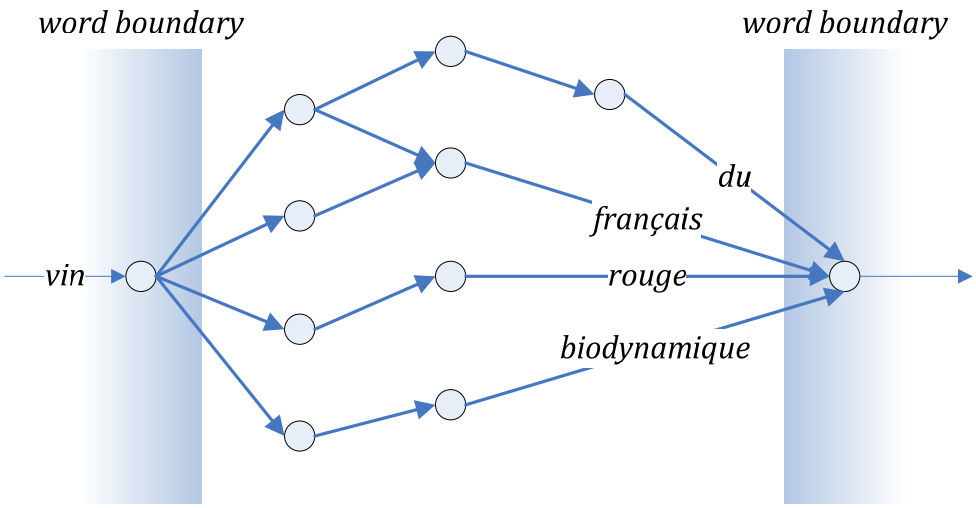

Figure 3.20: Possible paths branching from "vin".

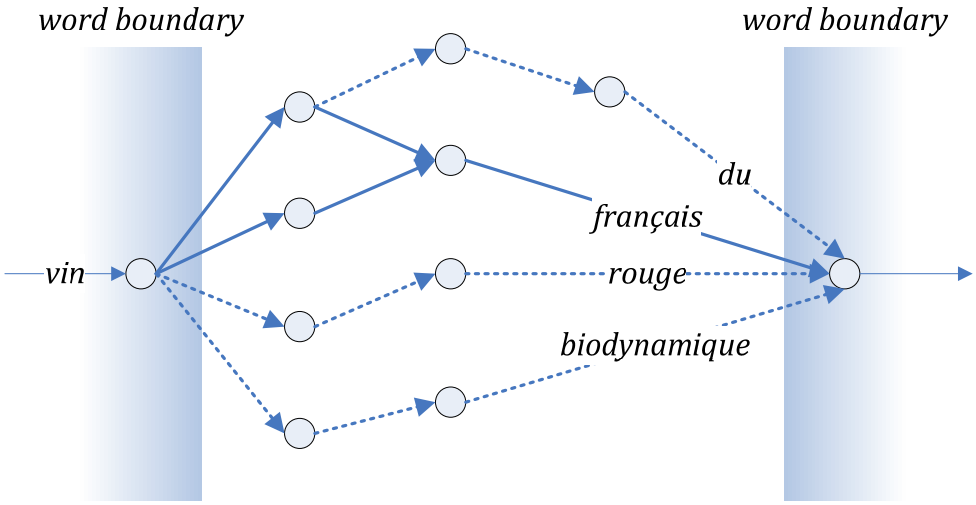

Figure 3.21: The paths that match the alignment string "français".

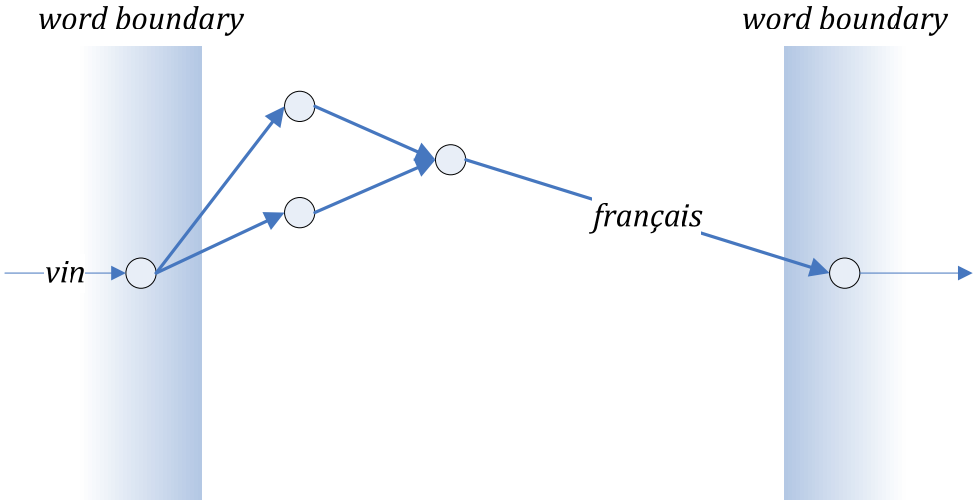

Figure 3.22: Extracting the desired paths.

Performance

The extracted graph generally consists of hundreds of states and arcs. For a few long sentences, the number of states and arcs is in the order of a few thousands. Carrying out alignment on such a small space is extremely fast, no matter how large the original graph is. In addition, the look-up table technique proposed in the previous section is also used in alignment for determining the best token at each state. Experiments demonstrate that our sub-graph extraction approach greatly reduces the time for alignment, yielding about 0.05 real-time on a desktop machine.

3.8 Summary

In this chapter, we presented several discriminative training criteria, notably the MCE criterion. It is an error rate based approach that combines the decoding errors into a loss function to reflect the training improvements. Recent developments in WFST allow various knowledge sources to be combined in a single search space, and to be optimized so as to yield an equivalent network with better time and space efficiency. Our training approach applies the WFST techniques to construct the decoding graph, and uses MCE to reduce the decoding errors by updating the transition weights. Moreover, we also implement several fast decoding techniques to speed up the training procedure.

In the next chapter, we will present and discuss our experimental results.

Chapter 4. Experiments

In the previous chapters, statistical modeling problems were investigated as well as their fundamental assumptions, and various estimation techniques were discussed. Conventional modeling approaches use maximum likelihood estimation (MLE) methods to estimate the model parameters. However, discriminative training of acoustic models, which aims at optimizing the separation between the reference and the hypothesis, has been shown to outperform MLE in many ASR tasks. In Section 3.2.2, discriminative training using minimum classification error (MCE) criterion was extended from language modeling to update parameters in an integrated decoding graph. In this approach, transition weight adjustment is iteratively performed to minimize the error rate. Our previous work (Lin and Yvon, 2005) conducted on Swiss-French Polyphone database (Chollet *et al.*, 1995) has demonstrated a significant improvement in error rate reduction. In this chapter, we apply this approach to a much more complex large vocabulary task. Parameter update rules and error rate reduction are analyzed from various angles.

4.1 Study on Small Vocabulary System

In our previous work, we carried out experiments on a small vocabulary task (Lin and Yvon, 2005) (presented in Appendix D, p. 126). It extends the concept of MCE training on language models and formulates the discriminative training procedure on an integrated decoding graph. This training framework is similar to what we used for ESTER experiments (presented later), except that:

- Discriminative training is performed on a small graph.
- Learning rate ε is fixed and is selected empirically.

This task consists in recognizing isolated sequence consisting of a proper name followed by its spelling, as illustrated by:

bûche b u accent circonflexe c h e

In this task, the proper name and its spelling are represented by a phone sequence (mono-phone labels) and letter sequence respectively:

<s>-b-y-S-</s> <s>-b-u-accent-circonflexe-c-h-e-</s>

The search space is accordingly composed of two parts: a) a graph representing the pronunciations and b) a graph containing the name spelling. These two graphs are constructed by WFST techniques from language models estimated on the phones and letters of proper names respectively. In the Swiss-French

Polyphone database (Chollet *et al.*, 1995), there are 39 and 53 possible phones and letters (including <s> and </s>). Therefore, the graph constructed from a 2-gram language model contains 516 states and 2198 arcs. Even if a 3-gram language model is used, the search space contains only 8154 states and 32467 arcs.

Experimental results showed that decoding errors are significantly reduced both on the training set and the test set, yielding a 6.5% absolute improvement of error rate on a decoding graph constructed from a 2-gram language model. A close examination on the effect of discriminative training also shows that confusion pairs are reduced for the top 30 most frequent ones, except for a few pairs that are acoustically and distributionally very similar.

4.2 The ESTER Database

ESTER[16] is a French radio broadcast news database (Galliano *et al.*, 2005; Gravier *et al.*, 2004). The corpus was collected from four different sources: *France Inter, France Info*, *Radio France International* and *Radio Télévision Marocaine*. The ESTER database contains a large amount of recordings, including manually transcribed and non-transcribed portions. Approximately 90 hours of manually transcribed data was used for training the parameters of acoustic and language models, including 8 hours of data that was used as a development set. The test set consists of 10 hours of speech taken from five sources: the four above mentioned radio stations, and one hour from *Radio Classique*, for which no specific training data is available. Both during training and testing, only the manually transcribed portion of the ESTER database is used. Text corpus for training language models was taken from the newspaper *LeMonde*, covering approximately 400M words.

Data Sets

Our discriminative training procedure involves two separate training processes. The first training process estimates the parameters of the various knowledge sources such as acoustic models and language model. This process is performed using conventional MLE. The second is discriminative training. It updates the graph constructed from these knowledge sources. When the graph parameters have been updated by discriminative training, another data set is used to

[16] Campagne d'évaluation des Systèmes de Transcription Enrichie d'Émissions Radiophoniques http://www.afcp-parole.org/ester/index.html

estimate the performance improvements. Therefore, three data sets are needed to run the experiments. These datasets are chosen from the manually transcribed portion of the ESTER database:

- Development set (DEVSet): the data used for discriminative training.
- Test set (TESTSet): the data used to evaluate the performance of the graph updated by discriminative training process.
- Training set (TRAINSet): the data used for estimating the acoustic models and the language model.

The TESTSet contains 4 categories of recordings, listed in Appendix C.3 (p. 125).

Selection of Recordings

The decoding graph is constructed from a language model, composed with the dictionary, context-independent to context-dependent transducer and with graph optimization techniques which yield an integrated decoding network. We have used a 65k words vocabulary, a reasonable size for a large vocabulary ASR system. This means that the constructed decoding graph is not able to decode any sentence containing a word that does not belong to the vocabulary. Additionally, some characteristics of human speech, such as hesitations, are not integrated in the graph. To deal with these sounds, an extra network must be constructed and concatenated to the original graph, which makes it complicated to determine the updating position. In addition, distributing transition weights over hesitations does not make sense. Therefore, recordings have been removed from the development set if one of the following conditions is met:

- The transcription contains out-of-vocabulary (OOV) word(s).
- The transcription contains more than one terminal symbol </s> or no word label between <s> and </s>, such as <s> </s> bonjour à tous </s>.
- The transcription contains labels which denote hesitations.

The number of recordings and hours of speech for each set are listed below:

Data Set	Num. of Recordings	Hours
DEVSet	6489	5.30
TESTSet	3307	2.95
TRAINSet	70423	62.88

Table 4.1: Data sets for running the experiments.

4.3 Experimental Setup

In this section, the use of knowledge sources, including acoustic models and language model, is described in detail. The parameter selection process is also presented in the following.

4.3.1 Acoustic Models

Acoustic models are built from the TRAINSet. Parameters are estimated by Hidden Markov Model Toolkit (HTK)[17] (Young *et al.*, 2002). A decision tree is applied to synthesize the missing models, based on the available ones.

Synthesis of Unseen Models

Our original HMM set contains 21466 acoustic tri-phone models. Additionally, 1369 models are synthesized according to the decision tree by state-tying. Each model consists of 3 states, except the short pause model (one-state model) used only for the short silence between words. There are a total of 6238 distinct states. Each state is associated with a 39-dimensional probability density function taking the form of a mixture of 32 Gaussians, assuming a diagonal covariance.

Feature Extraction

Feature extraction is performed by the SPro[18] toolkit which is a speech signal processing toolkit implementing standard feature extraction algorithms. The length of individual time frame is 25 ms and is taken with a frame periodicity of 10ms. Each signal frame is a feature vector containing 39 elements. The first 13 elements are the 12 first Mel-frequency cepstral coefficients (MFCC) plus the *log*-energy value. The next 13 features are *delta coefficients* which are estimated by taking the first order derivatives of the first 13 elements. The last 13 are *acceleration coefficients* obtained from the second order derivatives of the first 13 elements.

4.3.2 Language Model and Graph

When an *n*-gram language model is constructed in the form of a graph, the graph

[17] http://htk.eng.cam.ac.uk/
[18] http://www.irisa.fr/metiss/guig/spro/

performance heavily depends on the quality of the language model. Generally, a large language model is supposed to cover more *n*-grams and yields a lower perplexity, which often indicates better performance. Various language models of different sizes are studied in our experiments.

Choice of Language Models

In our experiments, two 3-gram language models of distinct size and perplexity are chosen for comparison:

- A large language model denoted by *LM0*.
- A small language model denoted by *LM1*.

The first language model *LM0* is a large 3-gram language model obtained by linear interpolation of several language models. The associated weights are estimated such that the resulting model is improved. The second language model *LM1* is a "pure" ESTER 3-gram language model, whose probabilities are estimated from the transcriptions in the TRAINSet. No pruning is performed when estimating the model parameters.

In general, a large language model takes a lot of memory space. Many researchers have investigated the issue of reducing the size of language model while keeping the performance loss as less as possible (Chen *et al.*, 1998; Gao and Zhang, 2002; Goodman and Gao, 2000). Therefore, two more considerations have been taken into account in designing our language models.

(1) Language Model Pruning

Graph composition often requires a large amount of memory space, especially when the graph is composed with the context-independent to context-dependent transducer. Memory space allocation often fails when a large graph is to be constructed[19]. Due to the memory allocation problem, a large language model containing too many *n*-grams can not be directly used to build the graph without pruning.

In our experiments, the size of the large language model *LM0* is reduced with two pruning options. The first option is to prune *n*-gram probabilities if their removal results in model perplexity increase by less than a given threshold. The second

[19] In a 32-bit computer, the memory allocation limit is 3GB per process.

pruning option is to prune the *n*-grams whose *n*-gram probability is lower than corresponding back-off probability. The SRILM[20] (Stolcke, 2002) toolkit is used for language model pruning.

(2) Large Language Model of Lower Order

We felt it interesting to contrast the effect of discriminative training on a small 3-gram language model and a large language model of lower order. In our experiment, the 2-gram language model is produced by extracting the 1-grams and 2-grams from *LM0*.

The above considerations yield two more language models derived from *LM0*. For convenience, language models that are used to construct the graphs are listed in Table 4.2. *Graph1*, *Graph2* and *Graph3* represent the search space constructed from *LM1*, *LM2* and *LM3* respectively.

LM Notation	Graph Notation	Description
LM0	-	*a large 3-gram LM estimated on LeMonde*
LM1	*Graph1*	*a 3-gram LM estimated on TRAINSet*
LM2	*Graph2*	*a 3-gram LM by pruning LM0*
LM3	*Graph3*	*a 2-gram LM by pruning LM0*

Table 4.2: Language models and graphs used in our experiments.

Number of n-grams and Perplexity

The number of *n*-grams in each individual model is displayed in Table 4.3. The pruned 3-gram model *LM2* covers about 30% of 2-grams and 25% of 3-grams in *LM0*. The lower order language model *LM3* does not have any 3-gram entry. However, it contains all 2-grams of *LM0* and yields a lower perplexity than *LM1*. In Section 4.3.3, the baselines for each decoding graph will be presented to show the relationship between performance and perplexity on this task.

Graph States and Transitions

Following the notations presented in Table 3.1 and the graph construction principles introduced in (3.17), the evolution of the graph size during the construction procedure is shown in Table 4.4, Table 4.5 and Table 4.6. For every composition process, determinization is performed to eliminate the redundant arcs. When context-dependent phones are mapped to physical models,

[20] SRILM - The SRI Language Modeling Toolkit, http://www.speech.sri.com/projects/srilm/

determinization further reduces the graph size, as shown from G_3 to G_4. Finally, the operation sp is performed to incorporate optional silence at the end of each word.

Language Model	Order	1-gram	2-gram	3-gram	Perplexity
LM0	3	65001	17322565	15990056	82.59
LM1	3	65001	248739	102461	158.92
LM2	3	65001	5052090	4033834	86.42
LM3	2	65001	17322565	-	131.76

Table 4.3: The n-gram order, number of n-grams and perplexity of individual language models. LM3 contains all 2-grams of LM0.

Graph	States	Arcs
$G = Graph1$	75649	471608
G_1=pFST(wip($\pi(L' \circ G)$))	1590072	2681850
$G_2 = \det(G_1)$	595996	1102449
$G_3 = \det(C' \circ G_2)$	815162	1842915
$G_4 = \text{pFST}(\det(H \circ G_3))$	543409	1092851
$\text{sp}(G_4)$	824845	1655723

Table 4.4: Evolution of graph size when constructing Graph1.

Graph	States	Arcs
$G = Graph2$	972707	9806815
G_1=pFST(wip($\pi(L' \circ G)$))	9417059	35047728
$G_2 = \det(G_1)$	5680880	15428746
$G_3 = \det(C' \circ G_2)$	8122266	30164923
$G_4 = \text{pFST}(\det(H \circ G_3))$	4748568	15179397
$\text{sp}(G_4)$	6997044	19676349

Table 4.5: Evolution of graph size when constructing Graph2.

Graph	States	Arcs
$G = Graph3$	64776	17452164
G_1=pFST(wip($\pi(L' \circ G)$))	869569	33933803
$G_2 = \det(G_1)$	1016598	18732890
$G_3 = \det(C' \circ G_2)$	1644570	31249080
$G_4 = \text{pFST}(\det(H \circ G_3))$	574889	19831331
$\text{sp}(G_4)$	1842185	22365923

Table 4.6: Evolution of graph size when constructing Graph3.

4.3.3 Parameter Settings

Parameters for Discriminative Training

The parameters presented in this section are determined empirically. Table 4.7 shows the parameter settings for our experiments. The value α is the so-called "fudge" factor which scales the acoustic score so that acoustic score and language model score can be balanced in the *log* domain. The parameter γ controls the slope of the sigmoid function. It has an influence on the way a score difference is turned into a weighted update, as illustrated in Figure 3.4. The word insertion penalty (WIP) is selected empirically based on preliminary studies. There are two options for setting the learning rate ε: 1) either to adjust it dynamically for each training sample using the line search method, or 2) a fixed learning rate is used. Using a fixed learning rate is detrimental to the convergence rate and the stability of improvements. Our empirical results suggest that $\varepsilon = 1$ is a suitable choice. The last parameter is the upper bound of the score difference between the reference and the hypothesis. A transition weight update is performed only if the score difference is positive and smaller than this threshold.

Decoding Graph	α	γ	WIP	ε,Dynamic/Fixed	Upper Bound
Graph1	0.1	0.02	0.4	Line Search/1	200
Graph2	0.1	0.02	0.5	Line Search/1	200
Graph3	0.1	0.02	0.6	Line Search/1	200

Table 4.7: Parameter settings for discriminative training and testing.

Baseline

Using properly selected parameters α, γ and WIP, the baseline for each decoding graph is shown in Table 4.8. From left to right, the columns in the table respectively represent the correctness, substitution error, deletion error, insertion error, word error rate (WER) and sentence error. The computation of error rate is performed by the NIST scoring package SCLITE[21] (NIST, 2000). As Table 4.8 shows, the *Graph2* constructed from the *LM2* language model yields the lowest WER. It is noticeable that *Graph3*, which is built from a 2-gram language model, gives better performance than *Graph1*. Although *Graph1* is constructed from a 3-gram language model, it contains much fewer *n*-grams than other two graphs and yields the highest error rate.

[21] NIST Spoken Language Technology Evaluation and Utility
http://www.nist.gov/speech/tools/index.htm

Decoding Graph	Corr.	Substitution	Deletion	Insertion	WER	S.Err
Graph1	68.5	23.6	7.9	3.4	35.0	84.7
Graph2	73.8	18.3	7.9	2.3	28.5	78.7
Graph3	70.7	21.1	8.3	2.9	32.3	82.9

Table 4.8: Baseline results of three integrated decoding graphs on the DEVSet. Substitution errors are significantly reduced if the graph is constructed from a better language model.

Overall, the WER reflects the difference of language model perplexity as listed in Table 4.3. The graph constructed from the language model with lower perplexity yields better performance, as far as substitution errors are concerned. These baselines are the starting points for further error rate reduction. Based on the parameter settings of Table 4.7, the discriminative training experiments are performed on a machine equipped with Intel Xeon 3.6 GHz CPU and 6GB physical memory. At each time frame, the decoder keeps 20k candidates in the beam search decoding; for the alignment, reference paths and corresponding word strings are obtained by full search.

4.4 Experimental Results

In this section, experimental results will be presented for the various settings discussed in the previous section. In Section 4.4.1, we present our proposed fast decoding techniques in discriminative training. Decoding efficiency will be examined on different graphs. Section 4.4.2 illustrates the error rate reduction which is observed using a fixed and a dynamic learning rate. In Section 4.4.3, we contrast the performance of the deterministic update and the random update. Section 4.4.4 reports the training performance on a large graph with 16 training iterations; the potential improvements of performing more training iterations. Generally, a large and complex graph contains a lot of parameters to update, while the number of graph updates performed through discriminative training is often relatively much smaller. In Section 4.4.5, a comparison is made by introducing an additional and large data set to train the same graph. Error rate reduction is observed both on the development and the test data.

4.4.1 Decoding Efficiency

In Section 3.7.1 (p. 75), we have discussed the computation bottlenecks of decoding with WFST. This motivated our use of a look-up table to store the epsilon-closure relationship and our sub-graph extraction algorithm, aiming at

increasing the efficiency of decoding and alignment. In the following section, the decoding speed on graphs of various sizes will be presented, together with the memory allocation of each search space and the run time of individual functions in the decoder.

Pre-computed Likelihoods

In practice, discriminative training updates transition weights without adjusting acoustic model parameters. This implies that the acoustic score of a training sample will not be changed throughout iterations. Therefore, to speed up the training procedure, acoustic likelihoods[22] are computed in advance and are saved in a file for each training sentence.

Decoding Speed and Memory Allocation

With pre-computed likelihoods and our fast decoder, the decoding performance is reported in Table 4.9. It shows that our proposed sub-graph extraction technique yields a very time-efficient alignment. Since the sub-graph is small and is extracted from the large one, the cost of alignment is almost independent of the original graph size. Our experiments also show that, without pre-computed likelihoods, the performance would be slowed by less than 0.7 real-time (RT) for decoding and by $0.1\times\mathrm{RT}$ for alignment.

In addition to the high decoding speed, our decoder is also memory efficient. Table 4.9 shows the memory allocation for graphs of various sizes. Table 4.4 and Table 4.5 show that *Graph2* is approximately 9 times larger than *Graph1* in number of states, and contains about 12 times more arcs. However, the allocated memory size is only increased by a factor of 4. When doing alignment, an extra memory block is allocated for the sub-graph, the corresponding look-up table and a mapping table which maps the state and arc index to those in the large graph. Generally, the small block takes less than 1MB of memory.

Decoding Graph	Decoding	Alignment	Memory Space
Graph1	$0.7\times$RT	$0.05\times$RT	220MB
Graph2	$3.0\times$RT	$0.05\times$RT	960MB
Graph3	$2.8\times$RT	$0.05\times$RT	600MB

Table 4.9: Decoding speed in real-time factor and memory allocation of graphs.

[22] The acoustic score is a sum of the likelihood and of the acoustic state transition probability. Likelihoods are pre-computed and the transition probability is determined during decoding.

Discussion

Although an extremely fast decoder ($0.8\times$RT) has been reported in the literature (Saon *et al.*, 2005), this decoder only keeps track of the word history of hypotheses. Other information such as the state transition sequence can not be recovered. However, discriminative training on a phone-level decoding graph requires to store the complete paths from the starting state to the terminal state, such that the transition weight update can be performed on the corresponding path. Our decoder thus stores the complete paths of the top 20k candidates at each time frame. Moreover, in a large and complex graph, a state often has hundreds of successors, particularly at a word boundary. This high branching factor also slows down the decoder, which explained why we were unable to achieve sub-real-time speed [23].

4.4.2 Fixed and Dynamic Learning Rate

Discriminative training using a fixed learning rate gives an identical scale for parameter change while using a dynamic learning rate implies to determine the scale for each training sample. In this section, we compare discriminative training using dynamic and fixed learning rates in terms of error rate reduction on the development set and the test set. This comparison is carried out on *Graph1* using the same parameter settings except for the learning rate. Although *Graph1* is the smallest decoding graph in our experiments, it was built from a language model whose size is comparable to the language models used in other discriminative training experiments (McDermott *et al.*, 2000; McDermott and Katagiri, 2005). Training starts from a baseline WER of 35% as shown in Table 4.8. Eight iterations are performed with the random updating scheme (discussed in Section 4.4.3). Results are shown in Figure 4.1 and Figure 4.2.

Performance on the Development Set

Figure 4.1 shows limited improvements when using a fixed learning rate. The WER is reduced from 35.0% to 33.9%. Moreover, error rate reduction on the development set is quite small after the 4th iteration. When using a dynamic learning rate, discriminative training gradually reduces the WER from 35.0% to 31.6%, yielding a substantial improvement of 3.4% absolute. Performance improvement stabilizes after the 6th iteration. Roughly speaking, both dynamic

[23] We also implement a function in the decoder, which only updates the word-level history. This could improve 0.3×RT decoding speed.

learning rate and fixed learning rate yield better performance than the baseline result. Decoding errors on the development set are steadily reduced and the performance improvements also stabilize after a few iterations.

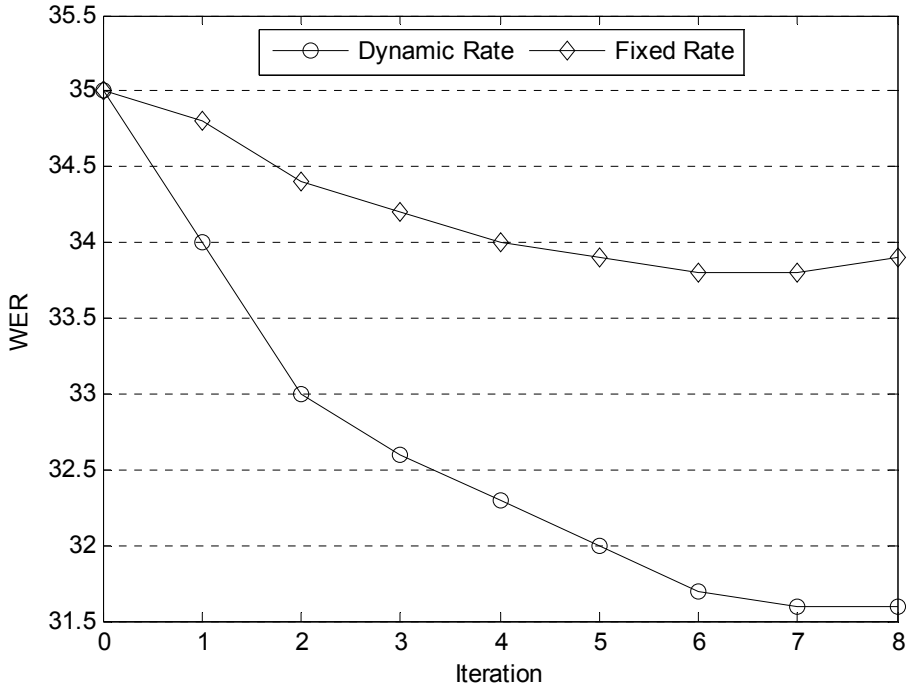

Figure 4.1: Error reduction for Graph1 on the DEVSet using a dynamic learning rate and a fixed learning rate.

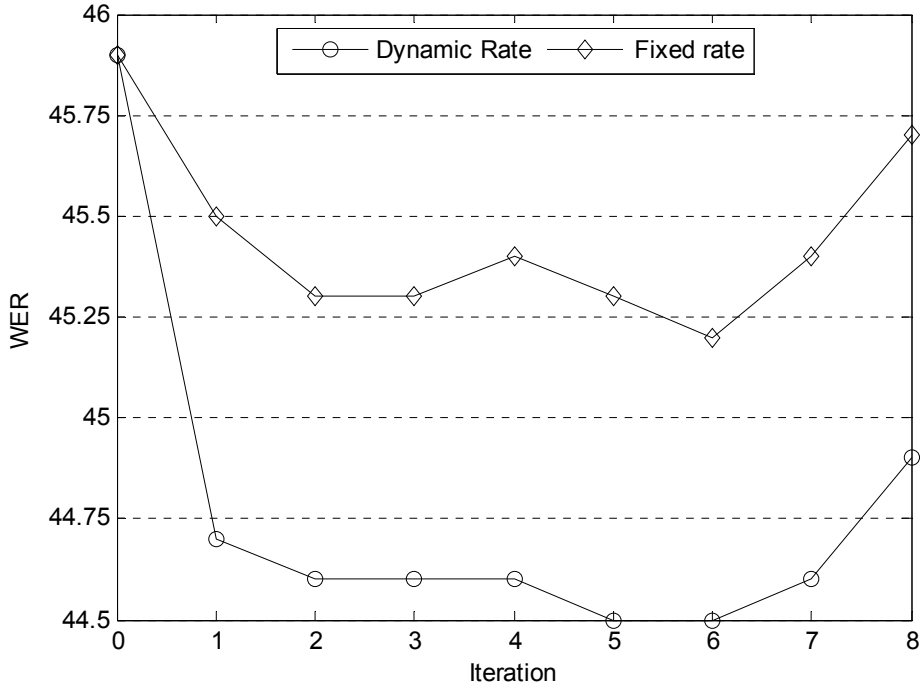

Figure 4.2: Error reduction for Graph1 on the TESTSet using a dynamic learning rate and a fixed learning rate. Over-fitting occurs after the 6th iteration.

Performance on the Test Set

Discriminative training yields substantial improvements on the development set. However, this error rate reduction is not generalized to the test set. Figure 4.2 shows the corresponding WER decrease for dynamic and fixed learning rates at each iteration. The 0^{th} iteration indicates the initial WER before discriminative training is performed. For the graph updated by fixed learning rate, the WER is decreased by 0.7% at the 6^{th} iteration, from 45.9% to 45.2%, of which 0.6% are reached after the 3^{rd} iteration. After the 6^{th} iteration, the error rate increases more quickly than the convergence rate, meaning that over-fitting occurs. This effect also happens for the graph updated by dynamic learning rate. Overall, dynamic learning rate yields more stable improvements on the test set before over-fitting occurs. It lowers the WER from 45.9% to 44.5%, which is 1.4% absolute to the initial point.

As illustrated in Figure 4.1 and Figure 4.2, dynamic learning rate yields more stable WER convergence and more WER reduction, both on development set and test set. As for the generalization problem and data over-fitting effect, they will be discussed later in Section 4.5.5.

4.4.3 Deterministic versus Random Update

The MCE training performs transition weight adjustment between two consecutive words that often involve a sequence of phones. However, the MCE criterion does not specify the exact position to update the transition weight. This gives several options to determine the arc for parameter adjustment.

- Update all the weights along the transition paths by the average of gradient or by giving more adjustment over the arc towards the initial state.
- Select one arc to update. This can be a deterministic or a random selection.
- For the paths containing a back-off transition, an option would be to keep the transition weight unchanged when the update yields a higher back-off probability than the *n*-gram probability.

Our implementation considers the back-off transitions as regular ones. For simplicity, we select an arc from the transitions of a word pair for parameter adjustment. In the following, we will compare the performance in error reduction between the deterministic update and the random updating schemes.

Transition Weight Updating Schemes

Assume a word pair of two consecutive words $W_1W_2 = \{a_1, a_2, ..., a_n\}$, where $a_1, a_2, ..., a_n$ are the corresponding arcs. Word labels W_1 and W_2 are produced at a_1 and a_n respectively. Two deterministic updating schemes are used for comparison, the update on the first arc a_1 and the update on the last arc a_n. In contrast to the deterministic update, the random updating scheme randomly chooses an arc along the paths with uniform probability.

Experiments are performed using *Graph1* on the development data set during 8 iterations. The scale of parameter update is determined dynamically by the line search method. All experiments are performed based on the parameter settings given in Table 4.7, starting from the baseline as shown in Table 4.8. The curves of WER reduction with three different updating schemes are displayed below. It is noticeable that the random updating scheme not only converges more quickly but also reaches a lower WER.

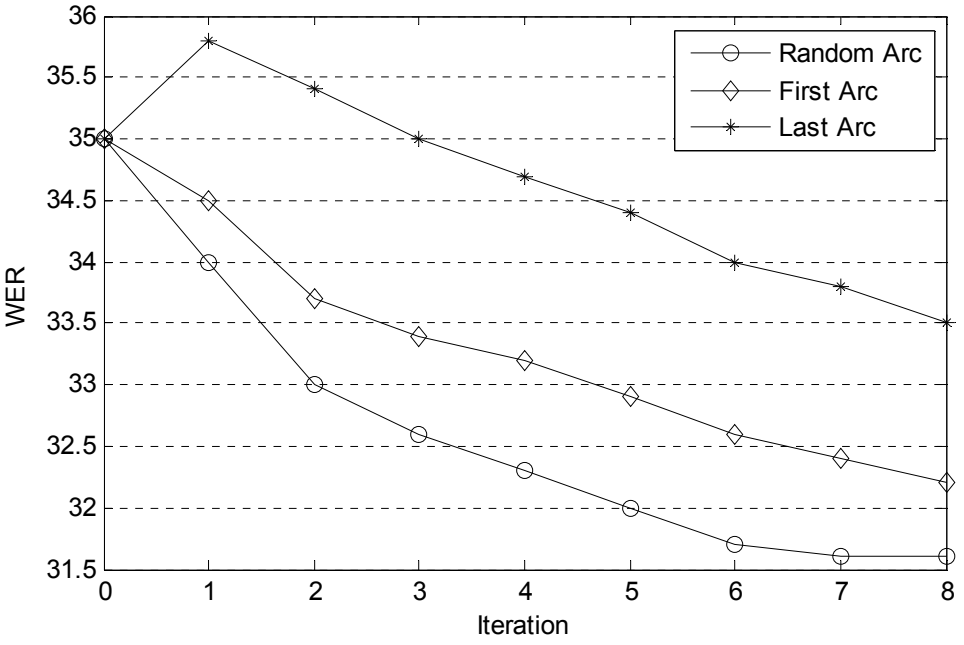

Figure 4.3: WER reduction on Graph1 by deterministic and random updates.

4.4.4 Error Rate Reduction on a Large Graph

An integrated decoding graph is a search space where a large number of parameters are distributed in a complex network. Some of the parameters, such as the transition weights, are not completely independent. This makes the training process difficult to reach the minimum error rate, particularly when the graph is constructed from a very large language model. However, a large graph

usually provides a better baseline for discriminative training. Therefore, given the same training data, we compare two graphs of different sizes to investigate the training behavior and potential improvements.

Training is performed using development set on *Graph1* and *Graph2*. Baselines are given in Table 4.8. Experiments are carried out based on the parameter settings given in Table 4.7, using a dynamic learning rate and the random updating scheme. In the following experiments, 16 iterations of training are performed. The performance of the first 8 iterations on *Graph2* is compared with the performance of *Graph1* represented in Figure 4.1 and Figure 4.2. From the 8th to the 16th iteration, the supplementary training iterations on *Graph2* are performed to see the possible error rate reduction in a large graph. Details of the error reduction on *Graph2* are presented in Appendix C.2 (p. 123).

Performance on a Large Graph

Figure 4.4 and Figure 4.5 show the improvements from the 0th iteration (the baseline). In Figure 4.4, a 3.4% absolute error rate reduction can be obtained from the baseline to the 8th iteration. This improvement on *Graph2* is similar to the performance on *Graph1* (the WER was reduced by 3.4% absolute) but starts from a better baseline. Experiments on *Graph1* and *Graph2* both give stable convergence on the development data. However, the improvement on the test set still suffers from the same generalization problem; only 1% of WER is obtained on *Graph2*, and approximately half of the improvement is achieved in the 1st iteration.

Training Iterations and Error Rate Reduction

Since the decoding graph involves a large number of parameters, several iterations of the training process may not be sufficient to give a proper transition weight distribution. Thus, eight more iterations are performed on the *Graph2* to investigate the training behavior. As Figure 4.4 indicates, training error is gradually reduced when more iterations are performed. Although the slope of convergence is getting smaller, around 1.5% absolute WER reduction is obtained from the 8th to the 16th iteration. That is, a total of 5% of training errors can be reduced from the baseline to the 16th iteration. The WER reduction on the development set is mainly due to the decrease of substitution errors and to a small decrease (1% absolute) of the deletion errors. Despite the significant improvements, error rate reduction on the development set almost does not yield any performance change on the test set from the 8th to the 16th iteration.

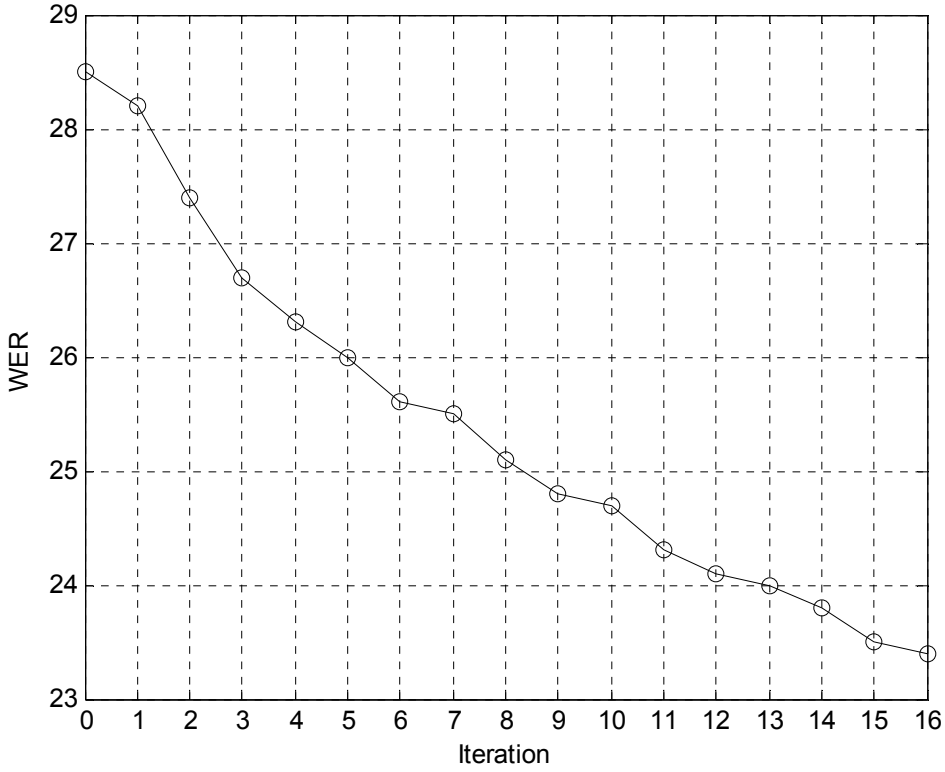

Figure 4.4: WER of Graph2 on the DEVSet.

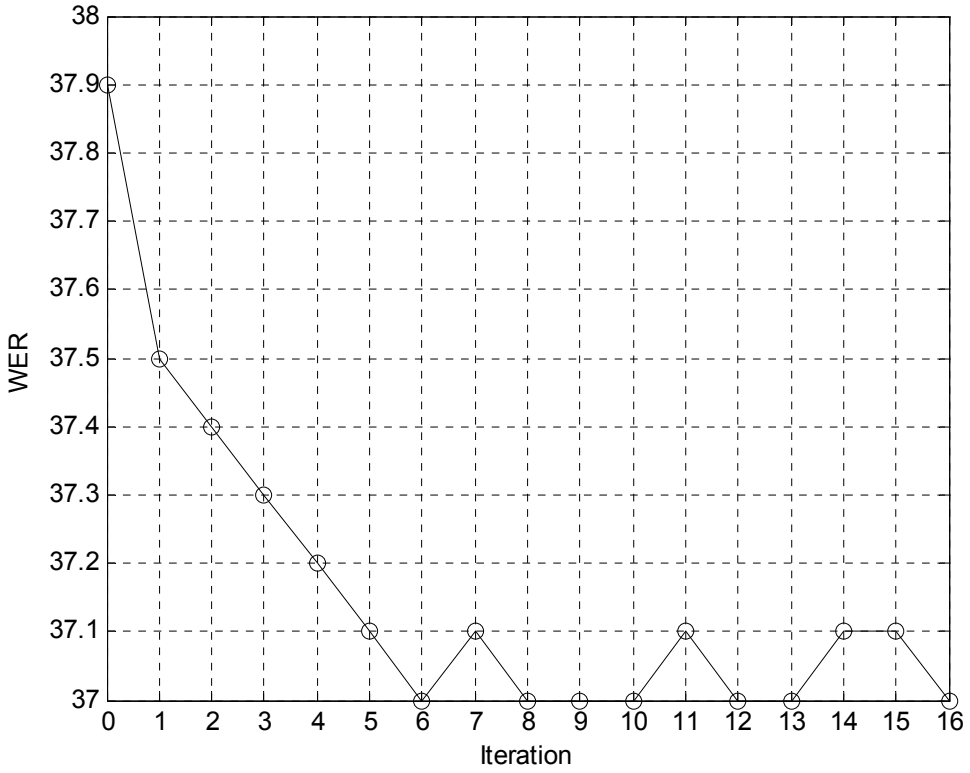

Figure 4.5: WER of Graph2 on the TESTSet.

Discussion

Experiments reported in the previous sections have shown the error reduction incurred through the discriminative training process. However, in practice, a graph may be constructed from a language model containing a large number of *n*-grams while the number of word pairs in the development set to update the graph is relatively small. This means that discriminative training updates only a small portion of the transition weights even if more training iterations are performed. In the next section, we will dig into this issue by introducing a larger data set to evaluate the potential of using a larger training set.

4.4.5 Training with a Larger Data Set

Occurrence of Word Pairs and Parameter Update

Discriminative training performs parameter update by increasing some transition weights while decreasing some other ones. The goal of the transition weight adjustment is to yield a proper distribution so that the error rate can be reduced. However, in practice, many word pairs will never be updated or the parameter update is performed given a specific history. Assume that the word pairs W_aW_b, W_aW_c and W_aW_d are distributed as below:

- W_aW_b: exists in the training set only.
- W_aW_c: occurs both in the training set and the test set.
- W_aW_d: exists only in the test set.

It is obvious that a transition weight update on W_aW_b may not yield any contribution to the performance on the test set but might help on some other test set. Updating the transition weight on W_aW_c is more likely to give an error reduction, since the word pair exists both in the training set and the test set. One little drawback is the non-locality effects. Both W_aW_b and W_aW_c start from the same word W_a. Transition weights on the path W_aW_c may be affected by the parameter change on W_aW_b. Technically, if the word pair exists only in the test set, there might even not be any transition to update, meaning that there is no arc for this word pair and that a weight update may occur only on the back-off transition. Due to the absence of this word pair in the training, a proper transition weight distribution on W_aW_d is difficult to achieve. Even if a couple of training iterations are performed, transition weight distribution of W_aW_d may not be changed, or will be modified in an arbitrary way, as a side effect of other changes, meaning that this error on the test set may never be fixed.

In this section, we use the data set originally used to train the language model for discriminative training. Experiments are performed on *Graph2*, based on the parameter settings given in Table 4.7. Starting from the baseline as shown in Table 4.8, one training iteration is performed on TRAINSet and DEVSet to compare the performance improvement. The number of recordings and hours of speech of the individual data set is given in Table 4.1.

Number of Word Pairs in the Data Set

Table 4.10 shows the number of word pairs in each data set. A word pair is obtained by extracting two consecutive words from the transcriptions. Thus, a word pair may occur several times. The number of distinct word pairs is shown in the 3rd column of Table 4.10. It can be seen that a word pair is updated 2 or 3 times on an average. However, the number of different word pairs is much less than the total number of parameters[24] in *Graph2*, as shown in Table 4.5.

Performance on the Training Set

Table 4.11 shows a significant improvement in performance when using a large data set in discriminative training. The large database TRAINSet contains 11 times more word pairs and 7 times more different word pairs than those in DEVSet. Discriminative training performed on this data set greatly decreases all kinds of training errors, particularly a high reduction of substitution error. Training result shows that a 7.9% of WER improvement is achieved over the baseline result.

Performance on the Test Set

As reported in the previous experiments, the improvement on the test set is much smaller than that on the training set, as shown in Table 4.12. When much more training samples are used to update the graph parameters, both substitution errors and deletion errors are reduced while insertion errors are slightly increased. Decoding results on the test set give a total of 1.1% absolute WER reduction, approximately 3 times more than the improvement obtained using DEVSet. Although the test set consists of four sources of recordings with distinct performance, as shown in Appendix C.3, discriminative training using a large data set still yields an improvement on each source.

[24] Since the acoustic model parameters are fixed, the number of parameters corresponds to the number of transition weights in the decoding graph.

Data Set	Num. of Word Pairs	Num. of Distinct Pairs
DEVSet	69423	31628
TRAINSet	814624	221736
TESTSet	40729	22265

Table 4.10: Number of word pairs in the data set.

Data Set	Corr.	Substitution	Deletion	Insertion	WER	S.Err
Baseline	73.8	18.3	7.9	2.3	28.5	78.7
DEVSet	74.0	18.1	7.8	2.3	28.2	78.3
TRAINSet	80.8	12.1	7.1	1.4	20.6	67.0

Table 4.11: Error rate reduction after the 1st iteration on the training set.

Data Set	Corr.	Substitution	Deletion	Insertion	WER	S.Err
Baseline	64.6	23.6	11.9	2.4	37.9	87.4
DEVSet	73.8	23.1	12.0	2.4	37.5	86.6
TRAINSet	70.7	22.8	11.2	2.8	36.8	86.2

Table 4.12: Error rate reduction after the 1st iteration on the test set.

Based on our previous experiments, it is likely that running additional training iterations with the large graph could bring us another 1% absolute WER reduction on the test set.

4.5 Improvements and Discussion

In the previous sections, experiments have shown that discriminative training has the potential of effectively reducing the errors. To further analyze the details of the improvement of discriminative training, we take the largest decoding graph (the *Graph2*) as an example to investigate the training results from the following perspectives:

- Error corrections: we study the reduction of errors before and after the training.
- Number of updates: we compare the number of graph updates in the course of training.
- Score difference: we observe the score difference between the reference and the hypothesis and examine how discriminative training re-distributes the transition weights.

- Coverage of parameters and the transition paths: we analyze the coverage of word pairs in the development set and the test set, and compare the transition paths. It may point out possible sources of improvements.

4.5.1 Detailed Analysis of the Corrections

In the following, we study the reduction of errors on the training set and the test set. To keep track of the error reduction, the top confusion pairs are extracted from the results produced by the SCLITE scoring tool. Since the improvement on the test set is limited, we focus on the confusion pairs that have been significantly reduced to discuss the possibility of generalizing discriminative training performance to unseen data. In Section 4.5.4, we will extend the comparison to investigate the issue of generalization in more detail.

Reduction on the Training Set

Previous experiments have shown that discriminative training gives significant improvements on the training set. The error reduction on the training set also produces a consistent result. Figure 4.7 shows that, for most of the pairs that are frequently confused when using the baseline graph, the number of errors has been decreased after discriminative training. Table 4.14 shows the number of errors for each confusion pair, before and after training.

Reduction on the Test Set

Graph parameter update performed on *Graph2* by using TRAINSet yields better improvements both in the training set and the test set, from which we take the decoding results on the test set before and after discriminative training. The top 10 confusion pairs are extracted from the baseline results to compare the improvements. The number of confusion pairs and the reduction are given in Table 4.13 and Figure 4.6. Several confusion pairs exist both in Table 4.13 and in Table 4.14. They are also ranked in the top 5 list, such as (*des*, *les*), (*et*, *est*) and (*la*, *le*). These confusions are effectively reduced after discriminative training is performed.

The word pair (*et*, *les*) shows a typical improvement on both sets. The words "*et*" and "*les*" are acoustically similar (also frequent in French) while the usage is quite different. With discriminative training to re-distribute the transition weights, there should be fewer errors occurring on this word pair.

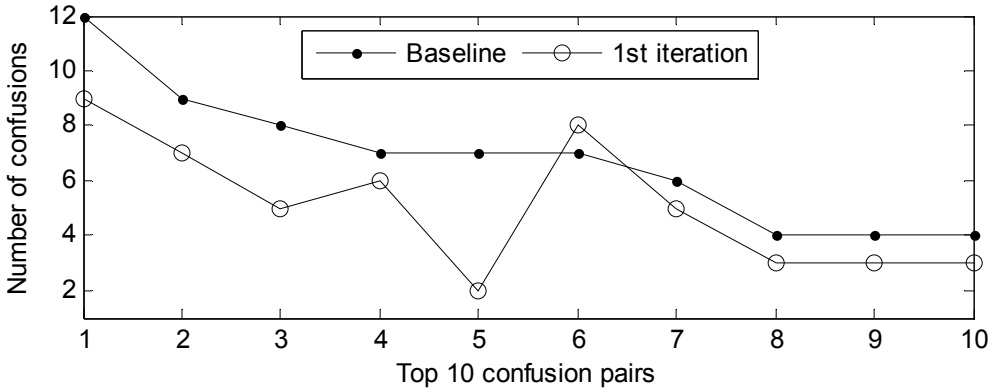

Figure 4.6: Reduction of confusion pairs on the TESTSet from the baseline to the 1st iteration. Discriminative training is performed on the TRAINSet.

Num.	Confusion Paris		Baseline	1st iteration
1	*des*	*les*	12	9
2	*et*	*est*	9	7
3	*la*	*le*	8	5
4	*des*	*de*	7	6
5	*et*	*les*	7	2
6	*à*	*de*	7	8
7	*l'*	*la*	6	5
8	*a*	*de*	4	3
9	*a*	*la*	4	3
10	*ce*	*le*	4	3

Table 4.13: Top 10 confusion pairs on the TESTSet. One iteration of discriminative training is performed on the TRAINSet.

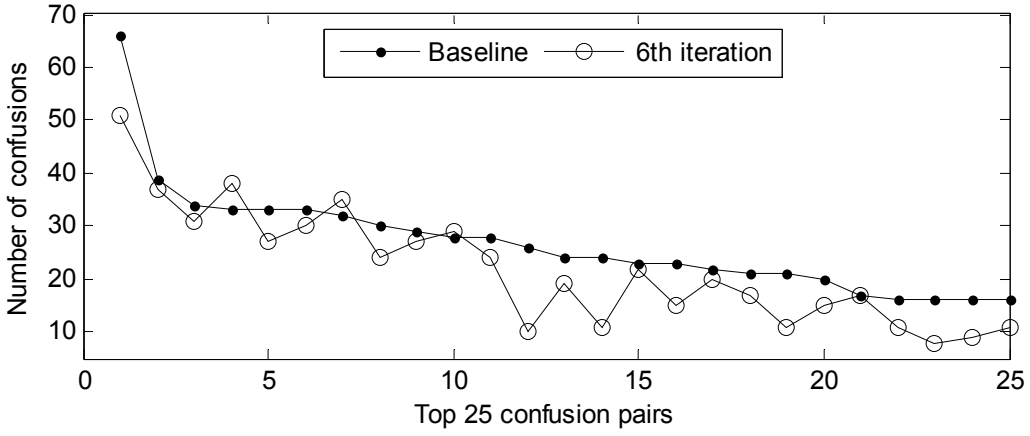

Figure 4.7: Reduction of confusion pairs on the DEVSet from the baseline to the 6th iteration.

Num.	Confusion Paris		Baseline	6th iteration
1	*des*	*les*	66	51
2	*est*	*et*	39	37
3	*les*	*des*	34	31
4	*et*	*est*	33	38
5	*la*	*le*	33	27
6	*un*	*le*	33	30
7	*à*	*de*	32	35
8	*l'*	*d'*	30	24
9	*les*	*le*	29	27
10	*des*	*de*	28	29
11	*l'*	*la*	28	24
12	*est*	*ces*	26	10
13	*au*	*le*	24	19
14	*le*	*les*	24	11
15	*il*	*qui*	23	22
16	*l'*	*le*	23	15
17	*l'*	*les*	22	20
18	*d'*	*de*	21	17
19	*et*	*les*	21	11
20	*deux*	*de*	20	15
21	*de*	*deux*	17	17
22	*de*	*le*	16	11
23	*de*	*que*	16	8
24	*et*	*il*	16	9
25	*ses*	*ces*	16	11

Table 4.14: Top 25 confusion pairs for the baseline system. The two columns on the right of the table show the reduction of confusion pairs from the baseline to the 6th iteration on the DEVSet.

4.5.2 Number of Graph Updates

Error Reduction and Graph Update

In the previous section, we have explained that discriminative training reduces the training errors by correcting the word pairs so as to make the reference and the hypothesis strings get more similar. This means that the number of updates

should also be decreased if the training procedure is effectively performed. To investigate the number of graph updates with respect to the training iterations, we take the sum of updates from the log file recording the details of discriminative training. The files are obtained from the experiments performed on *Graph1* and *Graph2* using the DEVSet. For the first 8 iterations, the corresponding number of updates is displayed on Figure 4.8.

Number of Updates and Graph Parameters

The *Graph2* starts from a good baseline (fewer updates) and the number of updates is gradually reduced both on *Graph1* and *Graph2*. Notice that the number of updates converges on *Graph1* while it continues to go down on *Graph2* due to the over-fitting effect.

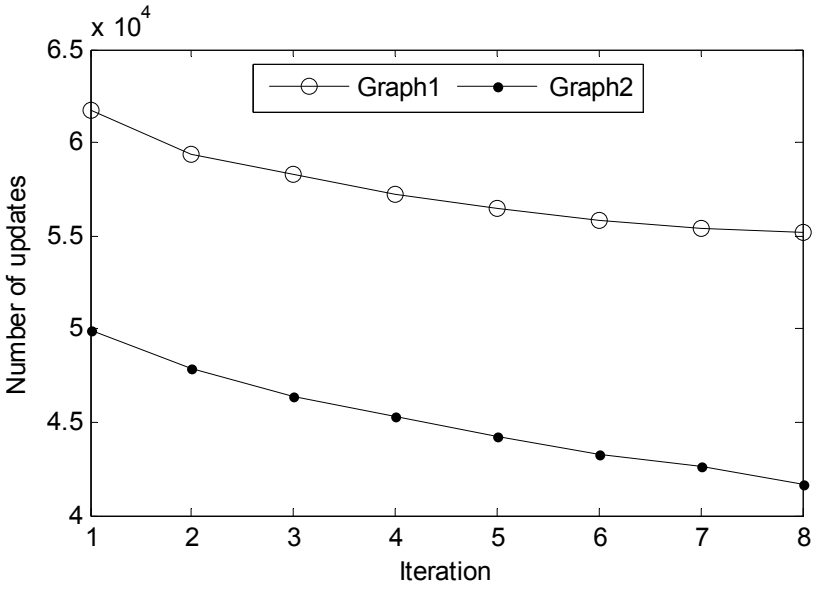

Figure 4.8: Number of parameter updates on Graph1 and Graph2 using the DEVSet.

4.5.3 Score Difference

When discriminative training using the MCE criterion is performed, the error reduction is achieved by adjusting the transition weights on the correct and incorrect words. In this section, we focus on the analysis of score differences between the reference and the hypothesis strings.

The score is a sum of the acoustic likelihoods and of the transition weights along the decoding path, as given in (3.1) (p. 50). When discriminative training is performed, the score difference between the reference and the hypothesis should be getting smaller. We investigate this issue by analyzing at the 1st and

the 6th training iteration on the log file mentioned in the previous section. Score difference in the intervals [0, 5), [5, 10),..., [45, 50) are grouped together. Their corresponding number of occurrences is shown in Figure 4.9.

From the 1st to the 6th training iteration, the number of files in each score interval is reduced, except for the range [0, 5), indicating that more hypotheses are closer to their references. Further investigation of the training samples also show that there are 1388 files whose reference and hypothesis are identical, both in scores and word strings at the 1st iteration. This means that the paths of state-level decoding and alignment are the same. Moreover, an increase of 217 files is obtained at the 6th iteration, meaning that discriminative training performs a proper transition weight adjustment to reduce the errors.

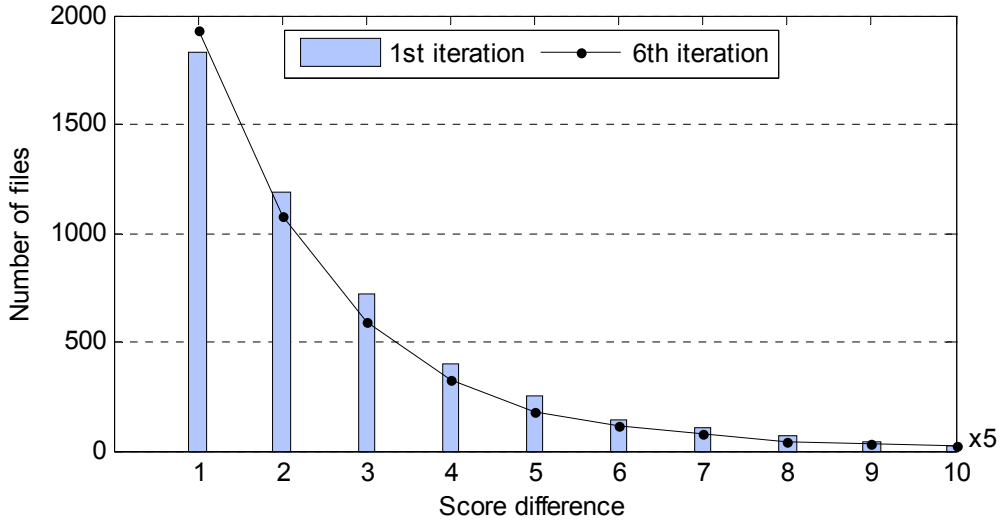

Figure 4.9: Number of files within the range of score difference. Experiments are performed on Graph2 over the development set.

4.5.4 Coverage of Word Pairs

Parameter Coverage and the Improvements

The experiments reported in the previous sections have shown that discriminative training successfully reduces the training errors on various graphs and data sets. In Section 4.5.2, we observed that the number of updates is relatively small as compared to the number of graph parameters. This means that even if the WER has been reduced, the improvement may not yield a comparable error reduction on the test set, since only a small fraction of the transition weights have been updated.

An interesting issue in the above discussion is: among the parameters updated by numerous training samples, how many of them could yield the improvement on the test set? In Section 4.4.5, this issue had been addressed experimentally, as we tried to increase the number of updated parameters; our experiments have shown that more improvements can be obtained both on the training data and the test data when a larger training data set is used. In this section, our investigation is based on the coverage of the updated parameters and the corresponding transition paths, to further explore the potential of discriminative training.

Coverage of Word Pairs

The word pairs are extracted from two consecutive words from the transcriptions in the data set, including the symbols *<s>* and *</s>* which respectively indicate the beginning and the end of an utterance. We extract the unique word pairs from the DEVSet and the TESTSet. The number of these pairs and the coverage on each data set are displayed on Figure 4.10.

The DEVSet and TESTSet respectively contain 37705+6632 and 16530+6632 word pairs, where the 6632 pairs exist on both sets. When discriminative training updates graph parameters, transition weight adjustment on the word pairs that exist in the training set only may not influence the performance on the test set. Figure 4.10 shows that this kind of word pairs is in the majority (85%). The test set contains 16530 pairs (approximately 71% of pairs on the test set) that are never be touched by discriminative training. To give further improvements, error rate reduction would have to benefit more from the word pairs that exist in both sets.

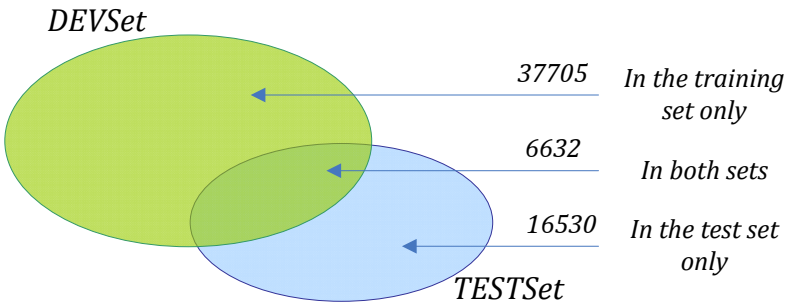

Figure 4.10: Number of occurrence and the coverage of word pairs in the training set and in the test set.

Transition Paths of Word Pairs

We thus analyze the transition paths of the 6632 word pairs that exist in the training set (approximately 15%) to examine the transition weight updates at the state level. For each word pair, we compare its transition path on the training set and that on the test set. Assume that the word pair is W_aW_b. The results in Table 4.15 show that only 408 word pairs have identical transitions on both sets. There are 1920 pairs that do not match on the transition paths of W_b. This means that they only differ at some variant (or an optional silence) of a word. However, most of the word pairs differ in the beginning of the state sequence. This implies that these pairs are produced given different word history; one may benefit from the transition of an *n*-gram while the other goes through the back-off state.

Num. of Pairs	Description
4304 (9.7%)	State sequences are different on W_a (different history)
1920 (4.2%)	State sequences are the same on W_a but different on W_b (variants)
408 (0.9%)	State sequence exact match

Table 4.15: Comparison of transition paths.

4.5.5 Discussion

In this chapter, we presented and discussed the results of a series of experiments to investigate the potential of discriminative training on integrated decoding graphs. Our proposed fast decoding techniques make the training procedure practical and yield an extremely fast alignment. To deal with the dependencies between the transition weights, a random updating scheme has been proposed, which gives a better error rate reduction. It is noteworthy that G. Zweig and his co-authors (Kuo *et al.*, 2007), using a different implementation of the same idea, have independently observed the same kind of improvement of the first pass decoding, using the data of the RT-03 campaign and state-of-the art acoustic models.

Despite these successes, several problems still affect the effectiveness of discriminative training.

Over-fitting Effect and Generalization Problem

One problem commonly encountered in many parameter estimation methods is

the over-fitting effect. It occurs when the estimation procedure continues to maximize the likelihood or to reduce the score difference, while the improvement on the training data yields an opposite result on the test data. Discriminative training on the decoding graph suffers from this drawback; experiments on various decoding graphs and data sets all yield limited error rate reduction on the test set.

To reduce this effect, several heuristics could be used to decide when to stop the training iterations: setting a threshold for the reduction of errors between iterations, using cross-validation techniques, etc. A more principled approach, yet to be precisely defined, would be to penalize the transition weight updates.

Possibility to Translate the Error Reduction

In Section 4.5.4, we made a comparison to investigate how discriminative training translates the improvements on the training data to the test data. Results show that only a small portion of word pairs in the training data have more possibility to translate the improvements to the test set. Moreover, a further comparison on the transition paths show that the majority of the word pairs either use lower order *n*-gram probability or go through the back-off transition. Finally, it is worth pointing out that this discriminative training framework does not update the acoustic parameters. If the speaking style and the accent are different from those in the training set, updating just the transition weights may not be able to yield significant improvements on the test set.

Chapter 5. Conclusion and Perspectives

This thesis investigates the modeling problem in conventional automatic speech recognition (ASR) systems under a unified training framework. In our implementation, the search space is constructed in the form of an integrated decoding graph combining various knowledge sources. Discriminative training using minimum classification error (MCE) criterion is applied to update the graph parameters so as to reduce the number of decoding errors. In this chapter, we review the main achievements of this work and discuss the perspectives it opens.

5.1 Integrated Decoding Graph

Integration of Knowledge Sources

Typical ASR systems require knowledge sources taking the form of a dictionary, acoustic models and a language model. The performance heavily depends on the way these models are estimated. However, most of the current research focuses on model optimization in isolation, thus neglecting the interdependency between different sources. Our implementations use WFST techniques to combine these sources into a single decoding graph. Based on this unified representation, we have formulated the WER minimization problem as the minimization of a function of the graph parameters. With a proper training approach, any transition weight adjustment would yield the reduction of the decoding errors.

Unified Framework and Discriminative Training

The decoding graph combining different knowledge sources can be considered as a search space involving a set of parameters. In the training framework, these parameters are taken as the input of an optimization function. Traditional parameter estimation approach applies maximum likelihood estimation (MLE) method to increase the likelihood. However, due to the lack of training samples, the MLE training approach may not yield the best performance. Therefore, instead of maximizing the likelihood, we apply discriminative training to update the graph parameters so as to minimize the errors.

5.2 Efficiency and Improvements

The main contributions of the thesis are:

A Less Graph Complexity, yielding a High Decoding Efficiency

In terms of time and space complexity, the WFST techniques are effective in building large scale systems. Our graph construction procedure achieves a great reduction in graph complexity when constructing a search space from large knowledge sources. In addition, we have proposed several fast-decoding techniques to make discriminative training practical. The first is weight pushing. Our algorithm follows the language model look-ahead principle to re-distribute the transition weights on the decoding graph so as to improve pruning. The second is the look-up table technique aiming at decreasing the search overhead of token passing in a large and complex graph. Moreover, the general principle of graph inversion allowed us to develop a sub-graph extraction approach for fast alignment. This approach extracts the possible paths from a large graph and represents the connections in a small search space. Our proposed fast-decoding techniques yield $3.0\times$RT decoding time and extremely fast $0.05\times$RT alignment speed.

Random Updating Scheme and Dynamic Adjustment

Our training framework applies MCE approach to update the graph parameters. After several training iterations have been performed, decoding errors are gradually reduced. The random updating scheme proposed in this thesis has been shown to yield better error rate reduction than deterministic update on a large decoding graph. Moreover, to give adequate parameter update for each individual training sample, the line search method is applied to dynamically determine the scale of transition weight adjustment.

Performance Improvements

With the above parameter update rule in our training framework, discriminative training yields an absolute of 3% WER reduction on the training set, and achieves a stable convergence. To investigate in more details the effect of discriminative training, we have examined the reduction of errors, the number of graph updates and the score difference between the reference and the hypothesis. Experimental results have demonstrated that not only the score

difference is getting smaller, but also the number of incorrect word pairs is reduced. A smaller, yet significant improvement of the performance is also observed on the test data. An investigation on the state transition paths of word pairs has shown that to effectively generalize to the test data, higher coverage of word pairs is required.

5.3 Future Directions

Discriminative training on large decoding graph has been demonstrated to produce less decoding errors by updating the transition weights. Several directions are conceivable to provide additional performance improvements.

Transition Weight Distribution

Transition weight distribution plays an important role on decoding performance. In this discriminative training framework, we have proposed a random updating scheme to alleviate the influence. Although the choice of the update position is very simple, more improvement could be achieved by determining the position sample-by-sample. For instance, instead of updating one single arc, transition weight adjustment could be performed along the path of a word string, where the position towards the initial state is given more weights so as to improve pruning.

Dynamic Decoding Graph

In our training framework, parameter update is performed on a static decoding graph. When the transition arcs of a word pair are along the path which is greatly determinized, parameter update is usually a compromise between the dependency and the effective adjustment. A further improvement could be achieved by creating an arc connecting states or duplicating partial paths with updated transition weights, thus providing additional degrees of freedom in the model. These new connections would make the graph non-deterministic. However, if they are created only on the path where weights are highly dependent, graph size would slightly increase while the decoding performance could be further improved.

Alternative Loss Function

Many MCE based training approach use the sigmoid function so that the several numerical search methods can be applied to find function's minimum. This

function translates the score difference into a 0-1 domain. However, the mapping curve and the domain range may not be suitable for a certain task. Several alternative loss functions such as the exponential loss or the hinge loss could be used as an alternative.

One-Best versus N-Best

Our training approach uses only the score of the one best hypothesis in the so-called anti-discriminant function. In many ASR tasks, the N-best hypotheses are often competitive with the best one. The MCE criterion allows the N-best hypotheses to be used to determine the parameter update: this approach would greatly affect the number of transitions that are updated for each training sentence; however, it would also make the computation of the gradient function more complex. The benefits of this extension remain to be experimentally assessed.

Acoustic Parameters Adjustment with Fast Alignment Process

In this training framework, discriminative training translates the score difference which combines acoustic likelihoods and language model probabilities to change the weights along the decoding path. Strictly speaking, the score difference sometimes comes much more from acoustic variability. In this case, it may not be suitable to translate this difference to the transition weight between words. Therefore, when the improvement by transition weight adjustment converges, acoustic parameters could be updated for the training samples that still produce errors. Remark that should we decide to also update the acoustic parameters, we could do so with very little computational overhead, as our sub-graph extraction techniques can be used as a very a fast alignment algorithm, even at the level of HMM states.

Appendix

A. Derivation of MCE

Following (3.5), the gradient of the loss function is derived respectively on the partial derivatives.

$$\nabla \ell_i(X,\Lambda,\Gamma_t) = \frac{\partial \ell_i}{\partial d_i}\frac{\partial d_i(X,\Lambda,\Gamma_t)}{\partial \Gamma}$$

(1) $\frac{\partial \ell_i}{\partial d_i}$:

Assume the misclassification function $d_i(X,\Lambda,\Gamma_t) = u$ and $-\gamma u + \theta = n$. The loss function can be re-written as below:

$$\ell_i = \frac{1}{1+e^{-\gamma d_i(X,\Lambda,\Gamma_t)+\theta}} = \frac{1}{1+e^{-\gamma u+\theta}} = \frac{1}{1+e^{n}} \tag{A.1}$$

Taking partial derivative of the loss function with respect to the misclassification function yields:

$$\frac{\partial \ell_i}{\partial d_i} = \frac{\partial}{\partial u}\left(\frac{1}{1+e^{n}}\right) = \frac{\partial}{\partial n}\left(\frac{1}{1+e^{n}}\right)\frac{\partial n}{\partial u} \tag{A.2}$$

which is a product of $\frac{\partial}{\partial n}\left(\frac{1}{1+e^{n}}\right)$ and $\frac{\partial n}{\partial u}$. By restoring temporary variables *n* and *u* for the loss function, the term $\frac{\partial}{\partial n}\left(\frac{1}{1+e^{n}}\right)$ is expressed as:

$$\begin{aligned}&\frac{\partial}{\partial n}\frac{1}{(1+e^{n})} = \frac{1}{1+e^{n}}\left(\frac{1}{1+e^{n}}-1\right)\\ &= \frac{1}{1+e^{-\gamma d_i(X,\Lambda,\Gamma_t)+\theta}}\left(\frac{1}{1+e^{-\gamma d_i(X,\Lambda,\Gamma_t)+\theta}}-1\right)\end{aligned}$$

yielding

$$\ell(d_i)(\ell(d_i)-1) \tag{A.3}$$

For the term $\frac{\partial n}{\partial u}$, it is obvious to have a constant form:

$$\frac{\partial n}{\partial u} = \frac{\partial}{\partial u}(-\gamma u+\theta) = -\gamma \tag{A.4}$$

Combining (A.3) and (A.4) gives:

$$\frac{\partial \ell_i}{\partial d_i} = \gamma\{\ell(d_i)[1-\ell(d_i)]\} \tag{A.5}$$

(2) $\frac{\partial d_i(X,\Lambda,\Gamma_t)}{\partial\Gamma}$:

Suppose N-best hypotheses are used and Γ is a vector that represents the transition weights along the decoding path. The misclassification function taking into account the reference W_0 and the N-best hypotheses $\{W_1,W_2,...,W_n\}$ is:

$$d_i(X,\Lambda,\Gamma_t) = -g(X,W_0,\Lambda,\Gamma_t) + G(X,W_1,W_2,...,W_n,\Lambda,\Gamma_t) \tag{A.6}$$

If acoustic model parameters are fixed, the loss function needs to be differentiated with respect to the transition weights. Thus, taking partial derivative of $d_i(X,\Lambda,\Gamma)$ with respect to Γ yields:

$$\begin{aligned}&\frac{\partial}{\partial\Gamma}(-g(X,W_0,\Lambda,\Gamma_t)) + \frac{\partial}{\partial\Gamma}G(X,W_1,W_2,...,W_n,\Lambda,\Gamma_t)\\ &= \frac{\partial}{\partial\Gamma}(-\log P(W_0\,|\,\Gamma_t)) + \frac{\partial}{\partial\Gamma}G(X,W_1,W_2,...,W_n,\Lambda,\Gamma_t)\end{aligned}$$

which can be represented by:

$$-I(W_0,s) + \frac{\partial}{\partial\Gamma}G(X,W_1,W_2,...,W_n,\Lambda,\Gamma_t) \tag{A.7}$$

where $I(W_0,s)$ indicates the number of times a transition weight s on the decoding path for word sequence W_0. The definition of anti-discriminant function $G(X,W_1,W_2,...,W_n,\Lambda,\Gamma_t)$ can be expressed as below (Juang *et al.*, 1997; Kuo *et al.*, 2002):

$$G(X,W_1,W_2,...,W_n,\Lambda,\Gamma_t) = \log\left(\frac{1}{N}\sum_{r=1}^{N} e^{g(X,W_r,\Lambda,\Gamma_t)\eta}\right)^{\frac{1}{\eta}} \tag{A.8}$$

Taking derivatives of G with respect to the transition Γ yields:

$$\begin{aligned}&\frac{\partial}{\partial\Gamma}G(X,W_1,W_2,...,W_n,\Lambda,\Gamma_t)\\ &= \frac{\partial}{\partial\Gamma}\left[\log\left(\frac{1}{N}\sum_{r=1}^{N} e^{g(X,W_r,\Lambda,\Gamma_t)\eta}\right)^{\frac{1}{\eta}}\right]\\ &= \frac{\partial}{\partial\Gamma}\left[\frac{1}{\eta}\left(\log\left(\frac{1}{N}\sum_{r=1}^{N} e^{g(X,W_r,\Lambda,\Gamma_t)\eta}\right)\right)\right]\end{aligned} \tag{A.9}$$

Suppose

$$\sum_{r=1}^{N} e^{g(X,W_r,\Lambda,\Gamma_t)\eta} = \Psi \tag{A.10}$$

(A.9) can be simplified as below:

$$\frac{\partial}{\partial \Gamma}\{\frac{1}{\eta}[\log(\frac{1}{N}\Psi)]\}$$
$$=\frac{\partial}{\partial \Gamma}[\frac{1}{\eta}(\log\Psi-\log N)]=\frac{1}{\eta}\frac{\partial}{\partial \Gamma}\log\Psi$$

and

$$\frac{1}{\eta}\frac{\partial}{\partial \Gamma}\log\Psi=\frac{1}{\eta}\frac{\partial}{\partial \Psi}\log\Psi\frac{\partial \Psi}{\partial \Gamma}=\frac{1}{\eta}\frac{1}{\Psi}\frac{\partial \Psi}{\partial \Gamma} \tag{A.11}$$

To compute $\frac{\partial \Psi}{\partial \Gamma}$ in (A.11), let

$$g(X,W_r,\Lambda,\Gamma_t)=\Phi \tag{A.12}$$

$$\frac{\partial \Psi}{\partial \Gamma}=\frac{\partial}{\partial \Gamma}\sum_{r=1}^{N}e^{g(X,W_r,\Lambda,\Gamma_t)\eta}=\frac{\partial}{\partial \Gamma}\sum_{r=1}^{N}e^{\Phi\eta}$$
$$=\frac{\partial}{\partial \Phi}\sum_{r=1}^{N}e^{\Phi\eta}\frac{\partial \Phi}{\partial \Gamma} \tag{A.13}$$

where

$$\frac{\partial}{\partial \Phi}\sum_{r=1}^{N}e^{\Phi\eta}=\eta\sum_{r=1}^{N}e^{\Phi\eta} \tag{A.14}$$

and

$$\frac{\partial \Phi}{\partial \Gamma}=\frac{\partial}{\partial \Gamma}g(X,W_r,\Lambda,\Gamma)=\frac{\partial}{\partial \Gamma}(\log P(W_r\mid\Gamma))=I(W_r,s) \tag{A.15}$$

We get

$$\frac{\partial \Psi}{\partial \Gamma}=\eta\sum_{r=1}^{N}e^{\Phi\eta}I(W_r,s) \tag{A.16}$$

That is, (A.11) can be expressed as:

$$\frac{1}{\eta}\frac{1}{\Psi}\frac{\partial \Psi}{\partial \Gamma}=\frac{1}{\eta}\frac{1}{\Psi}\eta\sum_{r=1}^{N}e^{\Phi\eta}I(W_r,s)=\sum_{r=1}^{N}\frac{e^{\Phi\eta}}{\Psi}I(W_r,s) \tag{A.17}$$

Replacing Φ in (A.17) by (A.12) gives:

$$\sum_{r=1}^{N} \frac{e^{g(x,W_r,\Lambda,\Gamma_t)\eta}}{\Psi} I(W_r,s) = \sum_{r=1}^{N} C_r I(W_r,s) \tag{A.18}$$

where

$$C_r = \frac{e^{g(x,W_r,\Lambda,\Gamma_t)\eta}}{\Psi}$$

Replacing Ψ by (A.10) , we get

$$C_r = \frac{e^{g(x,W_r,\Lambda,\Gamma_t)\eta}}{\Psi} = \frac{e^{g(x,W_r,\Lambda,\Gamma_t)\eta}}{\sum_{r=1}^{N} e^{g(x,W_r,\Lambda,\Gamma_t)\eta}}$$

The derivation of (A.9) is:

$$\frac{\partial}{\partial \Gamma} G(X,W_1,W_2,...,W_n,\Lambda,\Gamma) = \sum_{r=1}^{N} C_r I(W_r,s) \tag{A.19}$$

Thus, (A.7) can be expressed as:

$$\frac{\partial d_i(X,\Lambda,\Gamma)}{\partial \Gamma} = -I(W_0,s) + \sum_{r=1}^{N} C_r I(W_r,s) \tag{A.20}$$

Combining (A.5) and (A.20), we get

$$\nabla \ell_i(X,\Lambda,\Gamma_t) = \gamma\{\ell(d_i)[1-\ell(d_i)]\}[-I(W_0,s) + \sum_{r=1}^{N} C_r I(W_r,s)] \tag{A.21}$$

B. LM and Graph

B.1. LM in ARPA format

\data\
Ngram 1=7
Ngram 2=7
Ngram 3=3

\1-grams:

-0.6690068	</s>	
-99	<s>	-0.4973247
-0.6690068	de	-0.5351131
-0.6690068	le	-0.4973247
-0.6690068	rappel	-0.4973247
-1.146128	ses	-0.3480265
-1.146128	titres	-0.2754759

\2-grams:

-0.1249387	<s> le	0
-0.30103	de </s>	
-0.5351132	de ses	
-0.1249387	le rappel	0
-0.1249387	rappel de	-0.1760913
-0.2340832	ses titres	
-0.2340832	titres </s>	

\3-grams:

-0.1760913	rappel de </s>
-0.1249387	<s> le rappel
-0.1249387	le rappel de

\end\

B.2. Graph Representation of LM

0	1	<F>	-0.497325003
0	7	le	-0.124939002
1	12	</s>	-0.669007003
1	2	de	-0.669007003
1	3	le	-0.669007003
1	4	rappel	-0.669007003
1	5	ses	-1.14612806
1	6	titres	-1.14612806
2	1	<F>	-0.535112977
2	12	</s>	0.69897002
2	8	ses	-0.535112977
3	1	<F>	-0.497325003
3	9	rappel	-0.124939002
4	1	<F>	-0.497325003
4	10	de	-0.124939002
5	1	<F>	-0.348026991
5	11	titres	-0.234082997
6	1	<F>	-0.275476009
6	12	</s>	0.765917003
7	3	<F>	
7	9	rappel	-0.124939002
8	5	<F>	1
9	4	<F>	
9	10	de	-0.124939002
10	2	<F>	-0.176091
10	12	</s>	0.823908985
11	6	<F>	1
12			

C. Error Reduction

C.1. WER on Graph1

Iteration	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err
0	6489	75912	68.5	23.6	7.9	3.4	35	84.7
1	6489	75912	69.1	22.7	8.1	3.1	34	83.9
2	6489	75912	69.9	21.8	8.3	2.9	33	83.1
3	6489	75912	70.3	21.4	8.2	2.9	32.6	82.8
4	6489	75912	70.6	21.2	8.2	3	32.3	82.6
5	6489	75912	71	21	8	3	32	82.1
6	6489	75912	71.3	20.8	7.9	3	31.7	82
7	6489	75912	71.5	20.7	7.8	3.1	31.6	81.9
8	6489	75912	71.5	20.6	7.8	3.1	31.6	81.9

Table C.5.1: Error reduction of Graph1 on the DEVSet.

Iteration	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err
0	3307	44004	58.1	30.5	11.4	4	45.9	92.6
1	3307	44004	58.8	29.2	11.9	3.5	44.7	92.3
2	3307	44004	58.9	29	12	3.5	44.6	92.5
3	3307	44004	58.9	29.1	12	3.6	44.6	92.3
4	3307	44004	59.1	28.9	12	3.6	44.6	92.3
5	3307	44004	59.2	28.9	11.9	3.6	44.5	92.2
6	3307	44004	59.3	28.9	11.8	3.7	44.5	92
7	3307	44004	59.3	29	11.7	3.9	44.6	92.1
8	3307	44004	59.1	29.2	11.8	4	44.9	92.3

Table C.5.2: Error reduction of Graph1 on the TESTSet.

Iteration 0 indicates the baseline, which is the performance before discriminative training.

C.2. WER on Graph2

Iteration	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err
0	6489	75912	73.8	18.3	7.9	2.3	28.5	78.7
1	6489	75912	74	18.1	7.8	2.3	28.2	78.3
2	6489	75912	74.8	17.5	7.7	2.2	27.4	77.3
3	6489	75912	75.5	17	7.5	2.2	26.7	76.5
4	6489	75912	75.9	16.7	7.4	2.2	26.3	76.2
5	6489	75912	76.2	16.4	7.3	2.2	26	75.4
6	6489	75912	76.6	16.1	7.3	2.2	25.6	74.7
7	6489	75912	76.7	16	7.2	2.2	25.5	74.4
8	6489	75912	77.1	15.7	7.2	2.2	25.1	73.7
9	6489	75912	77.4	15.5	7	2.2	24.8	73.3
10	6489	75912	77.6	15.4	7.1	2.2	24.7	72.9
11	6489	75912	77.9	15.1	7	2.2	24.3	72.2
12	6489	75912	78	15.1	6.9	2.2	24.1	72
13	6489	75912	78.2	14.9	6.9	2.2	24	71.6
14	6489	75912	78.4	14.8	6.8	2.2	23.8	71.1
15	6489	75912	78.7	14.5	6.8	2.2	23.5	70.7
16	6489	75912	78.8	14.5	6.7	2.2	23.4	70.4

Table C.5.3: Error reduction of Graph2 on the DEVSet.

Iteration	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err
0	3307	44004	64.6	23.6	11.9	2.4	37.9	87.4
1	3307	44004	64.9	23.1	12	2.4	37.5	86.6
2	3307	44004	65.1	23	11.9	2.4	37.4	86.6
3	3307	44004	65.1	23	11.9	2.4	37.3	86.6
4	3307	44004	65.3	22.9	11.9	2.4	37.2	86.5
5	3307	44004	65.4	22.8	11.8	2.5	37.1	86.5
6	3307	44004	65.5	22.8	11.8	2.5	37	86.6
7	3307	44004	65.4	22.8	11.8	2.5	37.1	86.4
8	3307	44004	65.5	22.8	11.7	2.5	37	86.5
9	3307	44004	65.4	22.8	11.8	2.5	37	86.4
10	3307	44004	65.5	22.8	11.7	2.5	37	86.4
11	3307	44004	65.4	22.8	11.8	2.5	37.1	86.4
12	3307	44004	65.5	22.8	11.7	2.5	37	86.3
13	3307	44004	65.5	22.7	11.8	2.5	37	86.4
14	3307	44004	65.4	22.8	11.8	2.5	37.1	86.4
15	3307	44004	65.4	22.8	11.8	2.5	37.1	86.5
16	3307	44004	65.5	22.8	11.7	2.5	37	86.5

Table C.5.4: Error reduction of Graph2 on the TESTSet.

C.3. Performance on the Test Set

	Source of Recordings	Num. of Files	Hours of Speech
1	20041006_0700_0800_CLASSIQUE	740	0.75
2	20041006_0800_0900_CULTURE	917	0.70
3	20041007_0800_0900_INTER_DGA	996	0.80
4	20041011_1300_1400_INTER_DGA	654	0.70

Table C.5.5: Four different recordings in the test set and associated with an index.

Source	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err
1	740	10493	70.8	21.1	8.1	3.4	32.6	86.1
2	917	10448	63.4	24.1	12.6	2.3	39	86.5
3	996	12226	67.5	20.8	11.7	1.8	34.3	83.9
4	654	10837	56.3	28.6	15.1	2.3	46	95.4

Table C.5.6: WER on each source before discriminative training.

Source	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err
1	740	10493	72.4	20.1	7.5	3.8	31.4	84.3
2	917	10448	64.6	23.1	12.3	2.6	38	85.3
3	996	12226	69.2	19.9	10.8	2.2	33	82.5
4	654	10837	57.3	28.5	14.1	2.6	45.3	95.1

Table C.5.7: WER on the TESTSet after the 1st iteration is performed on TRAINSet.

Source	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err
1	740	10493	71.4	20.4	8.1	3.4	32	84.9
2	917	10448	63.8	23.4	12.9	2.3	38.5	86
3	996	12226	67.9	20.4	11.6	1.8	33.9	83.2
4	654	10837	56.3	28.4	15.2	2.2	45.9	94.6

Table C.5.8: WER on the TESTSet after the 1st iteration is performed on DEVSet.

Source	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err
1	740	10493	71.9	20.3	7.8	3.6	31.7	85.1
2	917	10448	63.8	23.3	12.9	2.3	38.5	85.8
3	996	12226	68.9	19.9	11.2	1.9	33	83.1
4	654	10837	57	27.9	15.1	2.3	45.2	94.5

Table C.5.9: WER on the TESTSet after the 6th iteration is performed on DEVSet.

D. Published Papers

Shiuan-Sung Lin, François Yvon, "Discriminative Training of Finite State Decoding Graphs", In INTERSPEECH, pp. 733-736, 2005.

Shiuan-Sung Lin, François Yvon, "Optimization on Decoding Graphs by Discriminative Training", In INTERSPEECH, 2007.

Discriminative Training of Finite State Decoding Graphs

Shiuan-Sung LIN, François YVON

GET/ENST and CNRS/LTCI, UMR 5141
lin@tsi.enst.fr, yvon@infres.enst.fr

Abstract

Automatic Speech Recognition systems integrate three main knowledge sources: acoustic models, pronunciation dictionary and language models. In contrast to common practices, where each source is optimized independently, then combined in a finite-state search space, we investigate here a training procedure which attempts to adjust (some of) the parameters after, rather than before, combination. To this end, we adapted a discriminative training procedure originally devised for language models to the more general case of arbitrary finite-state graphs. Preliminary experiments performed on a simple name recognition task demonstrate the potential of this approach and suggest possible improvements.

1. Introduction

Large-vocabulary automatic speech recognition (ASR) systems integrate three main resources: a set of acoustic models, a pronunciation dictionary, and a statistical language model. Acoustic models, usually based on the Hidden Markov Model (HMM) formalism, match a stream of acoustic parameters and predefined statistical models of acoustic units. The dictionary contains a deterministic or probabilistic mapping between sequences of acoustic units and words. The language model defines a probability distribution over word sequences, which encodes the most common local syntactic regularities. A major breakthrough has been the development of a methodology for combining these resources in a unified framework, based on the formalism of Weighted Finite-State Transducers (WFSTs, see e.g. [1] and the reference therein). In this formalism, decoding is performed using standard heuristic search procedures in an optimized finite-state transducer, yielding fast and accurate decoders.

One issue remains unsettled though, which relates to the specification of these knowledge sources. Each resource involves the estimation of myriad of parameters, which, according to common practices, are estimated in isolation, using separate training corpora. This means, for instance, that the definition of dictionary entries is performed independently from the acoustic modeling and the same goes for language models. As a result, many "small" decisions made during specification of these sources, regarding, for instance, the number and topology of HMMs, the modeling of pronunciation variants and the complexity of the language model, have to be experimentally validated, yielding a significant tuning overhead every time a system has to be set up. Another downside of this approach is that it yields unnecessarily large graphs, which are yet difficult to prune. We contend that a better integration of resources is needed during the model estimation step and propose a discriminative training methodology to achieve this goal.

Discriminative training (DT) is a general estimation technique which aims at setting model parameters so as to directly optimize the performance of a model or a function thereof, rather than the likelihood of training data. The availability of very-fast (sub real-time) decoders makes this learning strategy feasible. In the context of ASR, this methodology has been successfully applied to the estimation of acoustic [2,3] and linguistic models [3,4,5].

In this paper, we try to take the idea of DT one step further, and apply this methodology to adjust the various parameters of **the integrated decoding graph**. By working directly on a representation that includes all the knowledge sources, we expect to eventually come up with a set of parameters which will play its role better, i.e. improve the efficiency of the search procedure. Another benefit of this approach is to build search graphs which are beyond the reach of the standard combination procedure: for instance graphs where the probability of pronunciation variants depends on the language model history.

The integrated decoding graph contains all the resource parameters, and many more[1]: trying to simultaneously optimize all of them may well prove infeasible. In this paper, we thus make the simplifying assumption that acoustic model parameters are fixed and we only attempt to optimize the arc transition weights adjusting the graph to increase the discrimination capacity of the network in areas where the acoustic confusability is high. Starting with an initial graph configuration, our algorithm iteratively updates the graph parameter so as to increase the recognition rate, until convergence is reached.

This paper is organized as follows: we first introduce our discriminative training procedure, based on the minimum classification error (MCE) criterion and detail our own implementation. We then describe the task used to test this methodology and report experimental results. We finally discuss some open issues and draw perspectives for future work.

2. Discriminative training

2.1. The MCE model

In the MCE approach, the estimation of the model parameters aims at optimizing a function of the classification error rate [2]. Since the resulting optimization program does not lend itself to an analytical resolution, estimation is performed through an iterative parameter optimization procedure. In this section, we present this model, which extends the ideas originally introduced in [5], and from which we borrow the notations.

We assume that G is an integrated finite-state decoding graph, devised according to the procedures introduced in [1]. G contains two kinds of parameters: acoustic model parameters, associated with HMMs states Gaussian densities, and state transitions weights.

Given a word string W, a set of acoustic models Λ, a set of transition weights Γ and an observation sequence X, the

[1] The language model weight, the word insertion penalty…

conditional likelihood of X is approximated as the score of the best path in G for input X and output W. This score combines the individual acoustic likelihoods and transition weights according to:

$$g(X,W,\Lambda,\Gamma) = a(X,W,\Lambda) + b(W,\Gamma) \quad (1)$$

where $a(X,W,\Gamma)$ is a sum of acoustic likelihoods and $b(W,\Gamma)$ is a sum of transition weights. Speech decoding consists in finding the word sequence W_1 maximizing g over all possible word sequences.

If W_0 is the known correct word sequence, the performance of the recognizer can thus be expressed as a function of the difference between the score of the correct sequence and that of the best hypothesis. For a given input vector, we thus define the misclassification function as:

$$d_f(X,\Lambda,\Gamma) = -g(X,W_0,\Lambda,\Gamma) + g(X,W_1,\Lambda,\Gamma) \quad (2)$$

An erroneous recognition hypothesis thus simply translates into a strictly positive value for d_f, meaning that the correct word sequence is not the top ranking one according to g. To formulate an optimization procedure based on the misclassification error function l_f, the differentiable class loss function is introduced as:

$$l_f(X,\Lambda,\Gamma) = l(d_f(X,\Lambda,\Gamma)) = \frac{1}{1+\exp(-\gamma d_f(X,\Lambda,\Gamma)+\theta)}$$

where γ and θ are parameters which control respectively the slope and the shift factor of the sigmoid function. A standard iterative gradient procedure can then be defined, based on the following update rule for the set of transition weights:

$$\Gamma_{t+1} = \Gamma_t - \varepsilon.\nabla l_f(X,\Lambda,\Gamma_t) \quad (4)$$

If we consider that acoustic model parameters are fixed, the loss function needs only be differentiated with respect to the weight transition probabilities. Given that $b(W,\Gamma)$ is a mere sum of transition weights, the mathematical derivation exactly follows that of [5], yielding:

$$\nabla l_f(X,\Lambda,\Gamma_t) = \frac{\partial l_f}{\partial d_f}\frac{\partial d_f(X,\Lambda,\Gamma_t)}{\partial \Gamma} \quad (5)$$

where $\frac{\partial l_f}{\partial d_f} = \gamma l(d_f(X))(1-l(d_f(X))$.

We continue to work out the mathematics by taking partial derivatives of the term $d_f(X,W,\Gamma)$ with respect to the transition weight vector Γ, finally yielding:

$$\frac{\partial d_f(X,\Lambda,\Gamma_t)}{\partial \Gamma} = -I(W_0,s) + I(W_1,s) \quad (6)$$

where $I(W,s)$ represents the number of occurrences of the transition weight s on the best decoding path for W. This procedure can further be generalized by considering the N-best hypotheses, rather than the single best one (again, refer to [5] for the details).

Altogether, the training procedure consists in iteratively scanning the training corpus until convergence, using for each training sentence the update rule in (5).

2.2. Implementation

The training data only contains the correct output word sequence: the corresponding reference path is computed using a forced alignment procedure. If a transition is more frequent in the reference path than in the hypothesis path, its weight is increased; otherwise, it has to be decreased. For transitions which exist with the same frequency in the reference and in the hypothesized path, no update is performed.

Based on this general principle, many training regimes can be considered: in the experiments reported above, we made the following choices: (i) in all cases, the HMM internal transitions are frozen and are not updated: this allowed us to limit the graph expansion at the phone level; (ii) amongst the remaining transitions, we only update the ones whose output label is not empty. Two different update strategies were considered: in the first one, we do not update transitions which appear (albeit with different counts) in the reference and hypothesis path. In the second one, we update all the transitions having a different frequency, according to the update rule of equation (6).

All the experiments reported above were performed using our own decoder: a full search (involving no pruning) runs in about 0.8RT on a 3.0 GHz Pentium IV. The decoding graphs were prepared using the FSM Toolkit [7].

3. Experiments

The preliminary experiments reported hereafter were carried out on a simple name recognition task. Our main purpose was to get a better grasp of the behavior of the training procedure and of the influence of the various parameters on a well-understood task, using a relatively simple decoding graph. We first present the task and the database, before reporting the results of these experiments.

3.1. Task and database

The task consists in recognizing isolated sequences consisting of a proper name followed by its spelling, as illustrated by:

frana F-R-A-N-A

The recognizer's output is accordingly composed of two parts: a sequence of phone labels, followed by a sequence of letter labels. No dictionary look-up is performed to match the output with existing names; performance (WER) is simply computed as a function of the number of corrected recognized symbols (phones and letters) in the output. The main motivation for experimenting with this small vocabulary tasks was (i) to assess the influence of the various parameters involved in the procedure and (ii) to be able to analyze the output decoding graph and get a better understanding of the benefits of this training procedure.

Data for this task was extracted from the 'spelled word' category of the Swiss-French Polyphone database [8]. After cleaning invalid[2] entries, the database was randomly split into a discriminative training set (8427 names) and a testing set (1150 names), representing respectively 14.16 and 1.95 hours of recordings. Performance measurement being based on a phonetic match between hypotheses and references, each orthographic name was automatically converted into a sequence of phones, using a pronunciation dictionary whenever applicable and automatic pronunciation procedures otherwise.

For these tasks, acoustic models of varying sizes were estimated using the 'phonetic-rich' category of Polyphone (totaling 49.79 hours of speech): for each of our 39+2 phonetic units, context-independent models containing from 16 to 64 Gaussian mixtures per HMM state were considered.

The initial decoding graph was set up as follows: two language models were separately trained, one for phone sequences and one for letter sequences. The former was trained using automatic phonetic transcript of the 'phonetic-rich' category items in Polyphone: this yielded a phone-based language model encoding a general distribution of possible

[2] Entries for which one subpart is missing or which contain non-conventional spelling directives.

phone sequences. In contrast, due to lack of data, the letter language model was built using the same dataset that is used for discriminative training, i.e. spellings extracted from the 'spelled-name' category: this model already integrates some knowledge regarding the typical sequences of letters that occur in names. Each of these models was trained conventionally, using Maximum Likelihood estimation and standard smoothing procedures. Table 1 gives a quantitative description of the phone and letter language models, including the total corpus size used for estimation, their number of parameters and perplexity (PP).

	Corpus	Bigram		Trigram	
		Size	PP	Size	PP
Phone	1,411,699	1237	29.69	16865	10.77
Letter	77,185	1078	13.33	5027	9.51

Table 1: Phone and letter language models

Each language model is then turned into a weighted finite state acceptor (WFSA). The letter WFSA is then composed with a FST mapping letters to their spelling. The resulting WFST is further determinized[3] and concatenated with the phone WFSA to produce a phone-based decoding graph. This construction is illustrated on Figure 1. The bigram graph contains 516 states and 2,198[4] arcs; for the trigram graph these numbers are respectively 8,154 and 32,467.

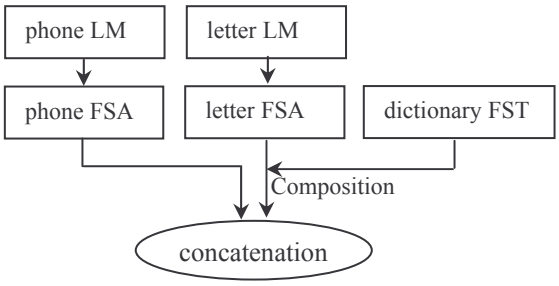

Figure 1: Baseline graph construction

3.2. Parameters selection

Before experimenting with the discriminative training procedure, we performed a number of experiments aiming at setting two parameters of the procedures: γ, which controls the slope of the sigmoid function, and ε, which is the increment parameter of the gradient descent. For the purpose of this study, we assumed that α, the language model weight, and the word insertion penalty, δ, are fixed, taking values α = 0.23, δ = 0.6 for the phone part and δ = 0.4 for the letter part. We also set θ=0.

The rationale for finding a reasonable parameter set is based on the following remarks. When the difference d_f, between reference and hypothesis scores is large, the loss function tends to one, and the corresponding update tends to zero (see equation (5). A first decision was made to set an upper bound on the value of this difference: sentences whose misclassification score exceed this threshold are not considered during training. Based on an analysis of the distribution of for d_f, in the training data, this upper bound was fixed so as reject only a small portion of the training data (about 3% of the sentences). Given the range of possible values for for d_f, we then choosed γ = 0.01 so as to control the value of the gradient. We finally set ε=10 to get a total increment factor of 0.1 for the gradient descent. With these parameters, the update factor for a transition lies between 0.011 and 0.025 on a log scale.

Choosing a smaller value for γ would have the effect of increasing the average update factor, yielding a faster convergence of the procedure, at the risk however of rendering the process unstable. While more experimental work is required to fine tune these parameters, the values used for our experiments seemed to yield a reasonable convergence rate.

3.3. Results

Baseline results are obtained with the 32-mixture acoustic models before starting the discriminative training procedure. After each training iteration, the graph is dumped on disk and used for testing. For each of our experiments, 5 training iterations were performed.

Figure 2 plots the evolution of recognition performance after each training iteration for the first training regime (see Section 2.2). Identical results were obtained with the second, more costly, training regime. Additional control experiments carried out with 16 and 64-mixture models also yielded a similar decrease pattern.

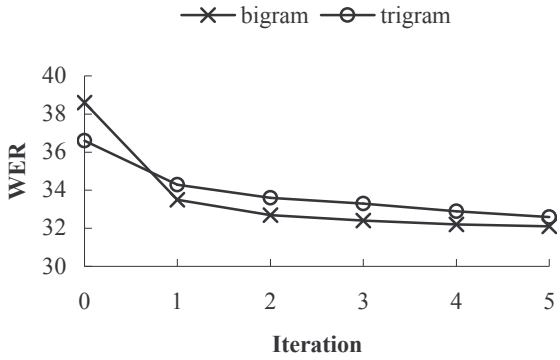

Figure 2: Evolution of the WER

For the bigram curve, most of the performance increase is achieved after one iteration, and the remaining iterations only bring a small improvement. Overall, the performance increases by 6.5 points for the bigram model, by 4 points for the trigram model. One iteration of training already brings a very significant increase (4% absolute for the bigram graph) in performance. The discriminatively trained bigram graph significantly outperforms the baseline trigram graph, even though it contains about ten times less parameters. This illustrates the uselessness of many parameters in the trigram baseline graph. After five iterations of training, the trigram graph is still lagging behind. This may be explained by the fact that this graph contains a larger number of parameters, resulting in a slower convergence rate: after 5 iterations, the performance continues to increase, albeit at a small pace.

Another perspective on the convergence of the algorithm is given by a closer examination of the update pace: for the bigram graph, after five iterations of training, the number of updates stops its decrease: the number of updates per iterations remains very high (about 120,000), suggesting that the same weights are repeatedly increased and decreased.

[3] Control experiments were ran without determinization and show no significant difference on this task.

[4] The fully expanded network would thus include about three times more states: the graph is here only expanded at the level of phones.

This stabilization is not observed for the trigram graph. Again, more experimental work is required to confirm these preliminary observations.

To investigate in more details the effect of discriminative training, we examined the 30 most frequent confusion pairs in the bigram baseline system. This tracking was performed independently on the phone part and on the letter part of the decoded utterances. Figure 3 and 4 display the evolution of the number of confusions for these pairs before and after training. As clearly appears on these figures, discriminative training manages to significantly reduce these frequently occurring confusions. For the phone part, only a handful of pairs show an increase in confusion. For most of the remaining ones, we get an improvement, which is all the more substantial as the related phones have a different distributional pattern: this is the case, for instance, of the pairs ɑ/a, a/t, i/t… In contrast, some pairs which are both acoustically and distributionally very similar remain difficult to discriminate: the top remaining errors in the phone part are: e/ɛ; o/ɔ; ɔ/o; b/d; t/p...

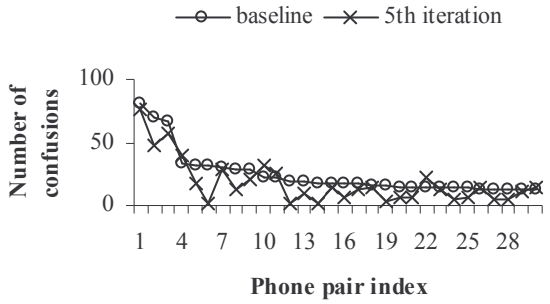

Figure 3: Evolution of the top 30 phone confusion pairs.

The same pattern of improvement is observed for spelled letters: some confusions are substantially reduced (e.g. between E (/ø/) and 2 (/dø/), or between A (/a/) and K (/ka/); some pairs nonetheless remain difficult to discriminate and continue to account for a large number of errors: B (/be/) vs. D (/de/), L (/ɛl/) vs. N (/ɛn/), M (/ɛm/) vs. N, S (/ɛs/) vs. F (/ɛf/)...

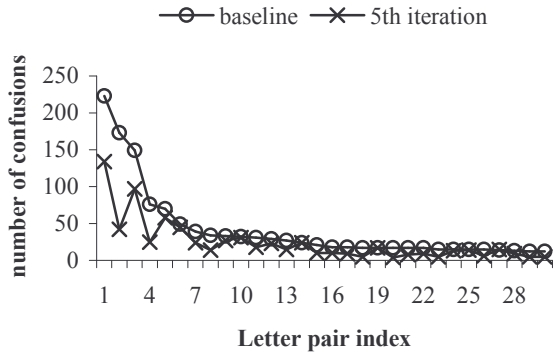

Figure 4: Evolution of the top 30 letter confusion pairs.

4. Conclusions

In this paper, we presented a discriminative training procedure aiming at adjusting the various parameters of a decoding graph so as to directly optimize the recognition performance. Results obtained on a simple name recognition task demonstrate the effectiveness of this methodology, illustrated by a 6.5% absolute improvement of the word error rate on a bigram graph. A close examination of the optimized graph reveals that discriminative training has the expected effect of facilitating, whenever possible, the discrimination between acoustically confusable phone and letter pairs. We are currently experimenting with alternative training regimes, such as considering all the available transitions for update, or taking the N-best recognition hypotheses into account. We are also trying to revise the graph update procedure so as to ensure that updates have a "local" impact: to see why this is important, consider the situation when the updated transition is a transition leaving a back-off state: its increase (or decrease) will impact the score of many paths which should not be changed. The idea is thus to introduce new states in the graph so as to guarantee that updates only affect the score of current reference and hypothesis paths. Additional experiments using richer acoustic models and larger vocabulary and language models are also required to assess more precisely the benefits of this new training strategy.

5. References

[1] Mehryar Mohri, Fernando C. Pereira and Michael Riley, ``Weighted Finite-State Transducers in Speech Recognition" *Computer Speech and Language, 16(1):69-88, 2002*.

[2] Biing-Hwang Juang, Wu Chou and Chin-Hui Lee, ``Minimum Classification Error Rate Methods for Speech Recognition", *IEEE Transactions on Speech and Audio processing, 5:3, pp. 266-277, 1997*.

[3] Ralf Schluter, Wolfgang Macherey, Boris Muller and Herman Ney, ``Comparison of Discriminative Training Criteria and Optimization Methods for Speech Recognition", *Speech Communication. Vol. 34, pp. 287-310, 2001*.

[4] Zheng Chen, Mingjing Li and Kai-Fu Lee, ``Discriminative Training on Language Model", *Proc. ICSLP'00, Beijing, China, 2000*.

[5] Hong-Kwang Jeff Kuo, Eric Fosler-Lussier and Hui Jiang, Chin-Hui Lee, ``Discriminative Training of Language Models for Speech Recognition", *Proc. ICASSP'02, Orlando, Florida, 2002*.

[6] Brian Roark, Murat Saraclar and Michael Collins, ``Corrective Language Modeling For Large Vocabulary ASR With The Perceptron Algorithm", *Proc. ICASSP 2004. Montreal, Canada, 2004*.

[7] Mehryar Mohri, Fernando C. Pereira and Michael Riley, ``General-purpose Finite-State Machines Software Tools". http://www.research.att.com/sw/tools/fsm, AT&T research, 1997.

[8] Jean-Luc Cochard, Gérard Chollet, Philippe Langlais and Andrei Constantinescu. "Swiss-French Polyphone: a Telephone Speech Database to develop Interactive Voice Servers", *Linguistic Databases*, CSLI Publications, John Nerbonne (Ed.), 1997.

Optimization on Decoding Graphs by Discriminative Training

Shiuan-Sung LIN, *François YVON*

GET/ENST and CNRS/LTCI, UMR 5141

lin@tsi.enst.fr, yvon@enst.fr

Abstract

The three main knowledge sources used in the automatic speech recognition (ASR), namely the acoustic models, a dictionary and a language model, are usually designed and optimized in isolation. Our previous work [1] proposed a methodology for jointly tuning these parameters, based on the integration of the resources as a finite-state graph, whose transition weights are trained discriminatively. This paper extends the training framework to a large vocabulary task, the automatic transcription of French broadcast news. We propose several fast decoding techniques to make the training practical. Experiments show that a reduction of 1% absolute of word error rate (WER) can be obtained. We conclude the paper with an appraisal of the potential of this approach on large vocabulary ASR tasks.

Index Terms: discriminative training, decoding graph, weighted finite-state transducer

1. Introduction

Most of the ASR research effort focuses on improving the performance of one specific component of the system, with the hope that it will improve the overall performance. This approach, does not take into account the dependency between the various knowledge sources. For instance, the design and estimation of acoustic models critically depends on the word pronunciation(s) that actually occur in the dictionary: a small dictionary might bear with simple models, while a large vocabulary system will require complex ones. Language modeling is performed separately from other resources using different, and often much larger, corpora. In addition, most modeling approaches perform parameter estimation separately for each resource, assuming that all the other parameters are fixed. The interdependency among knowledge sources is thus ignored, yielding under optimal performance.

Reliable estimation procedures for the various model parameters remain therefore a key issue for obtaining good performances. In the literature, the most commonly used estimation strategy is maximum-likelihood estimation (MLE). This approach attempts to estimate the parameters such that the likelihood of the training data is maximized. The principles of MLE rely on the availability of large training samples; however, the improvement in training does not always translates into better decoding performance [2]. This observation has led researchers to explore other estimation techniques, notably such as discriminative training (DT) techniques. In contrast to MLE, DT aims at optimizing the separation between good and bad hypotheses on the training samples. It is performed by formulating an objective function that, in some ways, penalizes parameters that are liable to confuse correct and incorrect words.

In the past years, various DT criteria such as maximum mutual information (MMI) [3] and minimum phone error (MPE) [4] have greatly improved the estimation of acoustic models. Discriminative techniques also have been shown to yield significant improvements in language modeling, such as minimum classification error (MCE) [2], minimum sample risk (MSR) [5], and reranking techniques based on the perceptron algorithm [6].

Recent advances in ASR systems have also promoted the use of weighed finite-state transducers (WFST), as a common underlying formalism for representing homogeneously the various knowledge sources [7]. A WFST is a type of finite state machine, whose state transitions carry input symbols, output symbols and arbitrary weights. A transition sequence from the initial state to the final state of a WFST represents a weighted mapping from inputs to the corresponding outputs. Using WFSTs to represent the various ASR components, different resources can be easily integrated in a single graph. Various optimization algorithms, such as determinization[1] and minimization, can be applied off-line, prior to decoding. These optimization techniques eliminate redundant arcs and states to yield an equivalent but more efficient graph.

Our previous work [1] proposed to combine these two techniques into a unified training framework: WFST are used to combine various sources into a finite-state graph, whose parameters are trained using DT. Our positive results on a simple name recognition task were recently confirmed in [8], which reports significant error rate reduction on a large vocabulary application.

In this paper, we extend our previous work to a much more complex large-vocabulary task, and analyze the achieved performance from several different perspectives. We detail several decoding techniques that had to be introduced in order to make training practical. In addition, we also investigate the strengths and shortcomings of this approach and discuss the new directions it opens.

2. Discriminative training on decoding graphs

2.1. MCE criterion

We follow here the notations of our previous work [1]. Assume G is an integrated finite- state graph. G contains two kinds of parameters: state transition weights and acoustic model parameters, which are associated with HMM states Gaussian densities. Given a word string W, a set of acoustic model Λ, a set of transition weights Γ and an observation sequence X, the conditional log-likelihood of X is approximated as the score of the best path in G for input X

[1] Determinization of finite-state transducers is not a well-defined notion [7]: in conformance with the literature, we will use the term here in a rather loose sense to denote exact and heuristic techniques aiming at removing duplicate paths on the input type of the transducer.

and output W. This score combines the individual acoustic log-likelihoods and transition weights along the decoded path:

$$g(X,W,\Lambda,\Gamma) = a(X,W,\Lambda) + b(W,\Gamma) \quad (1)$$

where $a(X,W,\Lambda)$ is the sum of acoustic log-likelihood and $b(W,\Gamma)$ is the sum of transition weights along the path from the starting state to the final state. Speech decoding consists of finding a word hypothesis W_{hyp} which maximizes g over all possible word sequences. If W_{ref} is the correct word sequence, the performance of the recognizer can be expressed as a function of the score difference between the reference and the best hypothesis. For a given input vector, the misclassification function is defined as:

$$d(X,\Lambda,\Gamma) = -g(X,W_{ref},\Lambda,\Gamma) + g(X,W_{hyp},\Lambda,\Gamma) \quad (2)$$

An erroneous recognition hypothesis simply translates into a positive value of $d(X,\Lambda,\Gamma)$, meaning that the correct word sequence is not the top ranking one according to g. The next step is to define a continuously differentiable loss function, $l(X,\Lambda,\Gamma)$, integrating the misclassification measure $d(X,\Lambda,\Gamma)$:

$$l(d(X,\Lambda,\Gamma)) = \frac{1}{1+\exp(-\gamma d(X,\Lambda,\Gamma)+\theta)} \quad (3)$$

where γ and θ are the parameters which control respectively the slope and the shift factor of the sigmoid function. Using generalized probabilistic descent (GPD) algorithm, a standard iterative procedure can be defined, based on the following parameter update rule for the transition weights:

$$\Gamma_{t+1} = \Gamma_t - \varepsilon \nabla l(X,\Lambda,\Gamma_t) \quad (4)$$

Assuming that the acoustic model parameters are fixed, the loss function needs to be differentiated only with respect to the transition weights. The derivation of (4) yields:

$$\nabla l_i(X,\Lambda,\Gamma_t) = \frac{\partial l_i}{\partial d_i}\frac{\partial d_i(X,\Lambda,\Gamma_t)}{\partial \Gamma} \quad (5)$$

Viewing Γ as a transition weight vector and taking partial derivatives of $di(X,\Lambda,\Gamma)$ with respect to Γ, the derivation of (5) finally yields:

$$\frac{\partial l_i}{\partial d_i} = \gamma l(d_i)(1-l(d_i)) \quad (6)$$

$$\frac{\partial d_i(X,\Lambda,\Gamma_t)}{\partial \Gamma} = -I(W_{ref},s) + I(W_{hyp},s) \quad (7)$$

where $I(W,s)$ represents the number of transition weight s on the best decoding path for W.

2.2. Parameter update rule

In MCE training, the parameters that have to be carefully selected are γ and ε, which respectively represent the slope of sigmoid function, that transfers a misclassification measure to the 0-1 domain, and a scale factor to update the parameters. Also recall that training is performed on a weighted finite-state graph, where each word is represented by a sequence of phones. A first requirement is thus to recover, through alignment, the complete (i.e. phone level) best path for the reference. In addition, implementing the update rule (7) requires to address two issues: 1) the choice of n-gram terms from the hypothesis and the reference word strings and 2) the arc position for transition weight update.

Since a correctly decoded word may follow an incorrect word, meaning that parameter update is based on an incorrect word history, we focus on the relationship between two consecutive words rather than considering a certain word history. The choice of the updated arc has a great influence on the overall transition weight distribution. In this implementation, the arc for parameter update is randomly chosen by random sampling with uniform probability amongst all possible candidates.

In addition, the value of γ is selected by relating the effect on the parameter adjustment to the score difference. Our experiments suggest that γ=0.02 is a suitable choice. The learning rate ε is determined dynamically for each training sample, using the line-search technique, so as to give an effective parameter update while keeping stable convergence.

2.3. Implementation

The decoding graph is built by compiling knowledge sources from typical ASR systems, namely the dictionary, the acoustic models and the language model, as weighted finite-state transducers (WFST). Using WFST techniques, the composition allows different level's representations (from a word to pronunciation sequence(s) and to associated HMMs) to be combined in a single graph. The determinization is used to reduce the graph size by eliminating the redundant paths, yielding an equivalent, yet more efficient graph. An optional silence (short pause model) is added at word boundaries; finally, a word insertion penalty (WIP) is added to balance the length of output word sequences.

Our decoder performs the search by using the general token passing [9] over a finite-state network. To speed up token propagation over ε-transitions, we use a look-up table to store the ε-closure relationship between states. This table is built off-line by traversing the graph and recording ε paths. Using a look-up table, a token is easily re-directed to its destination states without having to re-search the graph.

Alignment is different from decoding: the search procedure must find in the graph all the path which output a given word string W. Conceptually, this amounts to intersecting the output language of the WFST with W: our approach thus uses the general principle of graph inversion, which exchanges the input and output symbols on every transitions. By searching for W in this inverted graph, the relevant phonetic sequences can be computed in advance. The extracted graph generally consists of hundreds of states and arcs, no matter how large the original graph is.

Weights in the WFST can be distributed in many ways, which still result in an equivalent WFST. In typical ASR systems, it has been demonstrated that if the probability is properly redistributed, both the pruning efficiency and the search speed can be improved [10]. Our weight pushing algorithm is conceptually similar to the "tropical semiring" algorithm introduced in [7] and proceeds as follows: the graph is first reversed (i.e. all transitions are reversed); for all the states, the maximum transition weight among the incoming arcs is pushed towards the word boundary; finally the graph is reversed again. Some caution is taken to prevent weights from becoming too small, which would penalize too strongly the corresponding transition. This algorithm has a time complexity comparable with the algorithm of [10] but is more space efficient, as it is linear in the number of states.

3. Experiments

3.1. Experimental setup

Our experiments are carried out on a French radio broadcast news database ESTER [11]. Approximately 62.88 hours of manually transcribed data (TRAINSet) was used for training the parameters of acoustic models and a 3-gram language model (LM). The development set (DEVSet) and test set (TESTSet) respectively contains 5.3 and 2.95 hours of audio. Transcriptions containing out-of-vocabulary (OOV) words were removed from TRAINSet and DEVSet.

Our original HMM set contains 21466 acoustic models. Additionally, 1369 models are synthesized according to the decision tree by state-tying. Each model consists of 3 states, except the short pause model (one-state model) used to model the short inter-word silence. There are a total of 6238 distinct states, each of which is associated with a 39-dimensional probability density function taking the form of a mixture of 32 Gaussians, assuming a diagonal covariance.

Two 3-gram LMs are used to construct the decoding graphs: one is the above-mentioned 3-gram model; the other is a much larger 3-gram obtained by linear interpolation of several models trained on archives of the newspaper *LeMonde*, covering approximately 400M words. Both LMs contain 65k words of vocabulary. The resulting graphs respectively contain 824,845 states for 1,655,723 arcs and 6,997,044 states for 19,676,349 arcs. The so-called "fudge" factor α is used to balance the acoustic and language model scores in the log domain. Using empirically determined values for α, γ and WIP, the baseline word error rate for each decoding graph is shown in the following table:

LM	2-gram	3-gram	perplexity	Baseline
Graph1	248739	102461	158.92	45.9
Graph2	5052090	4033834	86.42	37.9

Table 1. *The number of n-grams, perplexity and baseline WER of individual decoding graphs.*

The decoder used in our experiments runs respectively at 0.7×RT and at 3×RT on Graph1 and Graph2 on a 3.6Ghz Xeon processor[1]. The alignment procedure runs in 0.05×RT. Faster decoders have been reported in the literature (eg. [12]), but these decoders only need to keep track of the word history of hypotheses. In these cases, crucial information for DT, such as the complete state transition sequence, cannot be recovered. Our decoder has to store the full state history of the top 20k candidate hypotheses at each time frame, yielding an extra-layer of book-keeping activity: this explains why we could not achieve real-time decoding speed on very large graphs.

3.2. Results

3.2.1. Deterministic versus random update

Consecutive words extracted from the reference and hypothesis word strings are represented as sequences of phones, and the MCE criterion does not precisely prescribe the exact transition weight to update. This leaves us with several options: 1) update all the weights along the transition paths, or 2) select (deterministically or randomly) one arc to update. 3) for paths which contain a back-off transition, an option would be to leave the weight unchanged when an update would yield a higher back-off probability than the *n*-gram probability.

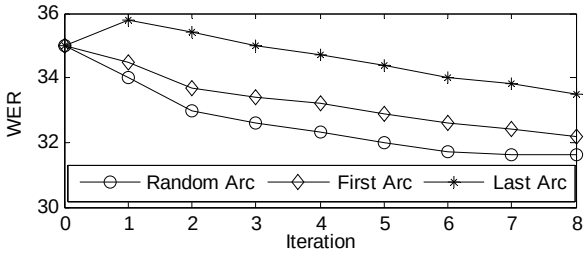

Figure 1: *Performance of different updating schemes.*

Our implementation treats back-off transitions as regular ones, and considers only one arc from the transition path for parameter adjustment. Experiments are performed using Graph1, starting from the baseline (WER=35) on the DEVSet. Two deterministic updating schemes are compared: update on the first arc and update on the last arc (see Figure 1). The random update scheme not only converges more quickly but also achieves a lower WER, confirming our previous findings [1]. This strategy is used in all the following experiments.

3.2.2. Error rate reduction on a large graph

A decoding graph contains a large number of parameters. Some of the parameters, notably the transition weights, are not entirely independent. This makes the training process difficult to reach the minimum error rate. However, a large graph usually provides a better baseline for DT. Therefore, given the same amount of training data, we compare two graphs of different sizes: DT is performed using DEVSet on Graph1 and Graph2. TESTSet is used to evaluate the performance of decoding graph after parameter adjustment. After 8 training iterations, we get the results displayed in Table 2.

	DEVSet	TESTSet
Graph1	35→31.6 (-3.4)	45.9→44.9 (-1)
Graph2	28.5→25.1 (-3.4)	37.9→37 (-0.9)

Table 2. *WER reduction on the development set and on the test set, using graphs of different size.*

The improvement on the DEVSet is similar for both graphs, with Graph2 starting from a much better baseline. On the TESTSet, WER reduction is smaller (:1% absolute), half of which is obtained after the 1st iteration itself.

In practical applications, a graph may be constructed from a language model containing a large number of *n*-grams, while the number of word pairs in the development set, which is used to optimize the graph, is comparatively much smaller. This means that DT updates only a small portion of the transition weights (no matter the number of iterations). In the next section, we use a larger training set for DT, so as to better evaluate the potential of this approach.

3.2.3. Training with a larger data set

These experiments are carried out using Graph2. One single training iteration is performed using the entire 62 hours of

[1] These figures are obtained using pre-computed acoustic likelihoods. We estimate that the on-line computations of likelihoods would increase the run-time by less than 0.7×RT for decoding and by about 0.1×RT for alignment.

TRAINSet for DT. This dataset contains approximately 11 times more word pairs, out of which 7 times more *different* word pairs than DEVSet. Results are presented in Table 3. With this larger dataset, decoding results on the test set give a total of 1.1% absolute WER reduction *in the 1st iteration*, approximately 3 times more than that obtained using DEVSet. Based on our experiments, it is likely that running additional training iterations with the large graph could bring us another 1% absolute WER reduction on the test set.

	Sub.	Del.	Ins.	WER
DEVSet	23.1	12.0	2.4	37.5
TRAINSet	22.8	11.2	2.8	36.8

Table 3. *WER reduction using different training data.*

4. Discussion and Improvements

Our experiments have shown that DT has the potential of effectively reducing the WER, even for large-vocabulary tasks. A first worry is the stability of our iterative optimization procedure: a close examination of the training log reveals that the number of updates is gradually reduced on both graphs; yet, even after 8 training iterations, the number of parameter updates performed during one iteration remain substantial (in the tens of thousands), suggesting that some transition weights do not completely stabilize.

We have explained that DT reduces the training errors by re-distributing the transition weights, yielding similar output strings to the reference, and smaller score differences between the reference and the best hypothesis. This is confirmed by the analysis of the score difference between the reference and the best hypothesis of the DEVSet: after 6 training iterations, the score of the reference has increased for almost all the training samples; and 217 originally erroneous training samples (3.34% of the training sentences) are completely corrected (reference and hypothesis are fully identical).

Results on the test set are positive, albeit by a smaller margin, suggesting that our algorithm is not generalizing well. In fact, a study of the word pairs that occur both in the test and in the training set reveals that about 40% of the word pairs in the former also occur in the latter. The situation is in fact much worse: even if an update is performed on a transition u→v during training, it might still not be useful during testing due to the difference in language model history (e.g. *wuv* appears in the training set and *w'uv* in the test set). Finally, only about 2.5% of the total number of word pairs in the test set are updated during training, clearly demonstrating the fact that our parameter update procedure can only provide with a limited improvement on this dataset.

Throwing in more training data is an effective, albeit computationally demanding, remedy to this situation. The improvements to our training procedure that are really needed should make the parameter updates less local: this is in fact what happens with traditional LM estimation procedures: any new instance of *uvw* will increase the likelihood of *w* in the context of *uw*; it will also increase, by a smaller margin, the likelihood of *w* following *v*. Two ways to make parameter update less "local" are being explored: one is to consider the top-N hypotheses rather than just the best one (see [13]); the second is to consider updating arcs *on every possible alignment* of the reference, rather that just the best one.

5. Conclusion and perspectives

DT on large decoding graph has been shown to be both computationally tractable and effective. In this training framework, parameter optimization is performed on a static decoding graph, whose transition weights are iteratively adjusted. We have presented the results of experiments on large vocabulary tasks which confirm the findings of [8] and discussed the strength and shortcomings of our DT procedure. Two other improvements are foreseen: 1) to get a fine-grained control of the effect of parameter updates. On a compact WFST, updating of one parameter changes the score of all the paths that use this transition, which can prove detrimental to the overall performance. 2) to create (and update) new transitions as needed, so as to minimize this effect; the downside being an increase of non-determinism in the graph.

Finally, we have thus far consider the acoustic parameters to be fixed; for some training samples, though, the acoustic mismatch is so bad that updating the transition weights only cannot recover the reference, or would do so at the price of unreasonable weight changes. Better ways to interleave acoustic and linguistic training are definitely needed to accommodate this kind of situation.

6. References

[1] S.S. Lin and F. Yvon, "Discriminative Training of Finite State Decoding Graphs," Proc. InterSpeech, pp. 733-736, 2005.

[2] Z. Chen, M.J. Li and K.F. Lee, "Discriminative Training on Language Model," Proc. ICSLP, 2000.

[3] L. R. Bahl, F. Jelinek and R. L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition", IEEE Trans. on Pattern Analysis and Machine Intelligence., vol(5), pp. 179-190, 1983.

[4] D. Povey and P. C. Woodland, "Minimum Phone Error and I-Smoothing for Improved Discriminative Training," Proc. ICASSP, pp.105-108, 2002.

[5] J. Gao, H. Yu, W. Yuan and P. Xu, "Minimum Sample Risk Methods for Language Modeling," Proc. HLT/EMNLP, 2005.

[6] B. Roark, M. Saraclar and M. Collins, "Corrective Language Modeling for Large Vocabulary ASR with the Perceptron Algorithm," Proc. ICASSP, 2004.

[7] M.Mohri, "Finite-State Transducers in Language and Speech Processing," Computational Linguistics, 23:2, pp. 269-311, 1997.

[8] H. K. Jeff Kuo, B. Kingsbury and G. Zweig, "Discriminative Training of Decoding Graphs for Large Vocabulary Continuous Speech Recognition," Proc. ICASSP 2007.

[9] S.J. Young, N.H. Russel, and J.H.S. Thornton, "Token Passing: a Simple Conceptual Model for Connected Speech Recognition Systems," Technical Report, Cambridge University: 1989.

[10] M. Mohri and M. Riley, "A Weight Pushing Algorithm for Large Vocabulary Speech Recognition," Proc. EuroSpeech, pp. 1603-1606, 2001.

[11] G. Gravier, J.-F. Bonastre, S. Galliano, E. Geoffrois, K. McTait and K. Choukri. "The ESTER evaluation campaign of Rich Transcription of French Broadcast News", Proc. LREC, 2004

[12] G.Saon, D.Povey, and G.Zweig, "Anatomy of an Extremely Fast LVCSR Decoder," Proc. InterSpeech, pp. 549-552, Lisbon, 2005.

[13] H.-K. Jeff Kuo, E. Fosler-Lussier, H. Jiang and C.-H. Lee, ``Discriminative Training of Language Models for Speech Recognition", Proc. ICASSP'02, Orlando, Florida, 2002.

Bibliography

Aho, A. V., Sethi, R., and Ullman, J. D., (1986) "Compilers: Principles, Techniques and Tools," Addison Welsey: Reading, MA.

Aldrich, J., (1997) "R.A. Fisher and the Making of Maximum Likelihood 1912-1922," vol(12), Statistical Science, pp. 162-176.

Allauzen, C., Mohri, M., and Roark, B., (2003) "Generalized Algorithms for Constructing Statistical Language Models," Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, pp. 40-47.

Altun, Y., Johnson, M., and Hofmann, T., (2003) "Investigating Loss Functions and Optimization Methods for Discriminative Learning of Label Sequences," Proceedings of the 2003 Conference on Empirical Methods in NLP, pp. 145-152.

Armijo, L., (1966) "Minimization of Functions Having Lipschitz-Continuous First Partial Derivatives," Pacific Journal of Mathematics, pp. 1-3.

Aydin, Z., Akgun, T., and Altunbasak, Y., (2005) "A Modified Stack Decoder for Protein Secondary Structure Prediction," IEEE International Conference on Acoustics, Speech and Signal Processing.

Azzopardi, L., Girolami, M., and van Rijsbergen, K., (2003) "Investigating the Relationship between Language Model Perplexity and IR Precision-Recall Measure," 26th Annual ACM Conference on Research and Development in Information Retrieval, SIGIR.

Bacchiani, M. and Ostendorf, M., (1999) "Joint Lexicon, Acoustic Unit Inventory and Model Design," vol(29), Speech Communication, pp. 99-114.

Bahl, L. R., Brown, P. F., de Souza, P. V., and Mercer, R. L., (1986) "Maximum Mutual Information Estimation of Hidden Markov Model Parameters for Speech Recognition," Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 49-52.

Bahl, L. R., Jelinek, F., and Mercer, R. L., (1983) "A Maximum Likelihood Approach to Continuous Speech Recognition," IEEE Transactions on Pattern Analysis, Machine Intelligence, pp. 179-190.

Baum, L. E., (1972) "An Equality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes," Inequalities, pp. 1-8.

Baum, L. E. and Eagon, J. A., (1967) "An Inequality with Applications to Statistical

Estimation for Probabilistic Functions of Markov Processes and to a Model for Ecology," Bulletin of the American Mathematical Society, pp. 360-363.

Bellegarda, J. R., (2004) "Statistical Language Model Adaptation: Review and Perspectives," vol(40), Speech Communication.

Bellman, R., (1952) "On the Theory of Dynamic Programming," Proceedings of National Academy of Science of the USA, pp. 716-719.

Beulen, K., Ortmanns, S., and Elting, C., (1999) "Dynamic Programming Search Techniques for Across-Word Modelling in Speech Recognition," Proceedings of International Conference on Acoustics, Speech, and Signal Processing, pp. 609-612.

Biem, A., (2006) "Minimum Classification Error Training for Online Handwriting Recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1041-1051.

Bishop, C. M., (1995) "Neural Networks for Pattern Recognition," Oxford University Press.

Blum, J. R., (1954) "Multidimensional Stochastic Approximation Methods," vol(25), Annals of Mathematical Statistics, pp. 737-744.

Caseiro, D. and Trancoso, I., (2002) "Using Dynamic WFST Composition for Recognizing Broadcast News," Proceedings of International Conference on Spoken Language Processing.

Chen, S. F., (2003) "Compiling Large-Context Phonetic Decision Trees into Finite-State Transducers," EUROSPEECH, pp. 1169-1172.

Chen, S. F., Beeferman, D., and Rosenfeld, R., (1998) "Evaluation Metrics For Language Models," DARPA Broadcast News Transcription and Understanding Workshop.

Chen, S. F. and Goodman, J., (1996) "An Empirical Study of Smoothing Techniques for Language Modeling," Proceedings of the 34th Annual Meeting of the ACL, pp. 310-318.

Chen, S. F. and Rosenfeld, R., (2000) "A Survey of Smoothing Techniques for ME Models," vol(8), IEEE Transactions on Speech and Audio Processing, pp. 37-50.

Chen, Z., Li, M. J., and Lee, K. F., (2000) "Discriminative Training on Language

Model," International Conference on Spoken Language Processing.

Chien, J. T., Huang, C. H., Shinoda, K., and Furui, S., (2006) "Towards Optimal Bayes Decision for Speech Recognition," vol(1), Proceedings of International Conference on Acoustics, Speech, and Signal Processing, pp. 45-48.

Chollet, G., Cochard, J. L., Langlais, Ph., and van Kommer, R., (1995) "Swiss-French Polyphone: a Telephone Speech Database to Develop Interactive Voice Servers," Linguistic Databases.

Chou, W., (2000) "Topics on Minimum Classification Error Rate Based Discriminant Function Approach to Speech Recognition," International Symposium on Chinese Spoken Language Processing.

Chow, Y. L., (1990) "Maximum Mutual Information Estimation of HMM Parameters for Continuous Speech Recognition Using the N-best Algorithm," vol(2), International Conference on Acoustics, Speech and Signal Processing, pp. 701-704.

Clarkson, P. and Rosenfeld, R., (1997) "Statistical Language Modeling Using The CMU-Cambridge Toolkit," Proceedings of EUROSPEECH, pp. 2707-2710.

Clarkson, P. R. and Robinson, A. J., (1997) "A Language Model Adaptation Using Mixtures and an Exponentially Decaying Cache," Proceedings of International Conference on Acoustics, Speech, and Signal Processing, pp. 799-802.

Collins, M., (2002) "Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms," Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1-8.

Collins, M. and Duffy, N., (2002) "New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron," Proceedings of ACL, pp. 263-270.

Collins, M. and Koo, T., (2000) "Discriminative Reranking for Natural Language Parsing," Proc. 17th International Conf. on Machine Learning, pp. 175-182.

Darken, C. and Moody, J., (1992) "Towards Faster Stochastic Gradient Search," vol(4), Neural Information Processing Systems, pp. 1009-1016.

Davis, H., Biddulph, R., and Balashek, S., (1952) "Automatic Recognition of Spoken Digits," vol(24), Journal of the Acoustical Society of America, pp. 637-642.

Digalakis, V., Monaco, P., and Murveit, H., (1996) "Genones: Generalized Mixture Tying in Continuous Hidden Markov Model-Based Speech Recognizers," IEEE Transactions Speech and Audio Processing, pp. 281-289.

Dijkstra, E. W., (1959) "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik, pp. 269-271.

Driessen, B. J., Sadegh, N., and Kwok, K. S., (1998) "A Robust Line Search for Learning Control," Proceedings of the 37th IEEE Conference on Decision and Control, pp. 3888-3892.

Duda, R. O. and Hart, P. E., (1973) "Bayes Decision Theory," Pattern Classification and Scene Analysis, pp. 10-43.

Finette, S., Bleier, A., Swindel, W., and Haber, K., (1983) "Breast Tissue Classification Using Diagnostic Ultrasound and Pattern Recognition Techniques: I. Methods of Pattern Recognition," vol(5), Ultrasonic Imaging, pp. 55-70.

Galliano, S., Geoffrois, E., Mostefa, D., Choukri, K., Bonastre, J. F., and Gravier, G., (2005) "The ESTER Phase II Evaluation Campaign for the Rich Transcription of French Broadcast News," Proceedings of INTERSPEECH, pp. 1149-1152.

Gao, J., Yu, H., Yuan, W., and Xu, P., (2005) "Minimum Sample Risk Methods for Language Modeling," HLT/EMNLP.

Gao, J. and Zhang, M., (2002) "Improving Langauge Model Size Reduction Using Better Pruning Criteria," ACL2002.

Good, I. J., (1953) "The Population Frequencies of Species and the Estimation of Population Parameters," vol(40), Biornetrika, pp. 237-264.

Goodman, J., (2000) "Putting It All Together: Language Model Combination," vol(3), Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 1647-1650.

Goodman, J., (2004) "Exponential Priors for Maximum Entropy Models," Proceedings of HLTNAACL, pp. 305-312.

Goodman, J. and Gao, J., (2000) "Language Model Size Reduction by Pruning and Clustering," Proceedings of International Conference on Spoken Language Processing.

Gopalakrishnan, P. S., Kanevsky, D., Nadas, A., and Nahamoo, D., (1991) "An

Inequality for Rational Functions with Applications to Some Statistical Estimation Problems," IEEE Transactions on Information Theory, pp. 107-113.

Gravier, G., Bonastre, J. F., Galliano, S., Geoffrois, E., Tait, K. M., and Choukri, K., (2004) "The ESTER evaluation campaign of Rich Transcription of French Broadcast News," Proceedings of Language Evaluation and Resources Conference.

Haeb-Umbach, R. and Ney, H., (1994) "Improvements in Beam Search for 10000-Word Continuous Speech Recognition," IEEE Transactions on Speech and Audio Processing, pp. 353-356.

Hart, P. E., Nilsson, N. J., and Raphael, B., (1968) "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transactions on Systems Science and Cybernetics SSC4, pp. 100-107.

Hoare, C. A. R., (1961) "ACM Algorithm 64: Quicksort," Communications of the ACM, p. 321.

Imai, T., Ando, A., and Miyasaka, E., (1995) "A New Method for Automatic Generation of Speaker-Dependent Phonological Rules," Proceedings of International Conference on Acoustics, Speech, and Signal Processing, pp. 864-867.

Iyer, R., Ostendorf, M., and Meteer, M., (1997) "Analyzing and Predicting Language Model Improvements," Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding.

Jelinek, F., (1969) "A Fast Sequential Decoding Algorithm Using a Stack," vol(13), IBM J. Res. Develop.

Jelinek, F., (1976) "Continuous Speech Recognition by Statistical Methods," vol(64), Proceedings of the IEEE, pp. 532-536.

Jelinek, F., (1997) "Statistical Methods for Speech Recognition," The MIT Press, Cambridge, Massachusetts.

Jelinek, F. and Mercer, R. L., (1980) "Interpolated Estimation of Markov Source Parameters from Sparse Data," Proceedings of the Workshop on Pattern Recognition in Practice, pp. 381-397.

Jelinek, F., Mercer, R. L., and Bahl, L. R., (1975) "Design of a Linguistic Statistical Decoder for the Recognition of Continuous Speech," vol(21), IEEE Transactions

on Information Theory, pp. 250-256.

Johansen, F. T., (1996) "A Comparison of Hybrid HMM Architectures Using Global Discriminative Training," Proceedings of the Fourth International Conference on Spoken Language Processing, pp. 498-501.

Juang, B. H., Chou, W., and Lee, C. H., (1997) "Minimum Classification Error Rate Methods for Speech Recognition," vol(5), IEEE Transactions on Speech and Audio Processing, pp. 257-265.

Juang, B. H. and Katagiri, S., (1992) "Discriminative Learning for Minimum Error Classification," vol(40), IEEE Transactions on Signal Processing, pp. 3043-3054.

Juang, B. H. and Rabiner, L. R., (2005) "Automatic Speech Recognition-A Brief History of the Technology Development," Elsevier Encyclopedia of Language and Linguistics, Second Edition.

Junqua, J. C. and Haton, J. P., (1995) "Robustness in Automatic Speech Recognition, Fundamentals and Applications," Kluwer Academic Publishers.

Kanthak, S., Ney, H., Riley, M., and Mohri, M., (2002) "A Comparison of Two LVR Search Optimization Techniques," Proceedings of International Conference on Spoken Language Processing.

Kanthak, S., Sixtus, A., Molau, S., Schluter, R., and Ney, H., (2000) "Fast Search for Large Vocabulary Speech Recognition," Verbmobil: Foundations of Speech-to-Speech Translation, W. Wahlster, Ed., pp. 63-78.

Katagiri, S., Lee, C. H., and Juang, B. H., (1990) "A Generalized Probabilistic Descent Method," Proceedings of the Acoustic Society of Japan, pp. 141-142.

Katz, S. M., (1987) "Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer," IEEE Transactions on Acoustics, Speech, and Signal Processing, pp. 400-401.

Kenny, P., Hollan, R., Boulianne, G., Garudadri, H., Lennig, M., and O'Shaughnessy, D., (1992) "An A* Algorithm for Very Large Vocabulary Continuous Speech Recognition," Proceedings of the workshop on Speech and Natural Language, pp. 333-338.

Kneser, R. and Ney, H., (1995) "Improved Backing-off for m-gram Language Modeling," Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 181-184.

Kuo, H. K. J., Fosler-Lussier, E., Jiang, H., and Lee, C. H., (2002) "Discriminative Training of Language Models for Speech Recognition," Proceedings of IEEE International Conference on Acoustics, Speech, Signal processing, pp. 325-328.

Kuo, H. K. J., Kingsbury, B., and Zweig, G., (2007) "Discriminative Training of Decoding Graphs for Large Vocabulary Continuous Speech Recognition," Proceedings of International Conference on Acoustics, Speech, and Signal Processing.

Lai, C. R., Lu, S. L., and Zhao, Q. W., (2002) "Performance Analysis of Speech Recognition Software," Proceedings of the Fifth Workshop on Computer Architecture Evaluation using Commercial Workloads.

Le Roux, J. and McDermott, E., (2005) "Optimization Methods for Discriminative Training," INTERSPEECH, pp. 3341-3344.

Lin, S. S. and Yvon, F., (2005) "Discriminative Training of Finite State Decoding Graphs," INTERSPEECH, pp. 733-736.

Lippmann, R. P., (1989) "Review of Neural Networks for Speech Recognition," vol(1), Neural Computation, pp. 1-38.

Lowerre, B., (1976) "The Harpy Speech Understanding System," Ph.D. thesis, Computer Science Department, Carnegie Mellon University.

Lowerre, B. and Reddy, R., (1990) "The Harpy Speech Understanding System," Readings in Speech Recognition, Morgan Kaufmann Publishers, San Mateo, CA, pp. 576-586.

Luenberger, D. G., (1989) "Linear and Nonlinear Programming," Second ed., Addison-Wesley, New York.

lyer, R. and Ostendorf, M., (1999) "Modeling Long Distance Dependence in Language: Topic Mixtures versus Dynamic Cache Models," IEEE Transactions on Speech and Audio Processing, pp. 30-39.

Magoulas, G. D., Vrahatis, M. N., and Androulakis, G. S., (1999) "Improving the Convergence of the Backpropagation Algorithm Using Learning Rate Adaptation Methods," vol(11), Neural Computation, pp. 1769-1796.

Mahajan, M., Beeferman, D., and Huang, X. D., (1999) "Improved Topic-Dependent Language Modeling Using Information Retrieval Techniques," Proceedings of International Conference on Acoustics, Speech, and Signal

Processing.

Maurice, G., (1987) "The Use of Finite Automata in the Lexical Representation of Natural Language," Lecture Notes in Computer Science, 377.

McDermott, E., Biem, A., Tenpaku, S., and Katagiri, S., (2000) "Discriminative Training for Large Vocabulary Telephone-based Name Recognition," Proceedings of International Conference on Acoustics, Speech and Signal Processing.

McDermott, E. and Hazen, T. J., (2004) "Minimum Classification Error Training of Landmark Models for Real-Time Continuous Speech Recognition," Proceedings of International Conference on Acoustics, Speech, and Signal Processing, pp. 937-940.

McDermott, E., Hazen, T. J., Le Roux, J., Nakamura, A., and Katagiri, S., (2007) "Discriminative Training for Large Vocabulary Speech Recognition Using Minimum Classification Error," IEEE Transactions on Audio, Speech and Language Processing, pp. 203-223.

McDermott, E. and Katagiri, S., (1994) "Prototype-Based Minimum Classification Error / Generalized Probabilistic Descent Training for Various Speech Units," vol(8), Computer Speech and Language, pp. 351-368.

McDermott, E. and Katagiri, S., (2005) "Minimum Classification Error for Large Scale Speech Recognition Tasks using Weighted Finite State Transducers," vol(1), IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 113-116.

Mohri, M., (1996) "On Some Applications of Finite-State Automata Theory to Natural Language Processing," Journal of Natural Language Engineering, pp. 61-80.

Mohri, M., (1997) "Finite-State Transducers in Language and Speech Processing," vol(23), Computational Linguistics, pp. 269-311.

Mohri, M., (2000) "Minimization Algorithms for Sequential Transducers," vol(234), Theoretical Computer Science, pp. 177-201.

Mohri, M., (2002) "Generic Epsilon-Removal and Input Epsilon-Normalization Algorithms for Weighted Transducers," International Journal of Foundations of Computer Science, pp. 129-143.

Mohri, M., Pereira, F., and Riley, M., (1996) "Weighted Automata in Text and

Speech Processing," Extended Finite State Models of Language: Proceedings of the ECAI Workshop, pp. 46-50.

Mohri, M., Pereira, F., and Riley, M., (2000a) "The Design Principles of a Weighted Finite-State Transducer Library," vol(231), Theoretical Computer Science, pp. 17-32.

Mohri, M., Pereira, F., and Riley, M., (2000b) "Weighted Finite-State Transducers in Speech Recognition," ISCA ITRW Automatic Speech Recognition: Challenges for the Millenium, pp. 97-106.

Mohri, M. and Riley, M., (1997) "Weighted Determinization and Minimization for Large Vocabulary Speech Recognition," EUROSPEECH, pp. 131-134.

Mohri, M. and Riley, M., (1998) "Network Optimizations for Large-Vocabulary Speech Recognition," vol(28), Speech Communication, pp. 1-12.

Mohri, M. and Riley, M., (1999) "Integrated Context-Dependent Networks in Very Large Vocabulary Speech Recognition," EUROSPEECH, pp. 811-814.

Mohri, M. and Riley, M., (2001) "A Weight Pushing Algorithm for Large Vocabulary Speech Recognition," Proceedings of EUROSPEECH, pp. 1603-1606.

Mor'e, J. and Thuente, D., (1994) "Line Search Algorithms with Guaranteed Sufficient Decrease," vol(20), ACM Transactions on Mathematical Software, pp. 286-307.

Nadas, A., Nahamoo, D., and Picheny, M., (1988) "On a Model-Robust Training Method for Speech Recognition," IEEE Transactions on Acoustics, Speech and Signal Processing, pp. 1432-1436.

Nekrylova, Z. V., (1975) "Rate of Convergence of the Stochastic Gradient Method," Cybernetics and Systems Analysis, pp. 218-222.

Ney, H. and Aubert, X., (1994) "A Word-Graph Algorithm for Large-Vocabulary Continuous-Speech Recognition," Proceedings of International Conference on Spoken Language Processing, pp. 1355-1358.

Ney, H. and Ortmanns, S., (1997) "Extensions to the Word Graph Method for Large Vocabulary Continuous Speech Recognition," International Conference on Acoustics, Speech, and Signal Processing, pp. 1791-1794.

Nilsson, N., (1997) "Artificial Intelligence: A New Synthesis," ISBN:

1-55860-535-5, MORGAN KAUFFMAN.

NIST, (2000) "NIST Spoken Language Technology Evaluation and Utility," http://www.nist.gov/speech/tools/index.htm.

Normandin, Y., (1991) "Hidden Markov Models, Maximum Mutual Information Estimation and the Speech Recognition Problem," Ph.D. thesis, McGill University.

Normandin, Y., (1995) "Optimal Splitting of HMM Gaussian Mixture Components with MMIE training," Proceedings of International Conference on Acoustics, Speech, and Signal Processing, pp. 449-452.

Och, F. J., (2003) "Minimum Error Rate Training in Statistical Machine Translation," Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, pp. 160-167.

Odell, J., Valtchev, V., Woodland, P., and Young, S., (1994) "A One Pass Decoder Design for Large Vocabulary Recognition," Proceedings ARPA Workshop on Human Language Technology, pp. 405-410.

Oerder, M. and Ney, H., (1993) "Word Graphs: An Efficient Interface between Continuous Speech Recognition and Language Understanding," vol(12), Proceedings of International Conference on Acoustics, Speech, and Signal Processing, pp. 119-122.

Ohler, U., Harbeck, S., and Niemann, H., (1999) "Discriminative Training of Language Model Classifiers," EUROSPEECH, pp. 1607-1610.

Ortmanns, S., Eiden, A., Ney, H., and Coenen, N., (1997a) "Look-Ahead Techniques for Fast Beam-Search," International Conference on Acoustics, Speech, and Signal Processing, pp. 1783-1786.

Ortmanns, S., Ney, H., and Aubert, X., (1997b) "A Word Graph Algorithm for Large Vocabulary Continuous Speech Recognition," vol(11), Computer, Speech and Language, pp. 43-72.

Ortmanns, S., Ney, H., and Eiden, A., (1996a) "Language-Model Look-Ahead for Large Vocabulary Speech Recognition," Proceedings of International Conference on Spoken Language Processing, Philadelphia, PA, pp. 2095-2098.

Ortmanns, S., Ney, H., Eiden, A., and Coenen, N., (1996b) "Look-Ahead Techniques for Improved Beam Search," Proceedings of CRIM-FORWISS Workshop, pp. 10-22.

Ortmanns, S., Ney, H., Seide, F., and Lindam, I., (1996c) "A Comparison of Time Conditioned and Word Conditioned Search Techniques for Large Vocabulary Speech Recognition," Proceedings of International Conference on Spoken Language Processing, pp. 2091-2094.

Paciorek, C. and Rosenfeld, R., (2000) "Minimum Classification Error Training in Exponential Language Models," Proceedings of the NIST/DARPA Speech Transcription Workshop.

Paul, D. B., (1992) "An Efficient A* Stack Decoder Algorithm for Continuous Speech Recognition with a Stochastic Language Model," Proceedings of International Conference on Acoustics, Speech, and Signal Processing, pp. 25-28.

Pereira, F., Riley, M., and Sproat, R., (1994) "Weighted Rational Transductions and Their Application to Human Language Processing," ARPA Workshop on Human Language Technology, pp. 249-254.

Pereira, F. C. N. and Riley, M., (1997) "Speech Recognition by Composition of Weighted Finite Automata," Finite-State Language Processing, MIT Press, pp. 431-453.

Pietra, S. D., Pietra, V. D., Mercer, R. L., and Roukos, S., (1992) "Adaptive Language Modeling Using Minimum Discriminant Estimation," Proceedings of International Conference on Acoustics, Speech, and Signal Processing, pp. 633-636.

Povey, D. and Woodland, P. C., (2002) "Minimum Phone Error and I-Smoothing for Improved Discriminative Training," Proceedings of International Conference on Acoustics, Speech, and Signal Processing, pp. 105-108.

Printz, H. and Olsen, P., (2000) "Theory and Practice of Acoustic Confusability," ASR 2000, pp. 77-84.

Rabiner, L. R., (1989) "A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," vol(77), Proceedings of the IEEE, pp. 257-286.

Roark, B., Saraclar, M., and Collins, M., (2004) "Corrective Language Modeling for Large Vocabulary ASR with the Perceptron Algorithm," Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing.

Roche, E. and Schabes, Y., (1997) "Finite-State Language Processing," MIT Press, Cambridge, MA..

Rosasco, L., De Vito, E., Caponnetto, A., Piana, M., and Verri, A., (2004) "Are Loss Functions All the Same?," Neural Computation, pp. 1063-1076.

Rosenblatt, F., (1958) "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," Psychological Review, pp. 386-408.

Rosenfeld, R., (2000) "Two decades of Statistical Language Modeling: Where Do We Go From Here?," Proceedings of the IEEE.

Sakti, S., Nakamura, S., and Markov, K., (2006) "Improving Acoustic Model Precision by Incorporating a Wide Phonetic Context Based on a Bayesian Framework," IEICE Transactions on ED, Special Section of Statistical Modeling for Speech Processing, pp. 946-953.

Sandness, E. and Hetherington, I., (2000) "Keyword-Based Discriminative Training of Acoustic Models," Proceedings of International Conference on Spoken Language Processing.

Sankar, A. and Lee, C. H., (1996) "A Maximum-Likelihood Approach to Stochastic Matching for Robust Speech Recognition," vol(4), IEEE Transactions on Speech and Audio Processing, pp. 190-202.

Saon, G., Povey, D., and Zweig, G., (2005) "Anatomy of an Extremely Fast LVCSR Decoder," INTERSPEECH, pp. 549-552.

Schluter, R. and Macherey, W., (1998) "Comparison of Discriminative Training Criteria," vol(1), Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 493-496.

Schluter, R., Macherey, W., Boris, M., and Ney, H., (2001) "Comparison of Discriminative Training Criteria and Optimization Methods for Speech Recognition," Speech Communication, pp. 287-310.

Schluter, R., Macherey, W., Kanthak, S., Ney, H., and Welling, L., (1997) "Comparison Of Optimization Methods For Discriminative Training Criteria," vol(1), EUROSPEECH, pp. 15-18.

Schultz, T. and Waibel, A., (1998) "Adaptation of Pronunciation Dictionaries for Recognition of Unseen Languages," Speech and Communication.

Shen, L., Sarkar, A., and Och, F. J., (2004) "Discriminative Reranking for Machine Translation," Proceedings of HLTNAACL.

Shimodaira, H., Rokui, J., and Nakai, M., (1998) "Improving the Generalization Performance of the MCE/GPD Learning," Proceedings of International Conference on Spoken Language Processing.

Singh, R., Raj, B., and Stern, R. M., (2002) "Automatic Generation of Sub-Word Units for Speech Recognition Systems," IEEE Transactions on Speech and Audio Processing, pp. 89-99.

Sloboda, T. and Waibel, A., (1996) "Dictionary Learning for Spontaneous Speech Recognition," Proceedings of International Conference on Spoken Language Processing.

Soong, F. K. and Huang, E. F., (1991) "A Tree-Trellis Based Fast Search for Finding the N-Best Sentence Hypotheses in Continuous Speech Recognition," ICASSP, pp. 705-708.

Spall, J. C., (2003) "Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control," ISBN 0-471-33052-3.

Steinbiss, V., Tran, B. H., and Ney, H., (1994) "Improvements in Beam Search," Proceedings of International Conference on Spoken Language Processing, pp. 2143-2146.

Stolcke, A., (2002) "An Extensible Language Modeling Toolkit," Proceedings of International Conference on Spoken Language Processing.

Viterbi, A. J., (1967) "Error bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm," vol(13), IEEE Transactions on Information Theory, pp. 260-269.

Warnke, V., Harbeck, S., Noth, E., Niemann, H., and Levit, M., (1999) "Discriminative Estimation of Interpolation Parameters for Language Model Classifiers," vol(1), International Conference on Acoustics, Speech and Signal Processing, pp. 525-528.

Widrow, B. and Hoff, M. E., (1960) "Adaptive Switching Circuits," IRE WESCON Convention Record Part IV, pp. 96-104.

Witten, I. H. and Bell, T. C., (1991) "The Zero Frequency Problem: Estimating the Probabilities of Novel Events in Adaptive Text Compression," vol(37), IEEE Transactions on Information Theory, pp. 1085-1094.

Woodland, P. C., Leggetter, C. J., Odell, J. J., Valtchev, V., and Young, S. J., (1995)

"The 1994 HTK Large Vocabulary Speech Recognition System," vol(1), Proceedings of International Conference on Acoustics, Speech, and Signal Processing, pp. 73-76.

Woodland, P. C. and Povey, D., (2002) "Large Scale Discriminative Training of Hidden Markov Models for Speech Recognition," Computer Speech and Language, pp. 25-47.

Ynoguti, C. A., Morais, E. S., and Violaro, F., (1998) "A Comparison between HMM and Hybrid ANN-HMM-based Systems for Continuous Speech Recognition," vol(1), Telecommunications Symposium, 1998. ITS '98 Proceedings. SBT/IEEE International, pp. 135-140.

Young, S. J., Evermann, G., Kershaw, D., Moore, G., Odell, J., Ollason, D., Valtchev, V., and Woodland, P., (2002) "The HTK Book," Cambridge University Engineering Department.

Young, S. J., Odell, J., and Woodland, P., (1994) "Tree-based State Tying for High Accuracy Acoustic Modelling," Proceedings ARPA Workshop on Human Language Technology, pp. 307-312.

Young, S. J., Russel, N. H., and Thornton, J. H. S., (1989) "Token Passing: a Simple Conceptual Model for Connected Speech Recognition Systems," Technical Report, Cambridge University.

Yu, X. H., Chen, G. A., and Cheng, S. X., (1995) "Dynamic Learning Rate Optimization of the Backpropagation Algorithm," IEEE Transaction on Neural Networks, pp. 669-677.

Zheng, J., Butzberger, J., Franco, H., and Stolcke, A., (2001) "Improved Maximum Mutual Information Estimation Training of Continuous Density HMMs," Proceedings of EUROSPEECH.

Zhou, D. X., (2002) "The Covering Number in Learning Theory," Journal of Complexity, pp. 739-767.

Zhu, X. and Rosenfeld, R., (2001) "Improving Trigram Language Modeling with the World Wide Web," Proceedings of International Conference on Acoustics, Speech and Signal Processing, pp. 592-597.