Confusion-based Statistical Language Modeling for Machine Translation and Speech Recognition

Final Report from 2011 CLSP Summer Workshop

Dan Bikel[f], Chris Callison-Burch[c], Yuan Cao[c], Nathan Glenn[d], Keith Hall[f], Eva Hasler[g], Damianos Karakos[c], Sanjeev Khudanpur[c], Philipp Koehn[g], Maider Lehr[b], Adam Lopez[c], Matt Post[c], Emily Prud'hommeaux[b], Brian Roark[b], Darcey Riley[h], Kenji Sagae[a], Murat Saraçlar[e], Izhak Shafran[b], Puyang Xu[c]

[a]USC, [b]OHSU, [c]JHU, [d]BYU, [e]Boğaziçi U., [f]Google, [g]Edinburgh, [h]Rochester

October 28, 2011

Contents

Chapter 1

Project Overview, Background and Acknowledgments

1.1 Project Overview

This project investigated semi-supervised methods for discriminative language modeling, whereby n-best lists are "hallucinated" for given reference (target language) text and are then used for training language models using the perceptron and related algorithms. We investigated these methods for both machine translation (MT) and automatic speech recognition (ASR) tasks in multiple languages (ASR) and language pairs (MT). We performed controlled experiments versus strong baselines for each task, comparing the results achieved from training on "hallucinated" lists with training with "real" n-best list output from the baseline systems, and report promising gains over baseline system performance for our new methods. For MT, we found that models trained on "hallucinated" lists typically achieve larger gains than those trained from fully supervised parallel corpora. For ASR, we found that methods based on extracting phrasal cohorts – similar to methods from machine translation for extracting phrase tables – yielded the largest gains of the various methods we explored for simulating n-best lists, with the resulting models achieving over half of the WER reduction obtained via the fully supervised methods.

Standard generative language modeling methods for ASR and MT involve counting n-grams from large corpora, normalizing them to produce multinomial models over a fixed vocabulary and smoothing (regularizing) the model to avoid assigning zero probabilities. Other than perhaps selecting training corpora to skew the model towards one domain or another, there is nothing in this modeling approach that is optimized for a particular task objective. Regardless of whether the application making use of the language model is ASR, MT or some other application, the same parameterizations will be derived from the corpora, despite the fact that the kinds of ambiguities that the model will be used to resolve are radically different in the applications. In ASR, alternative transcriptions will be acoustically confusable; in MT, acoustic confusability has nothing to do with the kinds of alternative translations that the language model will be scoring within the system.

There are language modeling methods that tune parameters for task-specific objectives, though these are typically fully supervised discriminative methods that require input speech or source language text together with transcriptions of the speech or target language translations of the source language text. Such training data is relatively sparse compared to the amount of monolingual text that is typically used to train generative language models. In this project, we examine methods that can be trained on just monolingual text, i.e., are not limited to costly parallel or transcribed data, but which are focused on resolving task-specific ambiguities by simulating confusion sets – likely confusable strings for given text. Once the confusion sets have been generated (in the absence of the input speech or source language strings), model parameters can be tuned to prefer reference sequences. Overall, this is similar to Contrastive Estimation [1], albeit with observed (reference) output sequences rather than input sequences.

The team that we put together for the summer workshop was large and diverse, and we collectively worked on many sub-topics related to this overall topic: both ASR and MT; general purpose software and algorithms; data and baseline system preparation; feature engineering; confusion set generation; machine

learning; and experimentation. This report is likewise large and diverse. Here we present a brief guide of what is to come in this report, along with the key team members involved in each sub-effort. Note that the names next to specific topics do not imply that there was no interaction between topics or that individuals worked only on these problems; rather to simply highlight primary area of focus of individuals.

- Chapter 1 presents the overall motivation for the project and a small amount of common background on learning and language modeling.
- Chapter 2 presents a **new software library**, destined for open-source release, written during the workshop by Dan Bikel and Keith Hall.
- Chapter 3 presents work on the **baseline MT systems**, including a detailed sanity-check comparison of three decoders – Moses, Joshua and cdec – and some improvements made during the course of the workshop, spearheaded by Philipp Koehn, Chris Callison-Burch, Adam Lopez and Matt Post.
- Chapters 4–6 present work on various issues of **feature engineering** for discriminative language modeling, including conversation-oriented features explored by Damianos Karakos; neural-net based features explored by Puyang Xu; and syntax-derived features explored by Darcey Riley.
- Chapter 7 presents experiments on supervised and semi-supervised **discriminative language modeling for MT**, including work using round-trip translation, phrase table pivot methods and minimum risk training methods, as explored by Eva Hasler, Yuan Cao, Emily Prud'hommeaux, Chris Callison-Burch and Phillip Koehn.
- Chapter 8 presents experiments on **confusion set generation in ASR** for discriminative language modeling, including low-level finite-state methods as well as word and phrase level cohort models, as explored by Kenji Sagae, Maider Lehr, Puyang Xu, Emily Prud'hommeaux, Nathan Glenn, Murat Saraçlar and Zak Shafran.
- Chapter 9 presents further experiments on confusion set generation, this time for **Turkish ASR**, which include morphologically-based confusion models, explored by Murat Saraçlar in collaboration with his students Arda Çelebi, Haşim Sak and Erinç Dikici, who were not at the workshop.

Work on this topic has continued beyond the end of the short 6-week workshop. Three conference papers have been submitted based on the work reported here, and more are in preparation. The work that was completed during this workshop has set the table for several strands of research that we hope to continue investigating in coming years.

In what follows, we give a (very) brief introduction to the perceptron and related algorithms, and how they are applied to discriminative language modeling, followed by acknowledgments and the report itself.

1.2 Background

1.2.1 Supervised discriminative language modeling

Discriminative language modeling has been used to optimize large vocabulary speech recognition performance, both with standard n-gram features [2, 3], as well as with morphological, syntactic and trigger features across a variety of languages and tasks [4, 5, 6, 7]. The paradigm in this approach is to run the baseline recognizer over training data and optimize a log linear model using some discriminative objective, such as global conditional log likelihood or with the perceptron algorithm. Because transcribed speech is required to train models with these methods, the amount of data that is available for such an approach is typically a small fraction of what is available to train generative language models, which only require text. The current project is focused on the problem of applying discriminative language modeling methods to training data that consists of just text. Should such methods prove successful, they could be applied to large scale text resources just as generative models are.

The perceptron algorithm was introduced by F. Rosenblatt in the 60s, for learning single-layer neural networks [8]. The basic idea is that, in the two-class separable case, one can learn an "optimum" feature weight vector simply by going through the training data and only updating the weight with those examples

Inputs: Training examples (x_i, y_i)
Algorithm: Set $\mathbf{w} = 0$
For $t = 1 \ldots T$
For $i = 1 \ldots N$
Calculate $z_i = \arg\max_{z \in \mathrm{GEN}(x_i)} \mathbf{w} \cdot \Phi(x_i, z)$
If $z_i \neq y_i$, then update $\mathbf{w} = \mathbf{w} + \eta(\Phi(x_i, z) - \Phi(x_i, z))$
Output: Weight vector $\mathbf{w}$

Figure 1.1: The perceptron algorithm for discriminative language modeling used in [3], following [9].

that get misclassified. In algorithmic terms, assuming that $\mathbf{w}(t)$ is the weight vector at time t, it is updated according to

$$\text{if } \mathbf{w}(t)\mathbf{x}_i y_i < 0, \quad \text{then} \quad \mathbf{w}(t+1) = \mathbf{w}(t) + \eta \mathbf{x}_i y_i,$$

where η is the *learning rate*, and y_i is the label $(+1/-1)$ of the training example $\mathbf{x}_i$.

In the context of discriminative language modeling for ASR, following the approach of [9], [3] converted the problem of re-ranking the output hypotheses (e.g., an n-best list) into a multi-class problem: at each iteration of perceptron, the n-best list is reranked according to the current perceptron weight vector, and the weight vector is updated if the top-ranked hypothesis is not the lowest-error hypothesis.

The perceptron algorithm for discriminative language modeling is shown in Figure 1.1. Here, x_i represents the i-th utterance, $\mathrm{GEN}(x_i)$ is the output space (set of possible candidates generated for x_i by the ASR system), and y_i is the minimum-WER (oracle) hypothesis in $\mathrm{GEN}(x_i)$. (Alternatively, y_i could be the manual transcription associated with x_i, but it turns out that using the oracle leads to a more stable algorithm.) Furthermore, $\Phi(x_i, z)$ is the set of features associated with hypothesis $z \in \mathrm{GEN}(x_i)$.

Variants of this basic algorithm can lead to models with improved sensitivity to specific task objectives, e.g., on-line passive-aggressive algorithms [10], which have been successfully used for discriminative modeling in MT [11, 12], albeit with significantly fewer features than are typically used for discriminative language modeling for ASR. Such methods typically vary z_i, y_i and η in the algorithm in Figure 1.1. Direct loss minimization [13] provides a very general framework for various methods of choosing z_i and y_i based on, for example, combinations of the loss and the model score.

Discriminative language modeling of the sort pursued in ASR, as discussed above, has not been pursued much for MT. Discriminative modeling for MT has typically been used with a much smaller feature set, as mentioned above. While discriminative language models for ASR have used millions or tens of millions of features [3], discriminative models in MT tend to focus on tens (MERT [14]) to maybe thousands (MIRA [11, 12]) of features. Within such an approach, various language model scores can be combined in ways that optimize system objectives, but n-gram parameters themselves are not included as features in the model. In this project, we examined methods for including larger scale models in the MT system.

For discriminative language modeling, we made use of a pre-existing codebase, written by Brian Roark, for training discriminative language models using the perceptron algorithm. This code had been used in prior studies on discriminative language modeling, including [15, 7, 16]. It was modified to provide support for the on-line perceptron like algorithms discussed above, and was used in empirical trials while the new software library (detailed in Chapter 2) was under development.

1.2.2 Confusion set generation

Simulating ASR errors has been pursued in the past, using a number of methods. Printz and Olsen [17] presented methods for calculating the acoustic confusability of words, for the purpose of estimating the word error rate (WER) of a model in a new domain. Chen et al. [18] provide some new methods in this vein. More recently, Tan et al. [19] pursued methods that exploited parallel corpora of ASR output and reference text to learn models of errors, using methods popularized for machine translation (MT). None of these methods have been directly applied to improving system performance.

Simulated errors have been used to improve MT systems [20, 21]. Exploiting simulated errors for ASR system improvements was explored in Jyothi and Fosler-Lussier [22] using weighted finite-state transducer approaches for generating sequences confusable with the reference. Kurata et al. [23] also used weighted

finite-state transducers, modeling acoustic similarity to generate confusable strings in a process they call "pseudo-ASR," which creates training data for a discriminative model that produced improved WER in Japanese call center data [24]. We will explore similar methods in this report, applied to large vocabulary continuous telephone speech systems, and contrast them with methods closer to the Tan et al. [19] methods cited above and the closely related methods of cohort extraction [25].

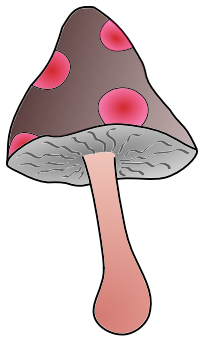
1.3 Acknowledgments

The work presented here was done as part of a 2011 CLSP summer workshop project at Johns Hopkins University. We acknowledge the support of the sponsors of that workshop. This work was also partially supported by NSF Grants #IIS-0963898 and #IIS-0964102, and by TUBITAK (Project No: 109E142). Murat Saraçlar was also supported by the TÜBA-GEBİP award. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsors.

Discussions with Mark Dredze about topic models are also gratefully acknowledged.

As team leader, Brian Roark would like to thank the rest of the team members for the terrific collective spirit of the project, which was one of good humor, individual responsibility, hard work, and responsiveness. **Everyone** on the team managed to make strong individual contributions, so that, collectively, this was a very productive project; but it was also a very fun project to work on and a fun group to be interacting with on a daily basis.

Thanks also to the other teams – led by Alan Black and Alex & Tamara Berg – for contributing to the overall collegiality of the event. Finally, thanks to the people at the Center for Language and Speech Processing at JHU – Monique Bates, Desiree Cleves, Justin Martin, and the faculty and students – for making CLSP such a nice place to work and hang out for the summer.



Part I

Software, Models, and Systems

Chapter 2

Distributed reranking software

2.1 Introduction

Creating effective software tools for research is a tricky business. The classic tension between flexibility and efficiency arises with greater urgency. We want researchers to be able to try out many different ideas easily, but we also want them to be able to have a quick code-test-evaluate cycle. In particular, one of the goals of generating confusions for discriminative language modeling was to scale up such models to training data sizes orders of magnitude larger than have been used previously. So, we not only need basic efficiency in our tools, but they need to be designed from the ground up with distributed computing in mind.

This chapter will describe the tools we have developed to solve not only our immediate research problem of exploring confusions for discriminative language modeling, but also the more general problem of reranking approaches to speech and language processing, including structured prediction. The desiderata for our tools were:

- "library quality" code
- industrial strength
- academic flexibility
- easy exploration of
  - different types of features
  - different update methods (*e.g.*, MIRA-style, direct loss minimization, loss-sensitive)
  - different learning methods (*e.g.*, perceptron-style, log-linear, kernel methods)
- modern, object-oriented design, complete with dynamic factories and dynamic composition for flexibility
- parallelizable, especially for distributed-computing environments

2.2 Data Format for I/O

Early on, our team discussed the relative merits of (a) rescoring lattices and other discriminative methods that were tightly coupled with the output of the baseline model versus (b) simply using a strict reranking approach applied to k-best lists. We settled on (b) reranking k-best lists. The primary reasons were the flexibility this would allow us in designing features, since k-best lists readily allow for sentence-level features in a way that, say, lattices do not, as well as the flexibility in designing tools, since it is far easier to define generic schemes of passing around k-best lists than it is for designing schemes to take speech lattices as well as machine translation hypergraphs or other, problem-specific data types.

2.2.1 Google Protocol Buffers

Supervised machine learning is based on the supply of labeled (supervised) examples from which a model is trained. A training algorithm is used to induce the model which best explains the supervised data, while at the same time being robust to data not seen in the supervised dataset (what is referred to as regularization of a model). The general paradigm is true for all supervised training algorithms and is also true in the discriminative training algorithms we use for our experiments in this project. Although not critical when building a small system which expects relatively little variation in the input data, the open-source toolkit we have developed is meant to be flexible enough to allow for a variety of data sources.

In order to avoid the need for overly complex data formats, we have chosen to adopt a formalism which allows one to augment the input format, allowing for flexible feature extraction and data manipulation/analysis. We opted to use a data format which mirrors the data-structures that are used internally for training. The Google protocol buffers[26] provide a programming-language independent specification framework to define data formats. The protocol buffers specification language is used by the protocol buffer tools to generate source-code for serializing and deserializing the data stored in the format. Code is generated to allow for native programming-language encapsulation of the data. For example, in C++ each item of data is stored in an object based on a object oriented data specification (a C++ *class*) allowing for access to the data.

As a by-product of the fact that this workshop targeted multiple tasks (ASR and SMT), we had to address incompatible tools and data formats, which led us to a universal data format. In order to transform data generated in proprietary formats, we developed tools to convert data to our protocol buffer format. These tools are delivered with the open source toolkit in:

```
./lm-confusion/dataconvert/
```

FST input (Roark's format)

```
./lm-confusion/dataconvert/asrnbestproto
```

SMT input (Köhn's format)

```
./lm-confusion/dataconvert/mtnbestproto
```

Each of these tools takes the predefined format as input and outputs a serialized set of protocol buffers. These are encoded with Base64 encoding and (optionally) compressed (Base64 encoding is needed as protocol-buffers are not self-delimiting). The disk-based format is able to be manually inspected using the *protoview* tool, found in the repository at:

```
./lm-confusion/dataconvert/mtnbestproto
```

2.2.2 Protocol Buffer Snippet

Following is a snippet of the protocol buffer specification for the training data input format.

```
message CandidateMessage {
  optional CandidateType type = 1;
  optional string rawdata = 2;
  optional FeatureVecMessage feats = 3;
  repeated ScoreMessage score = 4;
}
message CandidateSetMessage {
  optional string sourcekey = 1; // Unique key for this candidate set.
  optional string referencestring = 3; // Index in the original file.
  repeated CandidateMessage candidate = 4;
  optional int32 goldindex = 5;
  optional int32 bestscoringindex = 6;
}
```

This specification is automatically compiled into C++, java, or python source code to be used by the training and inference tools. The protocol buffer specifications is delivered with the toolkit (one for the model format and one for the data format).

```
./lm-confusion/proto/model.proto
./lm-confusion/proto/data.proto
```

2.3 Core learning framework

Consider Algorithm 1, which describes the training procedure for a generic perceptron-style algorithm. Each training example e_i comprises a set of *candidate hypotheses*, each of which is projected via some function Φ into a *feature space*, $\mathbb{R}^N$. We typically think of Φ as being a suite of *feature functions*, one per dimension. The model itself is defined as a weight vector in this space, w. Decoding, or inference, is carried out simply by taking the dot product of the model and a test instance. More generally, any kernel function K may be used. The training procedure iterates over the training data T—each iteration is called an *epoch*—until the NeedToKeepTraining() predicate returns `false`. Often, such a predicate is based on the average loss of the current model on some held-out development data D, which is the purpose of the Evaluate(D) line in the Train(T) procedure.

Algorithm 1 Training algorithm for perceptron-style reranking models.

Let $e_i = \{c_1, \ldots, c_k\}$ be a *training example*, where each c_j is a *candidate hypothesis*.
Similarly, let $d_i = \{c_1, \ldots, c_k\}$ be a held-out development data example, also consisting of k candidate hypotheses.
Finally, let K be a *kernel function*.

```
procedure Train(T = {e_1, ..., e_n}, D = {d_1, ..., d_m})
    while NeedToKeepTraining() do
        TrainOneEpoch(T)
        Evaluate(D)
    end while
end procedure

procedure TrainOneEpoch(T)
    foreach training example e_i do
        ScoreCandidates(e_i)
        if NeedToUpdate() then
            Update()
        end if
    end for
end procedure

procedure ScoreCandidates(e_i)
    foreach candidate hypothesis c_j ∈ e_i do
        c_j.score ← K(w_t, c_j)
    end for
end procedure
```

For the basic perceptron, the model starts out at time step 0 as the zero vector; that is, $w_o = \vec{0}$. The update is

$$w_{t+1} = w_t + R_t \left[\Phi\left(y_{\text{oracle}}\left(e_i\right)\right) - \Phi\left(\hat{y}\left(e_i\right)\right) \right], \tag{2.1}$$

where y_{oracle} is a function that picks out the hypothesis towards which we want to bias our model, $\hat{y}$ is a function that picks out the candidate hypothesis we want to bias our model against and R_t is a *learning rate* or *step size*. Most often, y_{oracle} is defined to pick the hypothesis with the lowest loss relative to some gold-standard truth, and $\hat{y}$ is defined to pick the candidate hypothesis that scores highest under the current model

```
class Model {
 public:
  virtual bool NeedToKeepTraining() = 0;
  virtual void TrainOneEpoch(...) = 0;
  virtual bool NeedToUpdate() = 0;
  virtual void Update() = 0;
  // ... more methods ...
};
```

Figure 2.1: A first stab at defining a `Model` interface in C++.

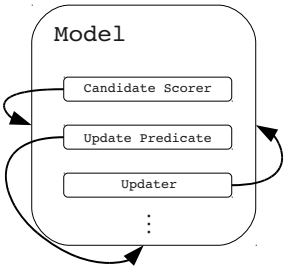


Figure 2.2: A pictorial view of how a `Model` wraps instances of other interfaces that specify the predicates and functions needed to carry out model training.

w_t. Under this scheme, we would technically not need a separate NeedToUpdate() predicate, because when $y_{\text{oracle}}(e_i) = \hat{y}(e_i)$, the basic udpate shown in Equation 2.1 is a "no op". In general, however, we would like the flexibility of defining an arbitrary predicate to determine whether an update is required.

2.3.1 Flexibility without sacrifice

Most of the variations of this basic learning method involve finding different ways of defining R_t, Φ, y_{oracle} and $\hat{y}$, along with the various procedures and predicates shown in Algorithm 1. Therefore, we would like our Reranker Framework to make it easy for the researcher to define these various functions, as well as to specify which ones to use at run-time.

An easy way to design a *flexible* system is to have a basic interface—let's call it `Model`—that specifies a *virtual function* for each operation that a researcher might want to experiment with. For example, since one might want to experiment with different definitions for the NeedToUpdate() method, one could simply add it to `Model`. In C++, this would look something like Figure 2.1.

This works well for flexibility, but consider the case where the researcher wants to experiment with the various, largely orthogonal definitions of NeedToUpdate(), NeedToKeepTraining() and all the other functions. The way one would do this in any object-oriented setting would be to define a concrete implementation of the `Model` interface for each instantiation of the cross product of possible variants one wanted to explore. Such an approach quickly becomes unwieldy.

A better approach is to use *dynamic composition.* That is, we keep the idea of a `Model` interface, but additionally have each `Model` instance wrap a set of predicate/manipulator objects, each of which itself conforms to an interface. Figure 2.2 shows a pictorial representation of this scheme. Crucially, a `Model` instance HAS-A `Updater` instance, and the `Updater` interface specifies to use the `Model` interface, and similarly for other objects with which a `Model` is composed. This mutually recursive relationship is what gives this scheme its great flexibility without the complexity associated with creating impossibly many implementations of the `Model` interface. The outline of this approach in C++ is shown in Figure 2.3.

```
class UpdatePredicate {
 public:
  virtual bool NeedToUpdate(const Model *model) = 0;
};

class Updater {
 public:
  virtual void Update(Model *model) = 0;
};

class CandidateScorer {
 public:
  virtual void Score(CandidateSet &candidate_set) = 0;
};
// ... more interfaces to be composed with Model

class Model {
 public:
  virtual bool NeedToKeepTraining() = 0;
  virtual void TrainOneEpoch(...) = 0;
  virtual bool NeedToUpdate() = 0;
  virtual void Update() = 0;
  // ... more methods ...
 private:
  // data members
  Updater *updater_;
  UpdatePredicate *update_predicate_;
  CandidateScorer *candidate_scorer_;
  // ... more objects composed with Model
};
```

Figure 2.3: The `Model` interface using dynamic composition with instances of other interfaces.

```
template <typename T>
class Factory {
 public:
  T *Create(const string &type);
};

// elsewhere, perhaps in our main method
Factory<Model> model_factory;
Model *perceptron_model =
  model_factory.Create("PerceptronModel");
//...
```

Figure 2.4: Dynamic factories in C++.

2.3.2 Dynamic object instantiation

As we discussed above, we employ dynamic composition to avoid defining a new subclass of `Model` every time we wish to explore a new combination of learning method functions. But how do we actually instantiate a specific, concrete `Model` implementation with concrete implementations of the objects with which it must be composed? We can certainly write a line—several lines—of code to do this, but if we do that, we are not much better off than if we had to define a subclass of `Model` for every combination we wanted to explore. Moreover, forcing the user to instantiate objects statically does not even begin to address the issue of feature function exploration. When we explore different features—different ways of defining the N feature functions that constitute Φ—we would ideally like to specify which ones we wish to employ at run-time.

The basic solution is to use a dynamic factory: a special class with a method that takes an object that is easily-specified at run-time, such as a `string`, as a parameter and returns a concrete instance of an abstract type, where the concrete type is determined by that input parameter. For example, suppose we have our abstract base type `Model` with a concrete implementation called `PerceptronModel`. In C++, we would like to be able to have something like what is shown in Figure 2.4. Having such `Factory` instances for dynamic object instantiation goes a long way toward making the system easy and flexible, but we would ideally like to specify how to construct entire complex objects, such as the `Model` that contains concrete instances of other interfaces.

2.3.2.1 Dynamic instantiation of complex objects

Given that C++ is a statically compiled language, how might we specify the construction of a complex object at run-time? The answer is that we use a simple, interpreted language. To ease the burden on the programmer, this language is very similar to the syntax used in the initialization phase of C++ constructors. To motivate our choice of syntax for this interpreted language, let's look at a simple C++ class called `Person`, shown in Figure 2.5.

As you can see, the `Person` class has three data members, one of which happens to be an instance of another class called `Date`. In this case, all of the initialization of a `Person` happens in the initialization phase of the constructor—the part after the colon but before the declaration phase block. By convention, each parameter to the constructor has a name nearly identical to the data member that will be initialized from it. If we wanted to construct a `Person` instance for someone named "Fred" who was 180 cm tall and was born January 10th, 1990, we could write the following:

```
Person fred("Fred", 180, Date(1990, 1, 10));
```

If `Person` were a `Factory`-constructible type in the Reranker Framework, we would be able to specify the following as a specification string to tell the `Factory` how to construct a `Person` instance for Fred:

```
Person(name("Fred"), cm_height(180),
       birthday(Date(year(1990), month(1), day(10))))
```

```
// A class to represent a date in the standard Gregorian calendar.
 class Date {
  public:
    Date(int year, int month, int day) :
      year_(year), month_(month), day_(day) { }
  private:
    int year_;
    int month_;
    int day_;
 };

 // A class to represent a few facts about a person.
 class Person {
  public:
    Person(const string &name, int cm_height, const Date &birthday) :
      name_(name), cm_height_(cm_height), birthday_(birthday) { }
  private:
   string name_;
   int cm_height_;
   Date birthday_;
 };
```

Figure 2.5: A simple C++ example to motivate our interpreted language syntax.

As you can see, the syntax is very similar to that of C++. It is, to a first approximation, a combination of the parameter list and the initialization phase of a C++ constructor.

2.3.2.2 Interpreted language details

Unfortunately, we cannot get this kind of dynamic instantiation in C++ for free; we need some help from the programmer. However, we have tried to make the burden on the programmer fairly low, using just a handful of macros to help declare a `Factory` for an abstract base class, as well as to make it easy to make that `Factory` aware of the concrete subtypes of that base class that it can construct. Finally, each `Factory`-constructible abstract type needs to implement a single method called `RegisterInitializers` to specify the names of its data members that this language can initialize. The formal, BNF grammar for our language is shown in Figure 2.6.

2.3.2.3 Putting it all together

The Reranker Framework defines a common base class for all feature extractors called `FeatureExtractor`. Now, `FeatureExtractor` instances are `Factory`-constructible, and so the `FeatureExtractor` class ensures its concrete subclasses have a `RegisterInitializers` method. One of the concrete implementations of `FeatureExtractor` provided with the Reranker Framework is `NgramFeatureExtractor`, which extracts n-gram features from a training or test instance's text on the fly.[1] That class has two data members that can be initialized by a factory, one required and one optional. To show you how easy it is to "declare" data members that need initialization, here is the exact code from the `NgramFeatureExtractor::RegisterInitializers` method:

```
virtual void RegisterInitializers(Initializers &initializers) {
  bool required = true;
  initializers.Add("n", &n_, required);
  initializers.Add("prefix", &prefix_);
}
```

[1]This is one of the pieces of the Reranker Framework that is specialized for discriminative language modeling.

```
<spec>              ::=  <type> '(' <member_init_list> ')'
<type>              ::=  name of type constructible by this factory
<member_init_list>  ::=  <member_init> [ ',' <member_init> ]* [',']
<member_init>       ::=  <primitive_init> | <factory_init>
<primitive_init>    ::=  <member_name> '(' <literal> ')'
<member_name>       ::=  the name of the member to be initialized, as specified by
                         <type>'s RegisterInitializers method
<literal>           ::=  <string_literal> | <double_literal> |
                         <int_literal> | <bool_literal>
<string_literal>    ::=  a C++ string literal (a string of characters surrounded by
                         double quotes); double quotes and backslashes may be
                         escaped inside a string literal with a backslash; other
                         escape sequences, such as \t for the tab character, are
                         currently not recognized
<double_literal>    ::=  a string that can be parsed by atof
<int_literal>       ::=  a string that can be parsed by atoi
<bool_literal>      ::=  <string_literal> | <double_literal>
<factory_init>      ::=  <member_name> '(' <spec> ')'
```

Figure 2.6: Grammar for interpreted language for dynamic object instantiation in the Reranker Framework.

The above code says that the `NgramFeatureExtractor` has a data member `n_`, which happens to be an `int`, that is required to be initialized when an `NgramFeatureExtractor` instance is constructed by a `Factory`, and that the name of this variable will be `"n"` as far as the factory is concerned. It also says that it has a data member `prefix_`, which happens to be of type `string`, whose factory name will be `"prefix"`, and that is not required to be present in a specification string for an `NgramFeatureExtractor`.

Given the grammar described earlier, the following are all legal specification strings for constructing `NgramFeatureExtractor` instances:

```
NgramFeatureExtractor(n(3))
NgramFeatureExtractor(n(2), prefix("foo:"))
NgramFeatureExtractor(prefix("bar"), n(4))
NgramFeatureExtractor(n(2),)
```

As you can see, the order of member initializers is not important (because each has a unique name), and you can optionally put a comma after the last initializer. The following are illegal specification strings for `NgramFeatureExtractor` instances:

```
// Illegal specification strings:
NgramFeatureExtractor(prefix("foo"))
NgramFeatureExtractor()
NgramFeatureExtractor(n(3), prefix(4))
```

In the first two cases, the specification strings are missing the required variable n, and in the final case, the optional prefix member is being initialized, but with an int literal instead of a string literal.

As a more complex example, let us consider the abstract base class `Model`, which is also `Factory`-constructible. Each `Model` uses an instance of a comparator interface to determine the "gold" and top-scoring candidates when training. This is the primary means by which the Reranker Framework allows the user to specify a way of picking $\hat{y}$ in Equation 2.1. The specification strings for constructing `Model` instances reveal how recursion is used for constructing complex objects in this interpreted language. For example, to construct a `PerceptronModel` instance with a `DirectLossScoreComparator` , one would use the following specification string:

```
PerceptronModel(name("MyPerceptronModel"), score_comparator(DirectLossScoreComparator()))
```

The first member initialization, for the member called `name`, specifies the unique name you can give to each `Model` instance (which is strictly for human consumption). The second member initialization, for the member called `score_comparator`, overrides the default comparator used to compare candidate hypothesis scores. This second member is also an example of one `Factory`-constructible object being constructed as a data member of another `Factory`-constructible object.

2.3.3 Feature extraction

Another area that should be maximally flexible and yet easy to use is the specification of features, that is, the Φ of Equation 2.1. If one already has features encoded in either Roark's or Köhn's format, one may use the tools described in §2.2.1. That is one of the ways in which one can incorporate pre-computed features. Often, it is preferable to define many feature functions and choose among them on the fly. The Reranker Framework makes this easy, once again using the interpreted language for specifying which feature extractors to use, as demonstrated by the `NgramFeatureExtractor` example of the previous section. For even more flexibility, one might want to combine features that are extracted on the fly with different sets of features that have been precomputed but are sitting in separate files on disk. This scenario, too, is made easy by the Reranker Framework, which provides specialized `FeatureExtractor` implementations that can "extract" precomputed features from files.[2] The class `ExecutiveFeatureExtractor` handles the run-time construction and initialization of a suite of `FeatureExtractor` instances to be run on each problem instance at run-time. It uses an internal `Factory` instance to accomplish this.

2.3.4 Training

The traditional way to train a model for classification, reranking or structured prediction is to read the training data into memory and then iterate over it until training is complete. This way of training tends to minimize the time spent doing I/O and maximize the efficiency of training, but at the expense of memory. As Algorithm 1 shows, the basic perceptron algorithm involves "on-line" updating, and thus it is possible to read in each training example from file each time it is needed, only keeping the model's parameters persistently in memory. The Reranker Framework allows both the memory-intensive way of training as well as this "streaming mode" version of training. Crucially, training in "streaming mode" is essential to providing the distributed training framework, described in the next section.

2.4 Cluster-based distributed training

Learning techniques which are able to exploit unlabeled examples for training models often fall under the name semi-supervised learning; however, there are additional techniques which make use of unlabeled data (e.g., distance supervision[27]). Inducing training data from unlabeled text for discriminative modeling (the goal of this workshop) is another technique which can result in a relatively large set of training data (depending on the technique, this set is unbounded). Addressing the issue of huge sets of training data, we have developed a distributed training paradigm which can be executed on a cluster of compute nodes via the Hadoop MapReduce training framework.

The most direct approach to distributed, supervised learning is to split the training data into smaller subsets and process each of these subsets on a separate computing node (either multiple nodes on a single machine or across multiple machines in a cluster computing environment). The differences between distributed learning (and distributed optimization) algorithms arise at this point; the primary questions being, "what do we do on with each subset and how do we train a global model." One of the solutions is to simply distribute the data, train multiple independent models and then average their parameters; a technique known as parameter mixtures. Another technique is to use a batch training technique which collects statistics from each of the subsets, make an update to a global model and sends that model back to the clusters for the next iteration (epoch) of training. This is applied in the standard distributed gradient computation techniques.

The structured perceptron [9] and it's variants have proven to be effective in supervised, discriminative language modeling work [3]. We have centered the development of our open-source discriminative learning

[2] The class `AbstractFileBackedFeatureExtractor` is the base class for such feature extractors.

toolkit around perceptron-style algorithms, which are, by definition, on-line learning algorithms. Identifying the optimal solution for a distributed on-line optimization algorithm is still an open research question. We borrow from our previous work on distributed perceptron training in [28] and use the *Iterative Parameter Mixtures* algorithm for distributed computation.

2.4.1 Iterative Parameter Mixtures

Algorithm 2 Iterative Parameter Mixtures Perceptron

▷ Break S into K partitions
$S = \{D^1, \ldots, D^j, \ldots, D^K\}$ ▷ e.g., $D^1 = (x_1^1, y_1^1) \ldots (x_{S/K}^1, y_{S/K}^1)$
$t = 0$
repeat
  $t = t + 1$
  for $j = 1 \ldots K$ **do** ▷ Process in parallel
    $\theta_t^j = \theta_{t-1}$
    for $i = 1 \ldots |D^j|$ **do**
      if $\hat{y}_i \neq y_i$ **then**
        ▷ Update model Parameters
        $\theta_t^j = \theta_t^j + \Phi(y_i) - \Phi(\hat{y}_i)$
      end if
    end for
    $\Delta^j = \theta_t^j - \theta_{t-1}$
  end for
  for $l = 1 \ldots |w|$ **do** ▷ For each feature in the model (can be done in parallel)
    $\theta_t(l) = \theta_{t-1}(l) + \frac{1}{K} \sum_j \Delta^j(l)$
  end for
until converged

The iterative parameter mixtures (IPM) perceptron trainer is a distributed approximation to the standard perceptron algorithm. The general idea is to perform many partial optimizations of subsets of the training data followed by a synchronization of the models (partially) trained from these subsets. This process repeats for each epoch of training.

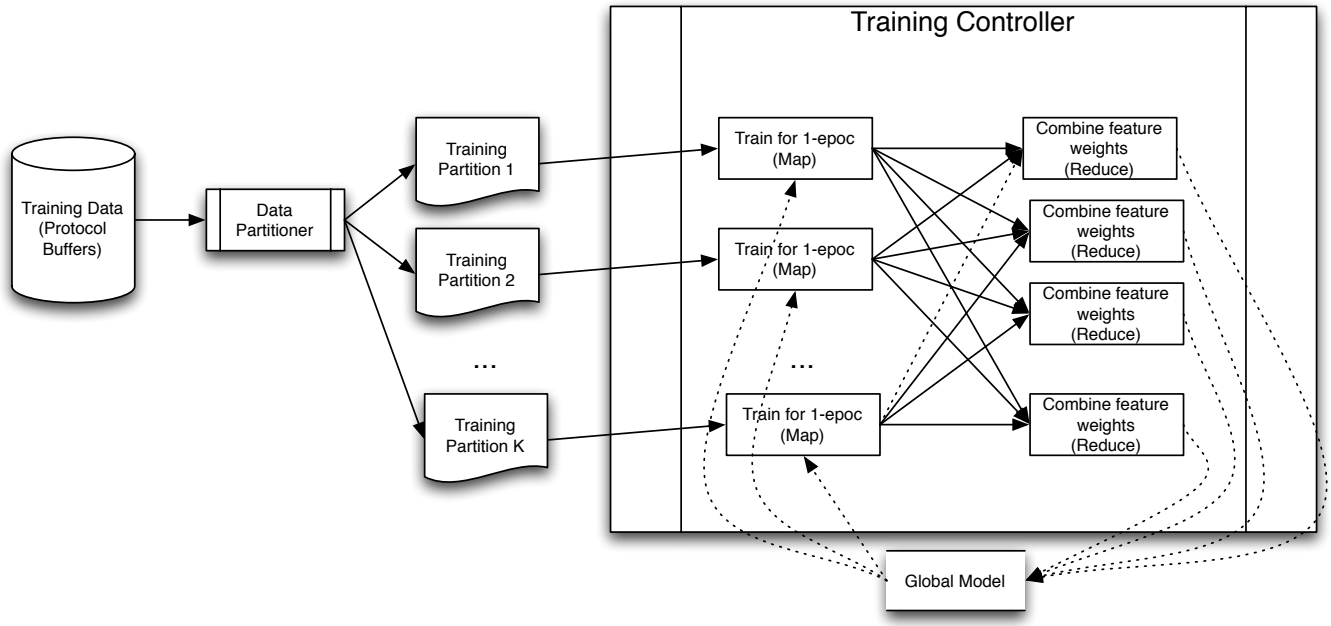
Algorithm 2 shows the IPM perceptron algorithms as presented in [28]. We split the data into K partitions: D^1 through D^K. Each of these partitions is processed independently on separate machines. During each iteration of training, each of these machines makes one pass through their partition of data, performing the standard structured perceptron up-date to their internal model. After all partitions have been processed, each machine sends it's version of the global model to another set of machines responsible for averaging the models. In order to do this later merging step, we partition each of the models based on their feature ids (e.g., each of these *merging* machines merges a subset of the feature space). Each machine will start its next iteration of training with the globally merged model.

There are some subtleties in the IPM algorithm which we have left out in this presentation (e.g., details of the parameter averaging). We defer to the original publication for full details on these topics. The implementation we are providing with the open source toolkit uses an error-weighted average (the contribution of each partitions average is weighted by the number of errors made over that partition in the most recent training epoch). This has the added benefit of being no slower to converge (in number of epochs over the training data) than that standard perceptron algorithm.

2.4.2 Hadoop Implementation

The iterative parameter mixtures algorithm described above is packaged as part of the open source toolkit an is primarily coded in python. Distributed computation is handled by way of using the Hadoop MapReduce computing framework. MapReduce is a general paradigm for distributed computation [29] where computation is broken into two general phases: mapping and reducing. Mapping is the process of taking an input dataset,

splitting it into partitions, and extracting key/value pairs from that data. Reducing allows us to take those key/value pairs and process them (where all values for a specified key will be in the same reduction partition).



The MapReduce framework is well suited for the iterative parameter mixtures learning algorithm. Parallel computation for each training-data partition is done in a mapping stage, where the model parameter mixing is done via a reducing stage. Figure 2.4.2 provides a general sketch of the mapreduce framework for the iterative parameters mixture algorithm we provide as part of the open source toolkit.

We have designed the training library to interact naturally with the MapReduce framework, providing a mechanism for encoding protocol buffers into file-formats that can be partitioned by the Hadoop MapReduce system.

Chapter 3

Comparison of Models and Search for Tree-Based MT Systems

The participants of the workshop included most of the principals behind three popular open source machine translation decoders for hierarchical and syntax-based models, namely

- Moses [30], at workshop: Koehn
- Joshua [31, 32, 33], at workshop: Callison-Burch and Post
- cdec [34], at workshop: Dyer and Lopez

We took advantage of this opportunity to compare the implementations. Our goal was to make the decoders compatible, i.e., being able to process the same models, gain insights from differences, spot bugs and implement improvements for each.

3.1 Comparison of decoders

3.1.1 Models

Moses and Joshua include grammar extraction programs that learn models from parallel corpora [35, 33]. We first compared these grammar extraction methods.

The main trade-offs between hierarchical and syntax-based models is between robustness and linguistic sensitivity. Syntax-based models promise to learn general patterns how nouns, verbs, noun phrases, clauses, etc. are processed during translation, but the introduction of the constrain that each non-terminal maps to a syntactic category severely limits what grammar rules can be extracted from a corpus. We may lose many a useful rule that runs contrary to linguistic constraints. Note also that we operate in a error-prone environment where automatically generated word alignments and syntactic parses are noisy.

To remedy this, a number of parse relaxation operations have been proposed. By binarizing the trees [36] more syntactic categories are introduced and hence more rules can be extracted. An extreme approach is SAMT [37], where labels for any span are introduce by pairing or otherwise grouping existing syntactic categories.

Table 3.1 shows results for Urdu–English for different grammar models. The results show that for this language pair, SCFG-based models outperform the phrase-based baseline, with the best result for a hierarchical model. A basic target syntax model performs worse, but various grammar relaxation schemes close the gap: left-binarization, right-binarization, and a reduced SAMT model that only allows for pairing of sibling non-terminals. The full-on SAMT grammar extractor Thrax [33] yields a model with much lower performance, matching the phrase-based model.

Model	Extractor	Test
Phrase-Based	Moses	21.3 (1.01)
Hierarchical	Moses	22.6 (1.01)
Target-Syntax	Moses	22.1 (1.00)
+ left-binarized	Moses	22.5 (1.00)
+ right-binarized	Moses	22.4 (1.00)
SAMT (reduced)	Moses	22.4 (1.00)
SAMT (full)	Thrax (Joshua)	21.3 (1.00)

Table 3.1: Performance of different grammar models, extracted by Moses and Joshua tools, on Urdu–English. Scores are reported as BLEU (length ratio).

3.1.2 Search

All of the decoders are capable of processing with any synchronous context-free grammar model, which may or may not use linguistic labels on the target side (the former is often called a target-syntax model, the latter a hierarchical model). During the workshop, we wrote conversion scripts that map between the model format used by Moses and the one used by Joshua and cdec. This enabled us to run the same model through any of the decoders.

We found some minor differences how the decoders score hypotheses with the models:

- Moses counts the beginning-of-sentence and end-of-sentence token as words in the computation of the word count feature. Since this is a constant addition to the score of any translation, it has no impact on search.
- Joshua uses the same weights for scoring glue rules and regular rules. This can be addressed by having different features in a Joshua rule table (when converting from Moses to Joshua), or duplicating the weights (from Joshua to Moses).
- At the time of the comparison there seemed to be a bug in the way Joshua scores glue rules.

Apart from these differences, the decoders produce the same model scores for the same derivations, thus allowing for direct comparison of search quality.

The three decoders employ a chart decoding algorithm, supported by a efficient prefix data structure for rules, and cube pruning for only generating the most promising hypotheses [38, 39]. One main difference lies in the division between the syntactic part of decoding and the language model integration.

Moses integrates syntactic parsing and language model integration: for each span, first applicable rules are looked up, and then the most promising hypothesis are generated. There is currently no distinction between target non-terminal types, thus for many non-terminals no hypotheses are created, even if rules exist.

In contrast, cdec first applies exhaustive syntactic parsing, essentially looking up all possible rules based on all possible syntactic categories for each span, and then integrates the language model, creating fully formed hypotheses.

In Table 3.2, we compare the performance of the three decoders when using the same grammar: the Urdu–English Thrax-SAMT grammar mentioned above, but tuned with Joshua. We note that this is a very rough comparison, since the different decoders use different beam search settings (pop-limit limits, span sizes, etc.). Also, cdec is mainly written for hierarchical models and had much difficulty in processing SAMT grammar with a large number of non-terminal labels.

The comparison of the different search implementations is ongoing work from which we expect assistance in finding bugs in the decoders and gain insight from different search algorithms. The main contribution of the work at the workshop is to get to the point of being able to run all three decoders on the same models.

Since the search graph of the chart decoder is a rather complex data structure that is hard to inspect, we worked on a visualization tool that allows us to interactively explore the behavior of the algorithms in practice. The tool is implemented in HTML5 and JavaScript, utilizing scalable vector graphics (SVG). It

Decoder	Test
Moses	21.0 (1.01)
Joshua	20.7 (1.01)
cdec	13.1 (1.09)

Table 3.2: Performance of the three decoders on the same Urdu–English SAMT grammar.

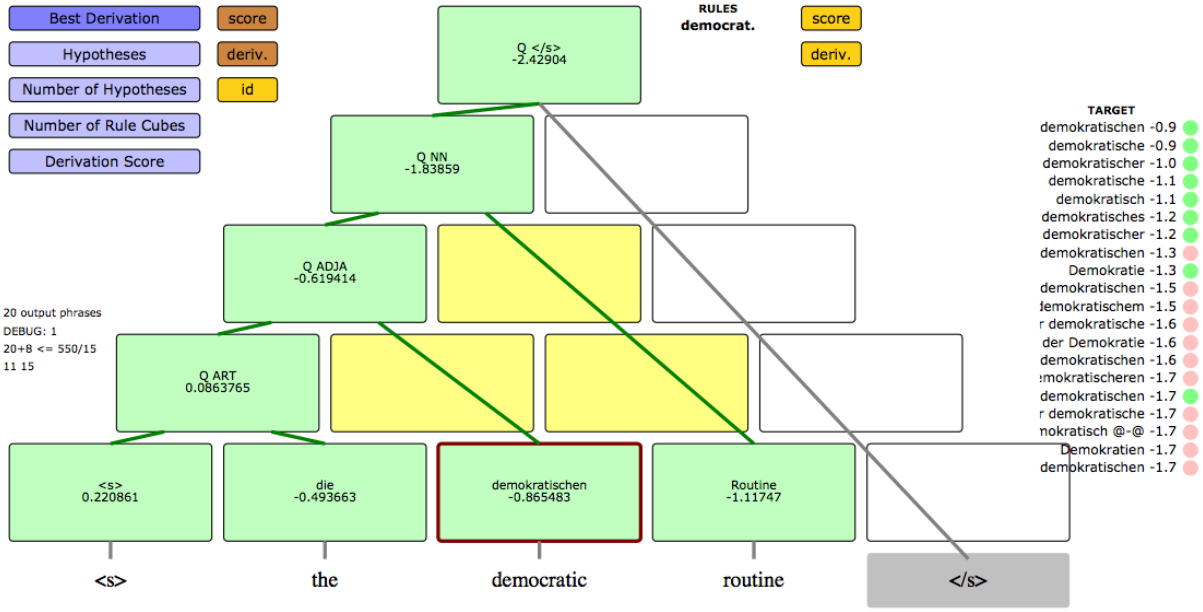


Figure 3.1: Search visualization for chart decoder

is integrated into the experimental management system (EMS) that ships with Moses. See Figure 3.1 for a screenshot.

We expect that tools like this will aid us to come to a better understanding of search for synchronous context-free grammar models.

3.2 Improvements to Baseline Feature Training

The workshop's main focus was to train features over the entire training set. However, traditionally, features are optimized over a small tuning set of a few thousand sentences. This is a far more manageable task; it requires much less computational effort. The main drawback of this traditional *tuning* [14] is that it only works reliable for features that are sufficiently frequent. Features have to occur a few times in the translation of a few thousand sentences. This eliminates, for instance, most lexicalized features.

However, due to the contributions of the workshop, we are now able to see a continuum of feature training methods (see also Figure 3.2):

- Large-scale feature training of millions of features over the entire training corpus
- Mid-scale feature training of thousands of features over large tuning sets (say, 5,000-10,000 sentences)
- Core feature training of a handful of features over traditional tuning sets (say, 1,000-2,000 sentences)

Furthermore, these methods are not mutually exclusive. Very sparse features that are trained over the entire training corpus (say, target bigram features) may then be consolidated into a single feature (the target bigram feature score) which may be re-weighted by traditional tuning. Ideally, we want to train as many

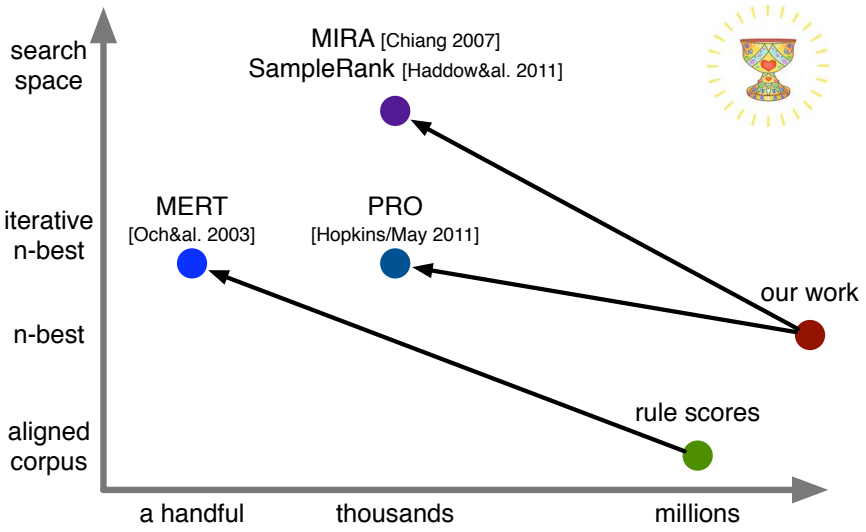


Figure 3.2: **Range of training methods for features:** Larger feature sets trained over large parallel corpora are computationally more expensive. Thus, less complex learning methods (re-ranking over n-best lists) are used.

features as possible over the largest parallel corpora available with the most sensitive methods (see the holy grail in Figure 3.2), but in practice it may be more efficient and reliable to use less expensive methods.

Hence, we also worked in this workshop on improving training methods for core features and mid-scale features in the traditional tuning setting.

3.2.1 Features

We added two different types of features to the Moses decoder. First, we added two features motivated by similar features in the Joshua/Thrax grammar extractor: a feature that indicates phrase pairs (i.e., translation rules) that have low count and a feature that indicates unaligned (inserted, dropped) words. The second type are sparse features that operate on the lexical translation of the most frequent words.

3.2.1.0.1 Coarse features Two features included in the Thrax grammar extractor were included in the Moses model estimation. The first addresses the issue of rare phrase pairs, whose conditional translation probabilities are often over-estimated. In the extreme case, a singleton source phrase has conditional direct translation probability of 1, because it only has one translation. We introduce a cost feature of –1/count (in log space). The feature has a high negative value for rare counts, but little impact for frequent phrase pairs.

A second feature counts the number of words that are unaligned, detecting inserted and dropped words. With a list of function words, this feature can make a distinction between unaligned function words (a common occurrence) and unaligned content words (typically a problem).

Table 3.3 presents results for these features for a phrase-based German–English system. For this language pair, the features have no effect.

3.2.1.0.2 Sparse features For frequent words, lexicalized features are feasible even in tuning (i.e., discriminative training on a several thousand sentence held-out set). We added the features used in a recent paper by [40] on their pairwise-ranked optimization (PRO) tuning method to Moses. All these features are count features that count

- number of times a specific source word is dropped
- number of times a specific target word is inserted

Features	Tune	Test
Baseline	24.8	21.3 (0.98)
+ Low count feature	24.8	21.3 (0.99)
+ Unaligned penalty	-	21.3 (0.98)

Table 3.3: Porting features from Joshua into phrase-based Moses for German–English. Scores are reported as BLEU (length ratio).

Features	German–English	English–German
Baseline	21.5 (0.99)	15.6 (0.96)
Word translation features	20.9 (1.01)	15.5 (0.97)
Source word deletion features	20.3 (0.98)	15.5 (0.96)
Target word insertion features	20.6 (0.98)	15.4 (0.96)
All features	21.2 (0.99)	15.4 (0.96)

Table 3.4: Sparse features used by [40], optimized with an re-implementation of their method, and tested on the German–English language pairs.

- number of times a specific source word is translated into a specific target word

The feature operate on the 50 most frequent words for the insertion and deletion feature and the 80 most frequent words for the translation feature.

Feature weights are optimized with PRO, results are given in Table 3.4. For both German–English and English–German, tuning was not able to properly learn feature weights in the face of the added sparse features. We are currently exploring alternatives to the tuning method.

At this point, the sparse features are only integrated into the phrase-based decoder, but they will be added to the chart decoder imminently.

3.2.2 Improvements to Minimum Error Rate Training

The well-established minimum error rate training (MERT) method is known perform poorly when many features are added (above 15–20, or so). It is also notorious for instability. If run multiple times, very different tuning and test results may be observed.

We implemented two changes to MERT that attempt to address these problems: optimization in random directions and re-using the optimized weight settings of prior iterations as starting points.

3.2.2.0.3 Random directions The first change addresses the problem that optimizing one parameter at a time is a very limited exploration of a high-dimensional space, and especially does not work when two (or more) features need to be traded off against each other. For instance, an increase in language model weight may have to be done alongside an decrease in the weight of the word count feature. A solution is to optimize not only parallel to the co-ordinate axis of the space, but also in random directions [41].

3.2.2.0.4 Historic best MERT runs over several iterations of first optimizing weights and then re-running the decoder with the weight setting to validate them. The algorithm might find good weight settings in an earlier iteration, but then gets off track and only finds worse ones later. If the historic best weight setting are re-used as starting points for the line search optimization, then this can be prevented. [42] also argue that this adds to the stability of the optimization.

Experimental results with these changes are reported in Table 3.5. While there are no large gains either in terms of better BLEU scores or lower variance. For all four settings, performance on the tuning set is slightly higher, but performance on the test set is mixed. Only for the German–English factored backoff model, which has the large number of 29 features, we see a small improvement of 0.1 BLEU points.

Urdu-English, SAMT Model	**Iterations**	**Tune**	**Test**
Baseline	11.6 (std 4.8)	22.73 (std 0.07)	21.54 (std 0.38)
50 random directions	9.4 (std 2.3)	22.82 (std 0.14)	21.58 (std 0.38)
Random directions + historic best	9.2 (std 5.9)	22.79 (std 0.23)	21.40 (std 0.37)
Urdu-English, Hierarchical Model			
Baseline	8.8 (std 2.2)	23.91 (std 0.18)	23.02 (std 0.42)
50 random directions	8.4 (std 3.3)	23.85 (std 0.35)	22.80 (std 0.70)
Random directions + historic best	12.0 (std 3.5)	24.03 (std 0.23)	22.89 (std 0.18)
German-English, Phrase-based			
Baseline	7.2 (std 14.3)	24.82 (std 0.04)	21.29 (std 0.05)
Random directions + historic best	6.6 (std 1.8)	24.88 (std 0.07)	21.28 (std 0.16)
German-English, Factored Backoff			
Baseline	12.0 (std 15.2)	24.89 (std 0.25)	21.35 (std 0.15)
Random directions + historic best	11.4 (std 7.6)	25.01 (std 0.12)	21.45 (std 0.12)

Table 3.5: Improvements to Minimum Error Rate Training (MERT): Results on 5 runs for each setting, with reported standard deviation between them.

Urdu-English, SAMT Model	**Iterations**	**Tune**	**Test**
MERT	11.6 (std 4.8)	22.73 (std 0.07)	21.54 (std 0.38)
PRO	10	-	21.33 (std 0.13)
Urdu-English, Hierarchical Model			
MERT	8.8 (std 2.2)	23.91 (std 0.18)	23.02 (std 0.42)
PRO	10	-	21.93 (std 0.36)
German-English, Phrase-based			
MERT	7.2 (std 14.3)	24.82 (std 0.04)	21.29 (std 0.05)
PRO	10	-	21.29 (std 0.02)
German-English, Factored Backoff			
MERT	12.0 (std 15.2)	24.89 (std 0.25)	21.35 (std 0.15)
PRO	25	-	21.58 (std 0.11)
PRO (max-iter 10)	10	-	21.54 (std 0.10)

Table 3.6: Minimum error rate training (MERT) vs. pairwise ranked optimization (PRO)

3.2.3 Pairwise Ranked Optimization

During the course of the workshop, [40] presented a paper proposing an alternative to MERT, which they call pairwise ranked optimization (PRO). In this method, pairwise samples from each n-best list are drawn and used to train a classifier that predicts the better scoring translation (according to sentence level BLEU+1) based on their feature scores. They show good results on some language pairs for the sparse feature functions we described above. We re-implemented this method during the workshop.

Table 3.6 shows a comparison between MERT and PRO. For the Urdu–English models, PRO performs worse, for German–English phrase-based model similar, but for the feature-rich German–English factored model we see gains. Across the board, variance is reduced.

We investigated a number of variants of the basic method. PRO is very fast, so that the bulk of the time is spent on generating the n-best lists. Can we get away with tighter beam settings during tuning? Table 3.7 shows that even by drastically reducing beam sizes, we obtain basically the same results. Thus we can run the decoder much faster, or increase the size of the tuning set. The table also includes numbers of the effect of increasing the tuning set from about 3000 sentences to about 7000 sentences. We see a gain of about 0.1 BLEU.

Finally, explored a number of additional variations:

- [40] suggest to interpolate at each iteration the current weight setting with the prior weight setting,

German-English, Phrase-based	Test
Baseline	21.29 (std 0.02)
+ pop-limit 1000	21.32 (std 0.03)
+ pop-limit 200	21.32 (std 0.02)
+ pop-limit 1000 + large tuning set	21.43 (std 0.02)
German-English, Factored Backoff	
Baseline	21.54 (std 0.10)
+ pop-limit 1000	21.55 (std 0.02)
+ pop-limit 200	21.58 (std 0.04)
+ pop-limit 1000, large tuning set	21.63 (std 0.05)

Table 3.7: Impact of tighter beam sizes and larger tuning sets on pairwise ranked optimization

Setting	Iterations	Tune	Test
PRO	10	-	21.63 (std 0.05)
+ historic-interpolation 0.1	25	-	21.59 (std 0.03)
PRO-MERT	9.6 (std 1.8)	23.28 (std 0.02)	21.55 (std 0.06)
+ historic best	9.6 (std 1.8)	23.28 (std 0.03)	21.52 (std 0.08)

Table 3.8: Variants on pairwise ranked optimization (German-English, Factored Backoff, pop-limit 1000, large tuning set)

with giving the prior weight setting 90% of the weight. The motivation is to further stabilize the method.

- We used the PRO generated weights as starting point for MERT optimization. This will result in better performance on the tuning set at each iteration and overcome discrepancies between sentence-level BLEU scores and corpus-level BLEU scores.
- When combining PRO and MERT as above, we may also use the optimized weight settings from all prior iterations as additional starting points.

As Table 3.8 shows, none of these variations are helpful for our feature-rich German–English factored backoff model.

Part II

Features for Discriminative Language Modeling

Chapter 4

Conversation-oriented semi-supervised features

The perceptron algorithm was used in [3] to estimate discriminative language models which correct errors in the output of ASR systems. In its original form, the algorithm simply increases the weight of n-gram features which appear in the correct (oracle) hypothesis and decreases the weight of n-gram features which appear in the 1-best hypothesis. In this paper, we show that the perceptron algorithm can be successfully used in a semi-supervised framework, where limited amounts of labeled data are available. The main idea is to use large amounts of *unlabeled* data to derive generalizable features for the labeled data, based on which the perceptron algorithm trains the discriminative model. Experiments conducted at the 2011 CLSP Summer Workshop on the conversational telephone speech corpora Dev04f and Eval04f demonstrate the effectiveness of the proposed approach. The perceptron algorithm is a *supervised* learning algorithm, which takes as input labeled data and learns a model that separates the data that belong to different classes [8]. The perceptron has more recently been used successfully in the area of language modeling [3, 43], where it takes as input N-best lists from an ASR or MT system and returns feature weights that re-rank the N-best lists with the goal of scoring the less-erroneous hypotheses higher. The features used in [3, 43] are n-grams; these n-grams will necessarily have to have a non-negligible overlap between the training and the test data, to have some hope that the learned model will be able to generalize. To improve generalization, some additional features such as morphological, syntactic and trigger features have been investigated across a variety of languages and tasks [4, 5, 6, 7].

In this paper, we investigate the use of *unlabeled* audio and its derivatives (word lattices) in order to guide the generation of features. Despite the fact that the unlabeled audio cannot be used in the perceptron algorithm directly (by virtue of the fact that supervised transcriptions are missing), it can still provide some form of confidence estimates for the *labeled* data. Therefore, even if the n-gram overlap is small, there are still features which provide additional means of confidence based on which the perceptron can learn to successfully re-rank the n-best lists. To the best of our knowledge, this is the first attempt at using unlabeled data for supplying a wide variety of features for a discriminative model.

Our experiments from conversational telephone speech (Dev04f and Eval04) show that the semi-supervised features offer statistically significant improvements over a strong ASR baseline, as well as a perceptron baseline which only uses 3-gram features. We follow standard perceptron training as presented in Section 1.2, with simple n-gram features forming our baseline discriminative language modeling condition.

The paper is organized as follows: Section 4.1 presents details about the semi-supervised features that are derived using unlabeled data; Section 4.1.5 presents other features, including supervised language model features; Section 4.2 presents our experimental setup and word-error-rate (WER) results, and finally, Section 4.3 contains concluding remarks.

4.1 Features

In this section, we show how to augment the feature vector Φ with a richer set of "semi-supervised" features, extracted using unlabeled data. We also describe some other features used in the trials reported in this chapter.

In order to tackle the issue of limited training data, and to make them complementary to the baseline n-gram features used in perceptron, these "external" features should satisfy criteria such as (i) they should be limited in number, to avoid overfitting; (ii) they should correlate with (multiple, and hopefully diverse) notions of confidence.

Note that what we propose here is not the same as what is usually done in semi-supervised learning, where the labeled data *propagate* their confidences to the unlabeled data [44]. Our approach is going in the opposite direction: we want to use the most relevant and confident unlabeled data to augment the features of the labeled data, over which discriminative training is applied.

In the rest of this section it is assumed that the unlabeled data are decoded with the same recognizer used to decode the supervised data which train the perceptron, as well as any other development or test data. The decoded output consists of lattices (or confusion networks).

4.1.1 Quantized rank features

Before moving on to discussing the specific real-valued measures that we derive from the unlabeled data, we will first discuss our use of quantized rank indicators as the actual features being used in the discriminative language model. For each measure F, we score each hypothesis h in the n-best lists with $F(h)$ and rank the n-best list according to the score. The rank k is then used to define a boolean indicator feature, e.g., RANK.F=k. To avoid feature sparsity, we use quantized ranks, i.e., rank intervals. The intervals near the top of the list are shorter, to emphasize the higher ranks.

4.1.2 Tf-idf Similarity Rank Features

The lattices of the unlabeled data are used in order to compute a vector representation of each conversation side. In the information retrieval framework, a conversation side plays the role of a "document". Specifically, each conversation side is represented by a vector, whose i-th entry is set equal to a measure of how frequent the word is in the conversation side, discounted by a factor that measures its frequency in the whole collection. A similar representation can be used with other document definitions.

Since each tf-idf vector is compared (using cosine distance) to a single hypothesis, we follow [45] in using two definitions of term-frequency, in order to reduce the influence of very frequent words. In addition to typically defined term frequency, we also use the probability of occurrence of a word in a document. This is computed in a lattice by summing together the posterior probabilities of paths which contain the word, and can be done easily with FSM operations. These probabilities are then convolved together to give the probability of occurrence of a word in a document. This measure of frequency is always bounded above by 1.

Thus, we compute two *tf* quantities for word w in a document c; the first contains the expected count in c, while the second is the probability of occurrence in c:

$$tf_c^{(1)}(w) = \sum_{\text{utt} \in c} \sum_{n=1}^{\infty} n \Pr(w \text{ appears } n \text{ times in utt}), \tag{4.1}$$

and

$$\begin{aligned} tf_c^{(2)}(w) &= \sum_{\text{utt} \in c} \sum_{n=1}^{\infty} \Pr(w \text{ appears } n \text{ times in utt}) \\ &= \Pr(w \text{ appears} \geq 1 \text{ times in } c). \end{aligned} \tag{4.2}$$

The document frequency (*df*) component is then calculated over the whole unlabeled corpus, generalizing directly to typical document frequency calculation:

$$df(w) = \sum_{c \in \mathcal{U}} tf_c^{(2)}(w). \tag{4.3}$$

The cosine similarity between a hypothesis $\mathbf{h} = (w_1, \ldots, w_k)$ and an unlabeled document $\mathbf{c} \in \mathcal{U}$ is then defined as

$$\text{sim}(\mathbf{h}, \mathbf{c}) = \frac{< \text{tfidf}(\mathbf{h}), \text{tfidf}(\mathbf{c}) >}{\|\text{tfidf}(\mathbf{h})\| \cdot \|\text{tfidf}(\mathbf{c})\|} \tag{4.4}$$

where tfidf($\boldsymbol{h}$) is defined as a vector whose elements correspond to the words of the decoding vocabulary and are equal to $tf(w)\log(D/df(w))$, where D is the total number of documents in the unlabeled corpus. The notation $<\cdot,\cdot>$ denotes vector inner product, and $\|\cdot\|$ denotes vector norm.

The features computed for each hypothesis $\mathbf{h}$ are the average and maximum of the set of tf-idf similarities:

$$\mathbf{f}^{(\text{AVG})}_{\text{tfidf}}(\mathbf{h}) = \frac{1}{D}\sum_{\mathbf{c}\in\mathcal{U}} \text{sim}(\mathbf{h}, \mathbf{c}), \tag{4.5}$$

$$\mathbf{f}^{(\text{MAX})}_{\text{tfidf}}(\mathbf{h}) = \max\{\text{sim}(\mathbf{h}, \mathbf{c})\}_{\mathbf{c}\in\mathcal{U}}. \tag{4.6}$$

This yields four features, since there are two versions of tf used to compute each of these features. We further double these to eight features by including versions of each that are multiplied by the ratio $|\mathbf{h}|/(|\mathbf{h}|+1)$, which is an increasing function of $|\mathbf{h}|$, as a means of penalizing very short hypotheses. All of these features are converted to boolean indicator features based on quantized ranks, as discussed in Section 4.1.1.

4.1.3 Topic-model Rank Features

The next set of features is computed using topic features. As described above, expected counts are computed from the lattices of the decoded audio and aggregated at the document (conversation side) level. These expected counts are then used to generate "pseudo-documents" as bags of words (each word is simply repeated as many times as the expected count, rounded to the nearest integer), based on which topic distributions are estimated with the Mallet toolkit [46]. Excluding function words, three different topic models are estimated, with 50, 100 and 200 topics respectively. Three more topic models are estimated (with 50, 100, 200 topics again), but with function words included.

Each topic t represents a distribution over words $p_t(w)$. To assign a topic-based likelihood to a hypothesis, all hypotheses of the conversation side it belongs to are first treated as a "document", based on which a distribution over topics $q(t)$ is estimated:

$$q^* = \arg\max_q \sum_{w\in\mathbf{c}} \log(\sum_t q(t)p_t(w)).$$

Next, given this estimate of topic proportions, the log-likelihood of a hypothesis $\mathbf{h}$ is computed as

$$LL(\mathbf{h}) = \sum_{w\in\mathbf{h}} \log(\sum_t q^*(t)p_t(w)).$$

This log-likelihood is then used to rank hypotheses, and results in boolean indicator features based on quantized ranks, as discussed in Section 4.1.1. Six different log-likelihoods are used to produce features in this way, corresponding to the 3 models with different numbers of topics, each either with or without function words.

4.1.4 Clustering Features

The next set of features is based on clusters over the unlabeled audio documents. The pairwise similarity between all pairs of documents is first computed using eq. (4.4) as the similarity measure. This generates a weighted graph, which is subsequently thresholded such that only edges with a tf-idf similarity above θ are kept. The resulting graph is then given as input to a clustering algorithm [47] which optimizes the normalized-cut objective function

$$NCut(G) = \min_{C_1,\ldots,C_k} \sum_{i=1}^{k} \frac{\text{links}(C_i, \mathcal{U}\setminus C_i)}{\text{links}(C_i, V)},$$

where $\mathcal{U}$ is the set of vertices (conversation sides) in the unlabeled audio, C_i is the set of vertices of the i-th cluster, and the function links(A, B) is equal to the sum of the weights of the edges between vertices in A and vertices in B.

New clustered "documents" are now created: the number of words in each cluster is set equal to the sum of the expected counts of its constituent documents. Two values of θ were used: 0.21 and 0.25, giving rise to two sets of features.

For each hypothesis $\mathbf{h}$, several symbolic features are generated as follows: for each word w in $\mathbf{h}$ whose *df* is in the top 95% percentile (that is, function words are largely excluded) all clusters $\mathcal{C}_w$ which contain the word are identified, and the cosine similarity (4.4) between $\mathbf{h}$ and each $C_i \in \mathcal{C}_w$ is computed. The symbolic features are then formed by concatenating together the identity of w and the identity of each cluster in $\mathcal{C}_w$ which results in the largest tf-idf similarity (to avoid sparsity, only the top 10% of the features are kept, over all $w \in \mathbf{h}$).

4.1.5 Other Features

In addition to the semi-supervised features, other features derived from the n-best lists were used. Specifically:

- **length:** To make sure that hypotheses do not deviate too much from the "centroid" of the lengths in a n-best list, two "delta" features are computed for each hypothesis: the first is the rank of the absolute difference between the length of the hypothesis and the mean length in the n-best list, while the second uses the median length instead of the mean.
- **original rank:** The original rank of a hypothesis is also used as a separate feature.
- **LM features:** Nine language models, computed from a variety of sources, are used to compute additional "supervised" features. The nine sources were comprised of the supervised transcripts of all 2000 hours, and the web data that were obtained from UW [48]. In total, about one billion tokens were used to generate these language models. The likelihoods of the languages models (computed for all n-gram orders 1..5) were again converted to ranks.

All of these ranks were quantized as presented in Section 4.1.1.

4.2 Experimental Results

All our experiments were done on conversational telephone speech (CTS) corpora. The IBM "Attila" speech recognizer [49] was trained on 2000 hours of speech, consisting of roughly 1000 hours of conversations from the Fisher corpus, and another 1000 hours of conversations from the Switchboard and the Call Home corpora. The same 2000 hours were decoded with the learned acoustic model, but the decoding was done using 20-fold cross-validated language models. Specifically, for each fold i, all manual transcripts of the rest 19 folds (i.e., excluding fold i) were pulled together to build a 5-gram language model that was used to decode i.

Four folds, amounting to roughly 337 hours, were used as *labeled* data for training the various perceptron models. Another 15 folds, amounting to roughly 1545 hours, were used as *unlabeled* data for generating the semi-supervised features.

A small held-aside fold (not included in the above) was used as a stopping criterion for perceptron; the algorithm would stop if the WER did not improve on it for 5 consecutive iterations.

The Dev04f corpus, together with the held-aside fold, were used with cross-validation, for tuning the scaling factor used in the combination of the perceptron output and the score obtained by the baseline recognizer, as well as deciding which set of features to use in the test data. The Eval04f corpus was used as the test corpus for reporting blind WER results.

Table 4.1 shows the WERs on the dev & eval datasets, for three sets of features: (a) Plain 3-gram features (baseline perceptron); (b) the best semi-supervised feature set which does not contain the strong (supervised) language model features, tuned on the held-out and Dev04f data using cross-validation; (c) the best semi-supervised feature set which contains the supervised language model features (mentioned in Section 4.1.5), again tuned on the held-out and Dev04f data using cross-validation. As can be easily seen

	dev	eval
1-best ASR	22.8	25.7
baseline percep (3gram)	21.9	25.1
percep (semi-sup features)	21.7	**24.8****
percep (semi-sup+LM features)	21.7	24.9*

Table 4.1: WER results using various feature sets.

from these results, the semi-supervised features resulted in a significant improvement over the baseline on the eval data ($p < 0.01$). The inclusion of the language model features seems to degrade the result slightly (24.8 to 24.9) but it is significantly better than the baseline perceptron at the $p < 0.05$ level.

4.3 Concluding Remarks

We have investigated the use of unlabeled audio for generating features that are given as input to a supervised discriminative algorithm for language modeling. These "semi-supervised" features attempt to convey various notions of confidence: relevance in terms of tf-idf cosine similarity, coherence as measured by topic models, and as measured by proximity to (unsupervised) clusters of the unlabeled audio. Results from conversational telephone speech transcription show that these features offer statistically significant WER gains, even surpassing the performance of strong language models trained on supervised data.

Chapter 5

Continuous Space Discriminative Language Modeling

Discriminative language modeling is a structured classification problem. Log-linear models have been previously used to address this problem. In this paper, the standard dot-product feature representation used in log-linear models is replaced by a non-linear function parameterized by a neural network. Embeddings are learned for each word and features are extracted automatically through the use of convolutional layers. Experimental results show that as a stand-alone model the continuous space model yields significantly lower word error rate(1% absolute), while having a much more compact parameterization(60%-90% smaller). If the baseline scores are combined, our approach performs equally well.

5.1 Introduction

A language model(LM) assigns scores to word strings and plays an important role in automatic speech recognition(ASR) and many other applications. Under the predominant source-channel paradigm for speech recognition, the LM is generally used as the source of prior information, which evaluates the well-formedness of each hypothesis. Therefore, standard LMs are usually estimated from well-formed text in a maximum likelihood fashion. Specifically, the joint probability of a word sequence is often factorized into the product of local probabilities as below,

$$P(w_1, w_2, ..., w_l) = \prod_{i=1}^{l} P(w_i|h_i), \tag{5.1}$$

where h_i is the word history preceding the i^{th} word w_i. The set of conditional distributions $\{P(w|h)\}$ can be easily estimated based on empirical counts in the text corpus.

Discriminative training of LMs has been proposed as an effective complement to standard language modeling techniques [50, 3, 43]. Instead of attempting to learn a distribution over all possible word sequences, the discriminative objective directly targets the acoustically confusable hypotheses that the ASR system produces. To train such a model, we usually require transcribed speech data. An existing recognizer can be used to decode the speech and generate the corresponding confusion sets, and then discriminative training will learn to separate the lowest-error hypothesis from its set of competitors.

In discriminative language modeling, the LM is usually parameterized as a global linear model. Each word sequence W is associated with a score, which is the dot product between the feature vector $\Phi(W)$ and the parameter vector Θ. The probability of W is given by

$$P(W) = \frac{e^{\Phi(W)\cdot\Theta}}{Z}, \tag{5.2}$$

where Z is a normalizer. The features are in general represented symbolically, namely each word is treated as a distinct symbol. If n-gram features are used, each distinct word sequence of length up to n activates a different feature. This set of features can be very large and grows quickly with the training data.

In contrast, continuous feature representations are usually more succinct. Each word in the vocabulary is represented as a continuous vector. Word sequences can be represented as the concatenation of the representation for each word. Neural networks are powerful tools to learn such representations, they are also capable of learning non-linear features automatically. Therefore, it's not necessary to define any high order product features(e.g. bigrams, trigrams...) which often drastically increase the number of model parameters, all the complex non-linear interaction among words in the sequence can be discovered automatically.

Besides the space advantage, continuous feature representations are often believed to be able to generalize better. By mapping each word into a shared continuous space, word similarities and sequence similarities can be exploited. Therefore, if we have not seen some word sequence in the test data, it's more likely that we have seen something close to that in the continuous space.

LMs have been trained in the continuous space before [51, 52], but only for the standard generative language modeling, where the goal is to model $P(w|h)$. This work is different in that it's the first *discriminative* LM trained in the continuous space. The most closely related work to our knowledge is [53]. The authors described a general deep architecture based on convolutional neural networks that can be used for various tasks. Our work resembles theirs in that we also use convolutional layers in our architecture, but the important distinction is that the task we have is a *structured* prediction problem, as opposed to more of the standard classification problems presented in their paper. Therefore, our proposed architecture carries noticeable differences.

As we will show, the proposed model has a strong space advantage and also gives competitive word error rate results. When used as a stand-alone model without combining with the baseline scores, it outperforms the standard discriminative LM by a significant margin.

The rest of the paper is organized as follows: We'll talk about the training of standard discriminative LM in Section 5.2, the same loss function will be used for the continuous space model. We'll explain how to change the feature representation into continuous space in Section 5.3. Some training issues with the proposed model will be addressed in Section 5.4. Experimental results will be presented in Section 5.5.

5.2 Perceptron and maximum conditional likelihood training

The model in (5.2) can be trained in several ways. Perceptron training and maximum conditional likelihood(MCL) training are two commonly used approaches. Both methods share the similar intuition–increase the probability of the oracle hypothesis W^*, while penalizing other hypotheses in the set of likely candidates output by the recognizer. We denote this set as $GEN(A)$, where A stands for acoustics. The loss functions of the two methods are shown in (5.3) and (5.4).

$$\mathcal{L}_P = \left(-\log \frac{P(W^*)}{P(\hat{W})} \right)^+, \tag{5.3}$$

$$\mathcal{L}_{MCL} = -\log \frac{P(W^*)}{\sum_{W \in GEN(A)} P(W)}. \tag{5.4}$$

Perceptron training is an iterative procedure, which requires in each iteration identifying the best hypothesis according the current model. The loss only occurs if the current model fails to rank the oracle hypothesis on top of the others. Meanwhile, MCL training directly considers all the competing hypotheses in $GEN(A)$, the loss exists for all training instances, therefore, the optimization is generally more complex.

It's worth noting that with the standard feature representation, it's generally hard to train with MCL loss function without some kind of feature selection. The number of features in $GEN(A)$ is often too large to fit in memory. Fortunately, this problem will disappear with the continuous feature representation that we'll introduce in the next section.

5.3 Feature learning using convolutional neural nets

Note in (5.2), the score assigned to the word string is given by the dot product between the feature vector and the parameter vector. The model is linear in nature, and the features have to be specified beforehand.

In order to capture the non-linear interactions among words in the sequence, we have to define features that consist of combinations of words(e.g. bigrams, trigrams...). The number of such features grows quickly with data. To describe our approach of representing features, we first replace the dot product with some non-linear function $g(W;\Theta)$, thus the model in (5.2) becomes

$$P(W) = \frac{e^{g(W;\Theta)}}{Z}. \tag{5.5}$$

The neural network can be used to parameterize $g(W;\Theta)$. It has the ability to learn proper embeddings for each word, and automatically extract non-linear features useful for our task. In order to assign a score to each word sequence W, our network architecture has to deal with input of variable length, which is not what the standard neural network can handle.

[53] presented an architecture that can be built on top of variable-length word sequences. Central to their methodology is the use of convolutional layers. Fig. 5.1 shows the first three layers of their architecture. Each

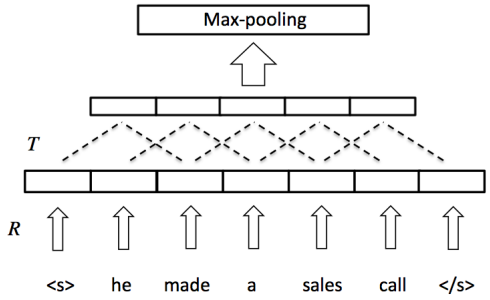


Figure 5.1: Feature learning on word sequences proposed in [53]

word in the sentence is associated with a continuous vector– this can be described as a look-up operation in the table R. The concatenated vector for the sentence does not have a fixed dimension either, therefore, features are extracted locally within fixed-size windows. As we can see, a linear transform T spans over only n words($n = 3$ here) and is applied to every n-gram in the sentence. Non-linear functions such as *tanh* are usually applied enabling nonlinear features to be detected.

For classification tasks such as the ones presented in [53], where the goal is to assign classes to the words in the input sequence, the features after the transform T in Fig. 5.1 usually has to go through a *max-pooling* layer which keeps only the top-k largest values in the feature vector. The resulting fixed-size vector can be used as input to the standard neural network layers.

Discriminative language modeling is a different classification task in which the classes are word sequences. The features we extract from each individual sequence are not used to construct the distribution over classes directly, thus the max-pooling layer is not necessary. The neural net in our architecture only has to output a score $g(W)$ for each word sequence W.

Fig. 5.2 shows the proposed architecture to parameterize $g(W)$. The first two layers are the same as in Fig. 5.1. After the shared transform T and some non-linear function, the extracted features for each n-gram undergo another shared linear transform F, the result of which is a score assigned to the n-gram. In order to obtain a score for the entire sequence, the scores for all n-grams are added up. Compared with the standard dot-product representation for $g(W)$, we replace the large number of n-gram features with a neural net structure that can be shared by all n-grams, allowing for a much more compact model.

Once we have a score for every hypothesis in $GEN(A)$, we can easily compute $P(W)$ and train our model within the composed neural network. The complete architecture of our approach is shown in Fig. 5.3.

5.4 Training issues

Albeit large, the architecture in Fig. 5.3 is not difficult to train. Gradient descent methods can be easily applied to all parameters in R, T and F with the help of back propagation. Taking advantage of the shared structures and tied parameters, the optimization procedure can usually be greatly simplified.

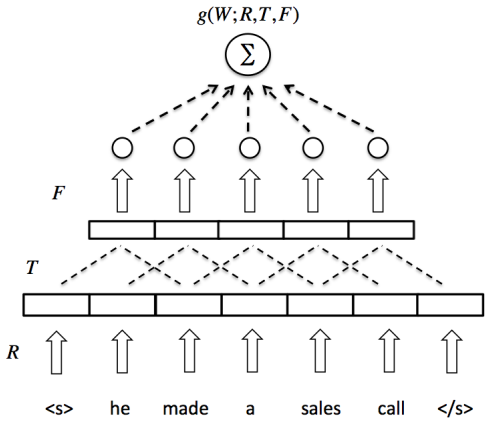


Figure 5.2: Neural net representation of $g(W)$ used in this paper

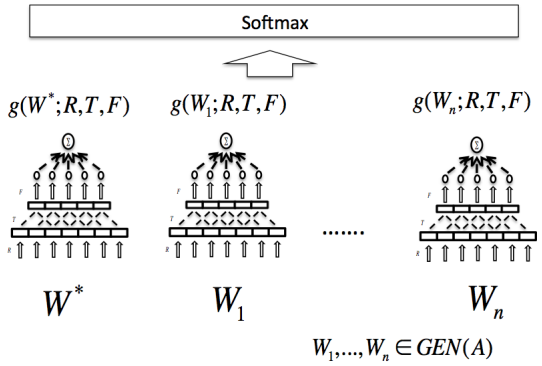


Figure 5.3: Full architecture for discriminative LM

When computing $g(W)$, we have to go through all n-grams for each W in $GEN(A)$. Fortunately, $GEN(A)$ usually contains hypotheses that share a lot of n-grams in common. Therefore, computing scores for these n-grams only has to be done once for each training instance. The back propagation stage can also benefit greatly from the fact that n-gram scores are linearly combined. To illustrate this, write $g(W)$ as the sum of each n-gram scores, namely $g(W) = g(s_1^W) + g(s_2^W) ... + g(s_{l_W}^W)$, where s_i^W denotes the i^{th} n-gram in W, and l_W is the total number of n-grams in W. Taking the gradient of the MCL loss function in (5.4), we have

$$\begin{aligned} \nabla \mathcal{L}_{MCL} &= -\nabla g(s_1^{W^*}) - \nabla g(s_2^{W^*}) ... - \nabla g(s_{l_{W^*}}^{W^*}) \\ &+ \sum_{W \in GEN(A)} P(W) \left(\nabla g(s_1^W) + \nabla g(s_2^W) ... + \nabla g(s_{l_W}^W) \right). \end{aligned} \tag{5.6}$$

As we can see, if some n-gram s appears the same number of times in W^* and all other W in $GEN(A)$, then $-\nabla g(s) + \sum_{W \in GEN(A)} P(W) \nabla g(s) = 0$, the contribution of s to the gradient is exactly zero. Therefore, it is also not necessary to go through all n-grams in the back propagation stage. Only n-grams that are not shared by $GEN(A)$ generate error signal to update the parameters.

In theory, any differentiable loss function can be applied on top of the architecture. However, the local optimum problem of training neural networks can make non-convex loss functions less desirable. The Perceptron training seems to have trouble reaching a good solution in our architecture. Therefore, we'll only report results using MCL trained models in the next section.

		dev	eval	#parameters
1-best ASR		22.8	25.7	
Perceptron	2 parts	22.3/31.1	25.1/30.2	1.7M
	4 parts	21.8/29.5	25.1/29.3	3.3M
	8 parts	21.7/29.4	24.8/29.1	5.3M
CDLM	2 parts	22.3/29.7	25.2/29.0	0.74M
	4 parts	21.9/29.0	24.9/28.5	0.32M
	8 parts	21.6/28.5	24.7/28.0	1.8M

Table 5.1: WER of Perceptron DLM and CDLM. Each cell contains the WER with/without combining with the baseline scores.

5.5 Experiments & Results

Our experiments are done on the English conversational telephone speech(CTS) dataset. The state-of-the-art IBM Attila recognizer is trained on 2000hrs of speech, half of them from the Fisher corpus, while the other half comes from Switchboard and Call-home. The baseline LM is a 4-gram LM trained on the 2000hrs of transcript, plus close to 1 billion words of web data. For training discriminative LM, the 2000hrs of transcribed speech data is divided into 20 partitions. The acoustic model trained on the 2000hrs is used to decode all the partitions. But the LM used to decode each partition only includes transcripts from the other 19 partitions. Such cross-validated language modeling is often used for training discriminative LM [3]. The hope is to avoid LM overfitting such that the confusions generated for each partition resemble those we'll see on the test data. We produce 100-best lists for all utterances in each partition for training discriminative LMs. The trained models are used to rerank the 100-best lists on the test data.

We train the standard averaged perceptron-based discriminative LM with trigram features in comparison with the continuous space discriminative LM(CDLM) using training sets of three different sizes. Training set I contains 2 partitions from Switchboard(130hrs). Training set II contains 3 partitions from Switchboard(245hrs) and 1 partition from Fisher(90hrs). Training set III set adds another 360hrs of Fisher data to training set II. The Dev04f corpus is used as the tuning set, consisting of 3.3hrs of speech. The final word error rates(WER) are reported on the 3.4hrs Eval04f corpus.

For training CDLMs, online gradient descent is used. The size of the feature extraction windows is fixed at three words, since it is generally thought that the standard discriminative LM generally does not benefit much from n-gram features beyond trigrams [3, 43]. The size of the word representation, and the size of the hidden layer are optimized on the development set. All the parameters in R, T, F are randomly initialized.

Table 5.1 shows the WER results of using the CDLM and the standard Perceptron DLM. Note that both the Perceptron and the CDLM scores can be combined with the baseline scores(AM+LM scores), which are generally important sources of information. The combination weights can be tuned on the develop set. As we can see, without the help of the baseline scores, the CDLM performs much better than the standard Perceptron. Nonetheless, the advantages disappear as the baseline scores are combined. The improvement achieved by the two approaches become almost identical. Being a stronger stand-alone model seems to prove the superior generalization ability enabled by the continuous feature representation. We may also have benefitted from the fact that the MCL loss function considers the entire confusion set. However, such advantages are apparently offset by the acoustic model and the maximum likelihood trained LM.

Besides the competitive performance, our CDLM technique consistently produces much smaller models(ranging from 60% to 90% smaller), and the sizes of which do not increase directly with more data. On the other hand, the number of features for the Perceptron DLM clearly has a tendency to increase as more training data becomes available.

5.6 Conclusion

We describe continuous feature representations for discriminative language modeling. Features are learned automatically through a novel neural network architecture.The resulting LM significantly outperforms the

standard Perceptron DLM as a stand-alone model. When combined with the baseline scores, our model performs equally well. The proposed architecture also produces much more compact models, reducing the number of parameter by 60%-90% in our experiments.

Chapter 6

Syntactic features

A perceptron is only as good as the features at its disposal, and one of the challenges of building a good discriminative model is deciding which features to use. Given the widespread success of n-gram generative language models, it is no surprise that n-grams also work quite well as features in a discriminative language model. Using n-grams up to trigrams as features, the perceptron does quite well, outperforming the baseline MT system in some supervised experiments.

But although n-grams can be effective on their own, it is useful to explore syntactic features as well. One advantage of syntactic features is that they are able to capture long-distance dependencies in the sentences. MT decoders are restricted to considering only local features of the sentence in order to keep inference tractable. Thus, unlike n-gram features, syntactic features give the perceptron access to information that is invisible to baseline MT system. Another advantage of syntactic features is that they are not as lexicalized as n-gram features, and thus they can generalize more easily to unknown words.

We explored many types of syntactic features, including both shallower features based on part-of-speech tags and deeper features which incorporated information from further up in the parse tree. In the next section, we discuss our experimental setup. In Section 6.2 we present results for the part-of-speech tag features, and in Section 6.3 we discuss the deeper syntactic features.

6.1 Experimental Setup

Table 6.1: Baseline Performance and Word n-Gram Features

	Baseline	**Perceptron**	
		Word 1-Gram	Word 3-Gram
↓TER	67.30	66.92	67.02
↑BLEU	20.51	19.85	20.10

We ran all of our experiments on the Urdu-English dataset, which contains about 88000 sentence pairs. We trained the perceptron, with MIRA, on fully supervised 100-best lists output by Joshua. All of our features were extracted from parse trees that we generated using the Berkeley parser, with the exception of the CCG supertag features, for which we used the C&C supertagger.

For each type of syntactic feature we tested, we ran one experiment using word 1-gram features, and another using word 3-gram features. This way, we could see how the syntactic features performed in compar-

Figure 6.1: A trigram.

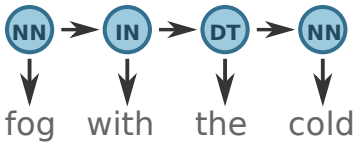


Figure 6.2: A sentence fragment with POS tags.

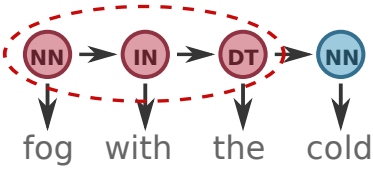


Figure 6.3: A part-of-speech n-gram.

ative isolation, as well as how they interacted with the standard word 3-gram features. Additionally, we ran two versions of each of these experiments: one where the perceptron was optimized for TER, and another where the perceptron was optimized for BLEU.

In Table 6.1, we compare our baseline machine translation system to the perceptron trained on just word n-gram features. While the perceptron outperforms the baseline system in terms of TER, it does worse than the baseline in terms of BLEU score. This is presumably because the baseline MT system was optimized for BLEU. In our experiments with syntactic features, we found that the perceptron consistently outperformed the baseline system in terms of TER, but had difficulty surmounting the baseline system's BLEU scores.

6.2 Part-of-Speech Features

We tested multiple types of part-of-speech tag features, which differed in how they combined POS tags with words from the sentence. Note that all of the features are text-based; two features whose representations differ by a single character are completely separate, and any resemblance of features to functions is purely aesthetic.

6.2.1 POS Tag n-Grams

The first type of features we tried were **POS tag n-grams** (see Figure 6.3). Within this framework, there were a number of choices to be made about how to represent the count of each POS n-gram in the sentence, and we tried out a few different options. In practice, it should be noted that when computing these features for a given sentence, we assumed that the sentence started with $n-1$ start-of-sentence POS tags, and ended with $n-1$ end-of-sentence POS tags.

- **POSCount(p,m)**: Let p be a POS-tag n-gram, and m be a positive integer. Then POSCount(p,m) is a binary feature which is turned on if and only if the POS n-gram p appears exactly m times in the sentence. Note that m must be positive; if a POS n-gram p appears 0 times in the sentence, there is no feature POSCount$(p,0)$, but instead the absence of p is represented implicitly by the fact that POSCount(p,m) will be turned off for all values of m. A problem with these features is that they do not take sentence length into account. A sentence of length five which contains five nouns, for instance, is very different from a sentence of length fifty which contains five nouns, but this type of feature has no way of capturing this distinction. The POSCountLength() and POSCountNorm() features discussed below were designed to address this concern.

- **POSCountLength(p,m/L)**: Again, let p be a POS tag n-gram and let m be a positive integer, and additionally let L be the length of the sentence. Note that in these features, all three numbers are written out explicitly; the '/' is just another character in the textual representation of the feature. These features are much like the POSCount() features discussed above, and differ only in that they

also include the sentence length. These features were designed to address the concern noted above, but unfortunately they were too sparse; our data did not contain enough sentences of each length. Thus we designed the POSCountNorm() to deal with this issue.

- **POSCountNorm(p,m)**: Here, p is once again a POS-tag n-gram, but m is now the number of times that n-gram appears in the sentence, normalized by the total number of POS n-grams in the sentence and binned by rounding to the nearest 10th (for unigrams) or 20th (for other values of n). If m is 0, it's because p appeared in the sentence but did not constitute more than a 10th or 20th of the total POS n-grams which appeared. For POS n-grams which actually appear 0 times in the sentence, there will be no POSCountNorm() feature turned on for that n-gram in that sentence, just like in the last two experiments. In this way, POS n-grams which appear with low frequency in sentences can be differentiated from POS n-grams which don't appear at all. This is especially important for higher values of n, where individual POS n-grams are likely to show up no more than once in a sentence.

- **POSExists(p)**: This binary feature will be turned on if and only if the POS n-gram p appears in the sentence. These features completely ignore the count of the POS n-gram p in the sentence. We tested these features in case the counts of the POS n-grams were irrelevant, and the mere existence of certain POS n-grams was enough to discriminate between good and bad translations.

- **POSList(p)**: Unlike the other POS n-gram features we tested, these are non-binary. Consider the list of POS n-grams which appear in a sentence. The value of POSList(p) is the number of times that p appears in that list. We tested these features in case the counts were useful for discrimination, but mentioning them explicitly made the features too sparse.

Table 6.2: POS Tag n-Grams – POSCount

	POSCount			
	↓TER		**↑BLEU**	
	Word 1-Gram	Word 3-Gram	Word 1-Gram	Word 3-Gram
POS 1-Grams	67.11	**66.71**	19.87	20.33
POS 2-Grams	67.04	66.99	19.89	20.10
POS 3-Grams	67.41	**66.89**	19.93	20.21
POS 4-Grams	67.37	67.03	20.04	20.19

Table 6.3: POS Tag n-Grams – POSCountNorm

	POSCountNorm			
	↓TER		**↑BLEU**	
	Word 1-Gram	Word 3-Gram	Word 1-Gram	Word 3-Gram
POS 1-Grams	**66.91**	66.97	19.81	19.99
POS 2-Grams	67.01	66.96	19.88	20.26
POS 3-Grams	67.13	66.98	19.93	20.47
POS 4-Grams	67.49	67.09	20.21	20.37

Tables 6.2 through 6.5 show the results for the POSCount, POSCountNorm, POSExists, and POSList features. Scores which appear in bold are those which improved on both baselines (the MT system score and the perceptron with just word n-grams). Overall, it seems that the POSCount features did best in terms of TER, while the POSCountNorm features had the best performance in terms of BLEU. This suggests that there are benefits to including the counts explicitly as part of the feature. Additionally, the POSExists features seemed to do better than the POSList features.

For TER, it seems that lower-order n-grams performed better. The best performance was seen on 1-gram features, and 4-grams did consistently poorly, suggesting that overfitting and data sparsity were issues for higher-order n-grams. For BLEU, 4-grams still performed poorly, but it is less clear whether 1-grams were

Table 6.4: POS Tag n-Grams – POSExists

	POSExists			
	↓TER		**↑BLEU**	
	Word 1-Gram	Word 3-Gram	Word 1-Gram	Word 3-Gram
POS 1-Grams	**66.89**	**66.85**	19.85	20.18
POS 2-Grams	67.05	66.99	19.89	20.09
POS 3-Grams	67.31	67.04	19.92	20.08
POS 4-Grams	67.11	67.12	20.34	20.34

Table 6.5: POS Tag n-Grams – POSList

	POSList			
	↓TER		**↑BLEU**	
	Word 1-Gram	Word 3-Gram	Word 1-Gram	Word 3-Gram
POS 1-Grams	67.12	67.11	19.87	20.01
POS 2-Grams	67.00	67.15	19.82	20.18
POS 3-Grams	67.11	**66.91**	20.02	20.30
POS 4-Grams	67.40	67.14	20.25	20.19

the best. Unsurprisingly, the performance was better for word 3-grams than for word 1-grams, both for TER and BLEU.

Although the POSList features performed worst in these experiments, most of the rest of our experiments just list features in that fashion, in part because this was easiest to program, and in part because preliminary experiments suggested that POSList features were not worse than the other three types shown here.

6.2.2 Interleaved POS/Word Tag n-Grams

Our next experiments tested features which incorporated both words and POS tags. First we examined **interleaved POS/word tag n-grams**, where some positions in the n-gram contain words and others contain POS tags. As an example, consider the following sentence with POS tags:

DT JJ NN ...
the big dog ...

Then some examples of interleaved POS/word n-grams for this sentence are

PW: DT big
WP: the JJ
PWP: DT big NN
WPW: the JJ dog

We characterize interleaved POS/word tag features by these acronyms (PW, WP, PWP, etc.), which describe the size of n is as well as indicating which positions are POS tags and which positions are words. Figure 6.4 shows another example of a PWP feature. These features can be thought of as less lexicalized versions of ordinary word n-grams.

Table 6.6 shows the results of our experiments with these features. As with the POSList features above, features here were listed, and thus features which appeared more than once in a sentence were counted as

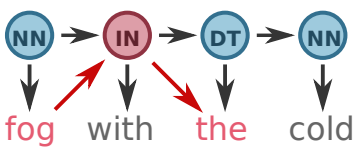


Figure 6.4: An interleaved POS/word n-gram PWP feature.

Table 6.6: Interleaved POS/Word n-Grams

	Interleaved POS/Word n-Grams			
	↓TER		**↑BLEU**	
	Word 1-Gram	Word 3-Gram	Word 1-Gram	Word 3-Gram
PW	67.17	**66.89**	20.20	20.27
WP	67.01	67.07	19.95	20.28
PWP	67.37	67.23	20.25	20.38
WPW	66.99	**66.85**	20.07	20.11
PWPW	67.16	**66.90**	20.26	**20.57**
WPWP	67.10	67.09	19.79	19.92
PWPWP	67.16	67.11	20.02	19.71
WPWPW	67.05	67.05	19.78	20.06
PWWWP	67.16	**66.84**	19.63	19.96
WPPPW	67.41	67.16	19.81	19.94
PWPWPW	**66.73**	67.26	20.05	20.00
WPWPWP	67.07	**66.88**	19.90	20.30

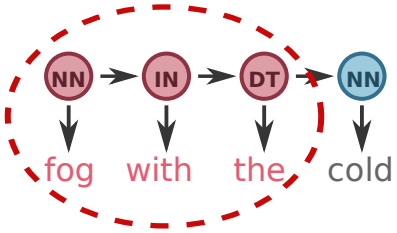


Figure 6.5: A combined POS/word n-gram.

many times as they appeared, making these features non-binary. Since these features are partially lexicalized, we do not expect them to appear multiple times per sentence, especially for the longer ones.

Results for these features were somewhat erratic, making it difficult to draw any conclusions. In most cases, the word 3-gram results were better than the word 1-gram results. However, it is hard to determine from these experiments whether longer or shorter interleaved n-grams were better. Also it should be noted that these features did manage to improve the baseline in some cases for both TER and BLEU.

6.2.3 Combined POS/Word n-Grams

Like the interleaved POS/word n-gram features, the **combined POS/word n-grams** use both POS tags and words. However, instead of including just the word or just the POS tag for a position in the n-gram, these features include both, as shown in 6.5. Again, these are non-binary features where the value of the feature represents the number of times that particular n-gram appeared in the sentence.

Table 6.7: Combined POS/Word n-Grams

	Combined POS/Word n-Grams			
	↓TER		**↑BLEU**	
	Word 1-Gram	Word 3-Gram	Word 1-Gram	Word 3-Gram
Combined 1-Grams	67.03	67.05	20.00	20.36
Combined 2-Grams	**66.81**	66.95	20.45	20.07
Combined 3-Grams	**66.89**	67.10	19.89	20.40

Results for these features are shown in Table 6.7. POS 2-grams and 3-grams appear to outperform POS 1-grams, but again it is difficult to draw conclusions from these numbers. For TER, word 1-grams did

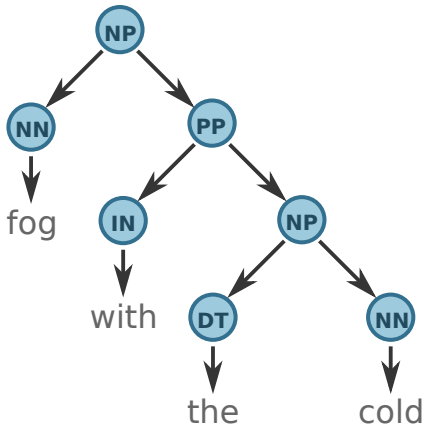


Figure 6.6: Part of a parse tree.

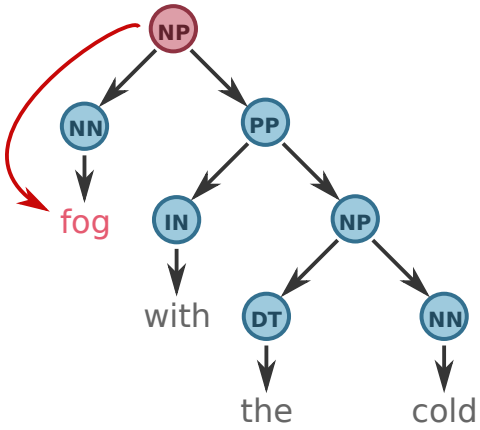


Figure 6.7: Example of a begin constituent feature.

better than word 3-grams, which can perhaps be attributed to the fact that word n-grams are somewhat redundant when using these features. These features seemed to perform about as well as the better interleaved POS/word n-grams.

6.3 Syntactic Features

In the previous section, we explored a number of features based around POS tags. These features showed some improvement over the baselines, especially in terms of TER. However, like word n-grams, POS tag n-grams are local and cannot capture long-distance dependencies in the data. In this section we explore deeper syntactic features that do.

6.3.1 Begin Constituent

The **begin constituent** features have the form BeginConstituent(C,W), where C is a constituent (possibly a POS tag), and W is the first (leftmost) word in the subtree rooted at that constituent in the parse tree. We hoped that these features would be able to discourage sentences where, for instance, an NP started with a singular noun when that noun should have been preceded by a determiner. For an example of the information captured by these features, see Figure 6.7. We also tested features of the form BeginConstituent(C,P) where C is a (non-POS-tag) constituent and P the leftmost POS tag in its subtree.

Table 6.8 shows the results for these features. We ran two sets of experiments, one for the POS version and one for the word version of these features. Neither set of features caused improvements over the baseline.

Table 6.8: Begin Constituent Features

	Begin Constituent			
	↓TER		↑BLEU	
	Word 1-Gram	Word 3-Gram	Word 1-Gram	Word 3-Gram
POS	67.09	67.20	19.98	20.06
Word	67.01	67.07	20.21	20.19

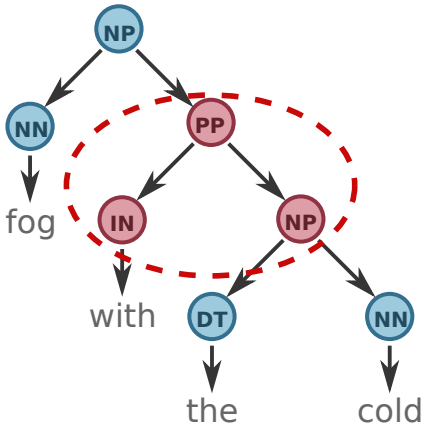


Figure 6.8: A CFG production feature.

6.3.2 CFG Rules

We then tested features which list the non-lexical **CFG productions** that appear in the parse tree for a sentence (see Figure 6.8). As with previous experiments, features which appear multiple times in the parse tree will be counted multiple times, and thus these are not binary features.

Table 6.9: CFG Production Features

CFG Productions			
↓TER		↑BLEU	
Word 1-Gram	Word 3-Gram	Word 1-Gram	Word 3-Gram
67.21	**66.58**	19.85	19.88

Table 6.9 shows the results for these features. Interestingly, these features do quite well for TER when combined with word 3-grams; in fact, these are the best results for TER. However, it is difficult to determine whether or not this is a fluke, especially considering that these features were less effective in the other three cases.

6.3.3 CCG Supertags

The **CCG supertag** features are similar to the combined POS/word features from above, except that in addition to POS tags and words they also include CCG supertags. For each value of n, we ran three versions of the experiment: one where the features included all three pieces of information (words, POS tags, and supertags), one where the words were excluded, and one where the part of speech tags were excluded.

The results of these experiments are shown in Table 6.10. Again, the results were somewhat erratic, and it is difficult to draw any conclusions. The best results were for supertag 2-grams, and in fact the supertag 2-grams, combined with word 1-grams, had the best BLEU score of any feature we tested. However, it is difficult to determine whether 2-grams were actually better than 1- or 3-grams, or whether supertag n-grams should be combined with word 1-grams or 3-grams, because there did not seem to be any conclusive patterns in the results.

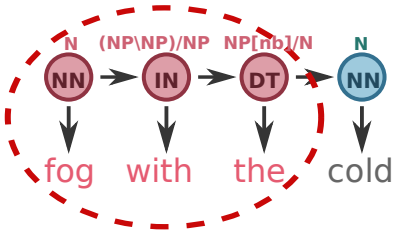


Figure 6.9: A CCG supertag feature.

Table 6.10: CCG Supertag Features

	CCG Supertag Features			
	↓TER		↑BLEU	
	Word 1-Gram	Word 3-Gram	Word 1-Gram	Word 3-Gram
1-Gram (no words)	66.96	66.94	19.79	20.12
1-Gram (no POS tags)	67.19	**66.85**	20.11	20.16
1-Gram (all three)	67.32	67.22	19.93	20.16
2-Gram (no words)	67.15	67.37	20.10	20.19
2-Gram (no POS tags)	**66.83**	67.08	**20.61**	20.17
2-Gram (all three)	**66.81**	67.00	**20.61**	20.27
3-Gram (no words)	67.00	67.06	20.20	20.27
3-Gram (no POS tags)	67.11	67.20	20.24	20.20
3-Gram (all three)	67.30	66.96	20.03	19.94

6.3.4 Interleaved Constituents and Words

The final type of feature that we tested involved **words interleaved with constituents**. These features generalize the interleaved POS/word features described above; for example, the WPW feature tested above is a subset of the WCW feature that we test in this section. Of all of our features, these ones most directly reveal long-distance dependencies, since entire constituents are skipped over so that the words on either end can interact. We also tested features where POS tags, rather than words, were interleaved with constituents. Since POS tags are a kind of constituent, POS n-grams are a subset of these features.

We ran two experiments on the WCW features; in the second experiment, beginning-of-sentence and end-of-sentence words and POS tags were added to the parse trees, while in the first they were not. We were unable to test the WCWCW features with word 3-grams, because the number of features involved exceeded the maximum imposed by the perceptron implementation we used for testing.

Table 6.11: Begin Constituent Features

	Interleaved Constituents and Words			
	↓TER		↑BLEU	
	Word 1-Gram	Word 3-Gram	Word 1-Gram	Word 3-Gram
WCW (1)	**66.66**	67.02	20.25	20.05
WCW (2)	66.94	67.03	20.33	20.31
WCWCW	67.04	—	19.85	—
PCP	67.21	67.22	20.34	20.47
PCPP	67.27	67.09	19.79	20.26
PPCP	67.45	**66.77**	20.26	20.44
PPCPP	67.10	67.25	20.02	20.26

Results for these features (shown in Table 6.11) were erratic again. These features did worse in terms of TER than other features we tested, and none of them improved the BLEU score. However, there were many

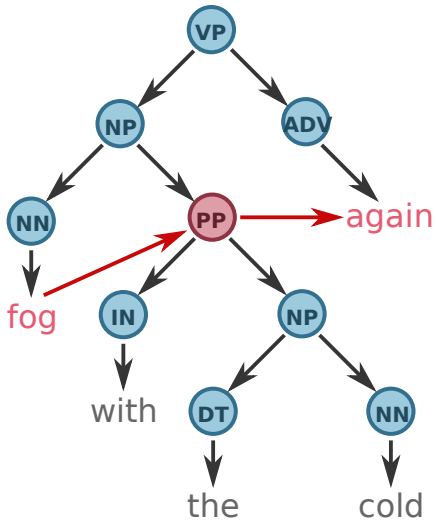


Figure 6.10: An interleaved constituent/word feature.

BLEU scores which came very close to the baseline, and overall BLEU scores seem to be higher for these features than for others.

6.4 Conclusion

In the experiments we ran, we found that the perceptron improves on the baseline MT system in terms of TER, and that many of the syntactic features we tested decreased TER even further. The best features for TER were the CFG productions combined with word 3-grams. In that experiment, the perceptron used 2298070 features, including the word n-grams as well as the CFG productions.

On the other hand, the perceptron had difficulty matching the baseline MT system's BLEU scores. Many of the syntactic features improved the BLEU score compared to using just the word n-gram features, but few did better than the MT baseline. The best features for BLEU were the CCG supertag bigrams combined with word 1-grams. In that experiment, 1006809 features were used, including the word 1-grams.

We were able to test many different feature types, running multiple experiments with each type of feature. This would not have been possible without the perceptron reranker, whose flexibility allowed us to test out new features quickly and easily.

Part III

Confusion set generation for Discriminative Language Modeling

Chapter 7

MT Hallucinations

7.1 Discriminative reranking

Discriminative reranking of n-best machine translation output is a way of bringing better translations to the top of the n-best list than the MT model predicted. Even though the decisions of the underlying system restrict the reranker in its ability to select good translations, reranking can still give improvements over the baseline results. Besides, since reranking works on complete candidate translations it is possible to compute global features that would not be available during search. It also offers the possibility of trying out more complex features offline, i.e. without the need to incorporate them into the decoder. Of course there is a trade-off between reduced decoding complexity and fully leveraging the discriminative power of additional features during search. However, even if we would ultimately like to incorporate as many features as possible into the decoder, there is benefit in being able to experiment on n-best outputs before making the effort of adding complexity to the decoder.

7.2 Data preparation for discriminative training

An important part of the work on this project was to prepare new data sets for discriminative training and to compare the usefulness of different approaches to creating discriminative training data for language modeling. We used baseline models for two different language pairs in both translation directions.

Urdu↔English (small): ∼1 million words/88K parallel sentences (newswire/ newsgroup data, MT08), hierarchical models

German↔English (large): ∼50 million words/1,8M parallel sentences (Europarl v6 news commentary v6), phrase-based models

For the English→German and German→English systems, additional data from the news domain (news2007-2011, together about 31M sentences) was used for training the language models. All systems were trained with the Moses toolkit. A line in the n-best lists produced by Moses typically looks like this:

```
0 ||| resumption of the session ||| d: 0 -0.0059175 0 0 0 0 0 lm:
-18.2142 w: -4 tm: -0.00593459 -3.06382 -0.0235307 -3.61361 0.999896
||| -1.3447
```

The first column contains the sentence id, followed by the translation string in the second column, the values for every feature function in the third column and the total weighted score in the last column.

7.2.1 Jackknifing

Different from generative language models that only look at examples of what constitutes a good sentence, discriminative language models (DLM) try to distinguish between target language strings by looking at both

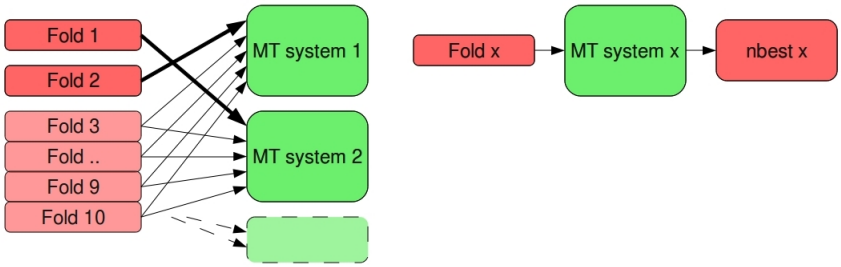


Figure 7.1: 10-fold split of training data

good and bad examples of target sentences. This implies that training data needs to consist of more than a single sentence per example and that we need reference translations to tell us which of them are more likely to be correct.

Since the task is to rerank n-best lists it makes sense to also train on n-best lists or similar data in order to match the test conditions as well as possible. Translating the training data with the baseline systems to produce n-best lists would not produce realistic data because the phrases in the translated text would have been seen during training of the baseline systems and hence the translations would be better than in a real test situation. Therefore, a jackknifing strategy is applied in order to translate the training data without any bias, as shown in figure 7.1. The training data is split into 10 folds, and an SMT system is trained for each set of 9 out of the 10 folds (10 systems per translation direction). Then each system is used to translate the respective remaining fold into an n-best list, hence producing translations for the whole training set.

7.2.2 Data conditions

With the jackknifing strategy in mind, three kinds of training data are produced for discriminative training as shown in figure 7.2. The "one-way" data condition simply follows the description above and produces n-best lists for all the training data by translating the 10 training folds. The n-best lists together with the reference translations are then used as discriminative training data. For the "round-trip" data condition, we assume that only the target side of the training data is available and produce n-best lists by round-trip translations from target to source and back to the target. When translating back to the target side, the 1-best translations from the first step are used as source sentences. This setup requires 10 SMT systems for each translation direction since the jackknifing strategy needs to be applied in both translations directions. The "monolingual round-trip" data condition is similar to "round-trip", with the only difference that we take additional monolingual data that is different from the training data, even though it needs to be similar in domain to avoid out-of-vocabulary issues. Being able to apply the discriminative training scheme to arbitrary amounts of monolingual data is very important for training sparse features and therefore the main objective.

The Urdu↔English baseline systems were used to translate additional monolingual data from the news domain (100K sentences from news2011) from English to Urdu and back to English. The German↔English systems were used to translate some of the news data that was already used for the language models (news2010, news2011). Because of overlap with the language model when translating back into English, each block of 100K sentences was split up into 10 folds and each fold translated with a language model trained without that fold.

7.2.3 Terminology

The intuition behind using n-best lists for discriminative training is to get examples of output sentences that the translation model could be confusing with the correct output. Therefore, we refer to hypothesis translations in n-best lists also as "confusions". The data conditions described earlier differ in the way these confusions are produced and therefore we distinguish between "real confusions", those that were created by translating from real source sentences, and "hallucinated confusions", those that were created by translating from MT output. In the next paragraph, we describe another way of creating hallucinated confusions with monolingual confusion grammars.

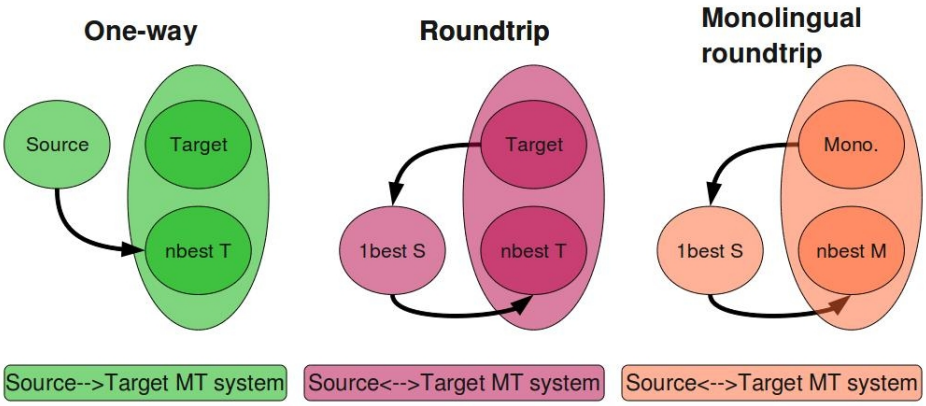


Figure 7.2: Data conditions for discriminative training

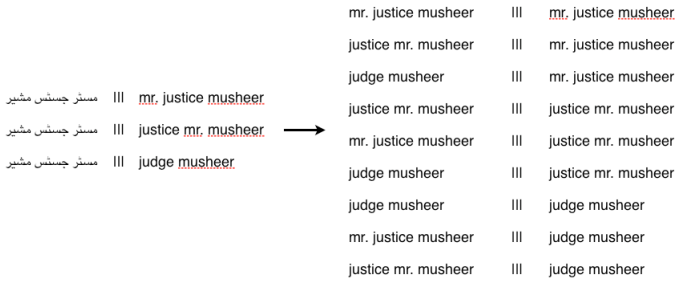


Figure 7.3: Examples of English→English confusion rules generated from an SAMT grammar

7.2.4 Pivot method

One way to approximate round-trip translation is to create an English-to-English confusion grammar using another language as a Pivot language. We start by using Joshua to build a standard one-way Urdu-to-English SAMT grammar from the 88K Urdu-English corpus described in section 7.2. Each rule in the grammar has an Urdu component and an English component. For each rule in the grammar, we locate every other rule in the grammar that contains the Urdu component of that rule. We then build new confusion rules (∼9M) linking the English components of those rules by pivoting on their common Urdu component, as illustrated in figure 7.3. When combining the English components of two rules into a single rule, we multiply any feature function scores that are probabilities, following [54], and recalculate all other scores.

Note that monolingual translations generated from a confusion grammar built in this way are likely to contain perfectly legitimate paraphrases, which will not be useful when training the discriminative language model. In order to identify these potential legitimate paraphrases, we build a second English-to-English confusion grammar from a German-English grammar built on a 88K-sentence subset of the German-English parallel corpus described in section 7.2. We then remove all rules from the Urdu-derived confusion grammar that also appear in this second German-derived confusion grammar (∼1,6M rules), leaving us with ∼7,4M rules.

7.3 Perceptron training for reranking

Our discriminative language models consist of a large number of binary n-gram features that need to be trained and we use online learning methods because they can handle an arbitrary number of features. The basic learner is a simple perceptron which we feed two translation hypothese at a time, a “good“ and a “bad“ hypothesis. The perceptron updates every feature that is present only in one of the two hypotheses. If a

```
..                      ..
1 31 1 9           1 31 8 7 6 5 9
1 32 1 9           1 32 9 8 7 6 9
1 33 4 9           1 33 5 3 1 0 9
1 34 1 9           1 34 8 6 4 2 9
..                      ..
```

Figure 7.4: Sentence-losses for source sentence 1, hypotheses 31-34, left: TER and reference length, right: n-gram matches and reference length

feature is present in the good hypothesis (oracle), its weight is increased. If the feature is present only in the bad hypothesis, its weight is decreased. In this fashion we iterate over the complete training set, gradually increasing the number of features and sharpening their distribution. While training the DLM on n-best confusions, the model learns to differentiate between oracle translations and other candidate translations. We can then apply that model to rerank n-best lists based on a combination of the baseline score and the DLM score.

7.3.1 Objective functions

Translation Edit Rate (TER) and BLEU score are commonly used metrics for evaluating the quality of machine translation output and both were used as objective functions for perceptron training. Because the perceptron updates are based on one input example (source sentence) at a time, the loss function needs to be evaluated separately for each input example. The BLEU score is a corpus-based metric and in order to evaluate it per sentence, we need to use a smoothed sentence-level version of BLEU. To avoid zero counts for n-gram precisions, n-gram matches are smoothed as in Papineni's implementation of BLEU[1] [55].

Since perceptron training is run on fixed n-best lists, the per-sentence losses can be computed in advance. The TER and BLEU losses are stored in separate files and can easily be substituted to switch between different objective functions. Figure 7.4 shows and example of TER and BLEU loss files.

7.3.2 Parameters

In addition to the simple perceptron training using the oracle and 1-best translation selected from the n-best list, some variations to the update and example selection were implemented. Instead of a perceptron update it is possible to select a MIRA-style update with a clipping parameter. MIRA seeks a minimum update to the weight vector such that model score difference of an oracle and a hypothesis translation reflects the loss between them as measured by an automatic error metric. The oracle can be chosen according to the minimal loss with respect to the reference or according to the maximum of (baseline score - loss) (direct loss). Likewise, the candidate to move away from can be chosen according to the maximum baseline score (1-best) or the maximum of (baseline score + loss) (chiang). Examples can be skipped when the loss of the oracle exceeds a predefined value (minimum bleu = 1 - maximum loss).

7.3.3 Experiments

In this section we present perceptron training results on an n-best reranking task for both language pairs. We do not report results of varying the baseline mixing factor when combining baseline and perceptron scores and the mixing factor was 1 for all experiments.

7.3.3.1 Urdu-English

Table 7.1 gives results for reranking the target sides of the Urdu↔English data. We compare perceptron models trained for both translation directions on one-way versus round-trip data, optimized towards BLEU versus TER and using four different update schemes, a simple perceptron update versus a MIRA update with 3 different values for clipping.

[1] ftp://jaguar.ncsl.nist.gov/mt/resources/bleu-1.04.pl

			Baseline	Perceptron	MIRA clip=0.1	MIRA clip=1	MIRA clip=10
ur→en	one-way	↑BLEU	22.61	22.10	**22.82**	22.28	22.10
	round-trip			22.22	22.52	22.51	22.40
	one-way	↓TER	65.96	**65.13**	**65.52**	**65.55**	**65.74**
	round-trip			**65.69**	**65.96**	**65.73**	**65.82**
en→ur	one-way	↑BLEU	13.77	13.34	13.30	13.38	13.32
	round-trip			13.35	13.49	13.67	13.66
	one-way	↓TER	73.68	**72.06**	**72.10**	**71.76**	**71.93**
	round-trip			**72.26**	**72.13**	**71.89**	**71.83**

Table 7.1: Perceptrons optimized towards BLEU (top) and TER (bottom), Urdu↔English, BLEU/TER scores on heldout set (3gram features)

We can state in general that there are almost no improvements when optimizing the perceptron towards BLEU while we see similar or better results for all experiments optimizing towards TER. This could either mean that TER is more sensitive towards the differences caused by reranking. Or, because the baseline models were originally tuned towards BLEU, we might just be observing a slight shift away from the best BLEU translations and towards the best TER translations that might even be independent of the new features. However, on the English→Urdu task the improvement is about 2 points of TER, which might indicate a real improvement. For both translation directions, the one-way data seems to produce slightly better results when optimizing towards TER and the round-trip data seems slightly better when optimizing towards BLEU. In general, the MIRA updates yielded better results than the simple perceptron update. There are some exceptions though, for example, when optimizing towards TER for Urdu↔English the perceptron update does better for both one-way and round-trip data. There was also some variation in the results for different clipping values and a value of 0.1 performed well in many cases. However, the best results on TER were achieved with clipping values of 1 and 10.

Results of perceptron training for Urdu→English with the different candidate selection methods mentioned in the last section are shown in table 7.2. Even though there is no real telling pattern, it seems that restricting the maximum loss of an oracle translation works better than the other selection methods for both one-way and round-trip data. Still, the results are all below the baseline of 22.61 BLEU.

In table 7.3 we compare perceptron experiments for Urdu→English with word 3-grams (word3), additional part-of-speech n-grams (word2+pos3,
word3+pos3) and added interleaved (PwP, wPw) word/POS n-grams
(word3+pos3+inter3). All experiments were carried out with a MIRA update and clipping of 1. In many cases, adding part-of-speech features improves over word n-gram features, and this is true for both one-way and round-trip data and both training objectives. But in most cases, the interleaved features do not seem to yield additional improvements on top of the POS features. Also, when adding POS features it seem to be sufficient to have word 2-grams instead of word 3-grams. For the monolingual data (only 20K sentences), we do not see an improvement with added POS features but this might be due to noise in this rather small data set. Still, figure 7.5 shows that the monolingual data yields comparable results than the parallel data and therefore our experiments should be repeated with larger amounts of monolingual data.

7.3.3.2 English-German

For this language pair, we are leaving out the results of experiments optimizing towards TER because the results were similar to the English↔Urdu experiments. Perceptrons trained to optimize TER improved over the baseline while perceptrons optimized towards BLEU did not. Figure 7.6 shows our results for reranking one fold of the English→German n-best lists, with three different sets of features (word n-grams, additional

Baseline=22.61	dl	chiang	max-loss	dl+chiang	dl+max-loss	dl+chiang+max-loss
one-way	22.05	22.13	*22.50*	22.05	22.34	21.95
round-trip	21.90	22.20	*22.25*	22.10	21.50	21.87

Table 7.2: Candidate selection methods on Urdu→English, BLEU scores on heldout set (3gram features, MIRA update with clip=5, dl=direct loss, max-loss was set to 0.9)

		word3	word2+pos3	word3+pos3	word3+pos3+inter3
one-way	BLEU	22.28	22.24	**22.35**	**22.46**
round-trip		22.51	**22.76**	**22.60**	22.32
monolingual		22.42	n/a	22.41	22.06
one-way	TER	65.55	**65.08**	65.58	65.79
round-trip		65.73	**65.10**	**65.26**	**65.38**
monolingual		65.70	n/a	65.62	67.01

Table 7.3: Perceptrons optimized towards BLEU (top) and TER (bottom), English↔Urdu, BLEU/TER scores on heldout set, comparing different feature sets

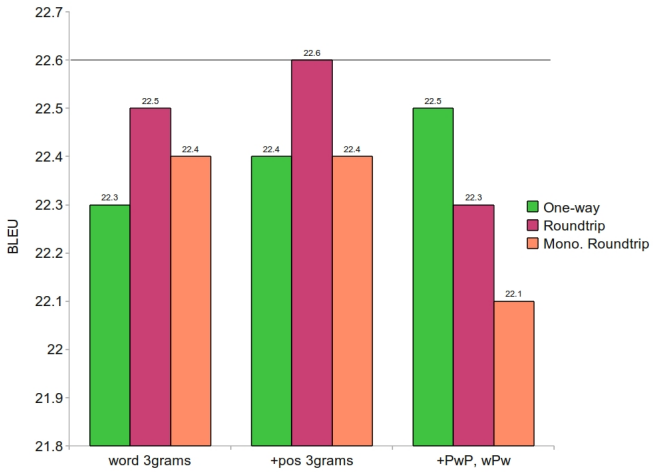


Figure 7.5: Perceptron optimized towards BLEU (ur-en), one-way/roundtrip: 88K, Mono. round-trip: 20K

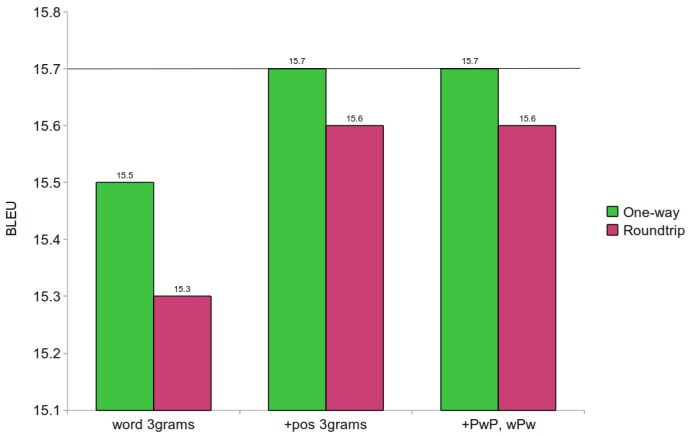


Figure 7.6: Perceptron optimized towards BLEU (en-de), one-way/round-trip: 180K (one fold)

POS n-grams and additional interleaved word/POS n-grams). All results stay below the baseline, but we see improvements for richer feature sets and a similar trend for one-way and round-trip data. Experiments with additional monolingual data are left for future work.

7.4 Retuning with discriminative LM

We have seen in the last section that we do not get the desired results with the setup of our reranking experiments. Therefore, the next idea was to run Minimum Error Rate Training on the baseline system together with the DLM and hence let MERT figure out how much weight should be given to the DLM. This results in a 2-step training process: Perceptron(dlm) + MERT(baseline+dlm). In standard MERT, a tuning set is decoded in every iteration and the model parameters are optimized on the resulting n-best list, until convergence. We modify this process slightly by including a step where discriminative features are computed on the n-best list and the new n-best is scored with the DLM. The DLM score is added in an additional column and the weighted model score in the last column is recomputed with the current weights:

```
882 ||| there is good satire ! ||| lm: -14.1295 tm: -11.7801 -10.3322
-8.43544 -7.77404 4.99948 1.99979 w: -3.04006 ||| -2.96601
882 ||| there is good satire ! ||| lm: -14.1295 tm: -11.7801 -10.3322
-8.43544 -7.77404 4.99948 1.99979 w: -3.04006 dlm: 0.567937 |||
 -2.89916
```

The MERT optimizer is run to optimize all baseline weights plus an additional DLM weight. However, when computing a new n-best list, the decoder is run with only the baseline features, since we did not integrate the DLM into Moses.

7.4.1 Experiments

7.4.1.1 Urdu-English

Table 7.4 shows results with retuned weights, before and after applying the DLM to rerank the n-best output. All results were averaged over three MERT runs to account for optimizer instabilities and in fact the variance between different runs was around 0.5 BLEU in some cases. For some single runs, the results were better before reranking, which should be investigated. Overall, however, we achieved good improvements over the baseline and for 3-gram features we get especially interesting results with the Pivot data. Additional features

Baseline=22.61	Feature set	1-best	Reranked 1-best
one-way.BLEU	word3	22.91	22.91
one-way.TER		22.84	**23.03**
round-trip.BLEU		22.49	**22.77**
round-trip.TER		22.79	**23.07**
mono.BLEU		22.73	**22.83**
mono.TER		22.87	22.88
Pivot.BLEU		22.99	**23.26**
Pivot.TER		22.68	**23.04**
one-way.BLEU	+pos3	22.72	**22.86**
one-way.TER		22.47	**22.98**
roundtrip.BLEU		22.83	**23.15**
roundtrip.TER		22.70	**23.11**
one-way.BLEU	+inter3	22.81	**23.17**
one-way.TER		22.83	**23.10**
roundtrip.BLEU		22.71	**22.95**
roundtrip.TER		23.04	**23.24**

Table 7.4: Results for Urdu→English with weights from MERT retuning, with perceptrons optimized towards TER and BLEU, MIRA updates and three different feature sets

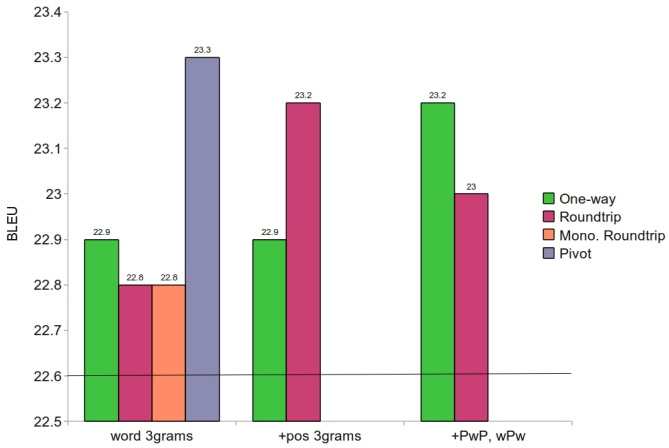


Figure 7.7: Retuning baseline + DLM (ur-en), results averaged over 3 MERT runs per experiment

improve over simple n-gram features in many cases, although not always to the extent that we would expect. Results are printed in bold when they improve both over the baseline and over the 1-best non-reranked result.

Figure 7.7 shows a graphical representation of our retuning results. Results for one-way data are sometimes better and sometimes worse than results for round-trip data, so on average they are comparable. We have only few data points for monolingual round-trip data so far, but considering that the data set was much smaller it looks on par with the one-way and round-trip data.

7.4.1.2 English-German

Table 7.5 shows results with retuned weights for English→German. Even though the improvements are in general smaller than for the Urdu→English experiments, we do get some improvements over the baseline when adding POS features on top of the n-gram features. Again, results are printed in bold when they improve both over the baseline and over the 1-best non-reranked result. For this language pair, there are more cases where the reranked results do not improve over the 1-best results with retuned weights, even if the final results are better than the baseline. This is non-intuitive and should be investigated.

Figure 7.8 shows a graphical representation of our retuning results for English→German. We beat the

Baseline=15.67	Feature set	1-best	Reranked 1-best
one-way.BLEU	word3	15.53	15.61
one-way.TER		14.12	15.05
round-trip.BLEU		15.03	15.48
round-trip.TER		15.28	15.62
one-way.BLEU	+pos3	15.19	**15.70**
one-way.TER		15.89	15.85
roundtrip.BLEU		15.73	15.71
roundtrip.TER		15.73	15.53
one-way.BLEU	+inter3	15.52	15.60
one-way.TER		15.61	**15.69**
roundtrip.BLEU		15.42	**15.72**
roundtrip.TER		15.71	**15.81**

Table 7.5: Results for English→German with weights from MERT retuning, with perceptrons optimized towards TER and BLEU, MIRA updates and three different feature sets

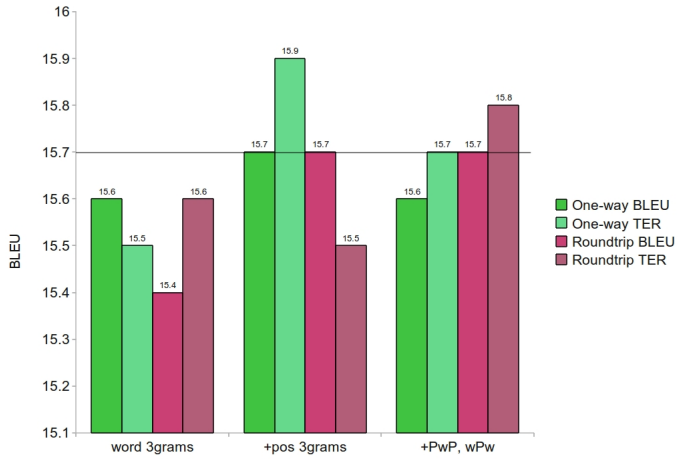


Figure 7.8: Retuning baseline + DLM (en-de), results averaged over 3 MERT runs per experiment

baseline only in few cases and not by much, but the results still show that we can get comparable performance for real and hallucinated data, which is an important finding for future work.

7.5 Minimum Risk Discriminative Training from MT Hallucinations

Choosing the expected loss or the risk as the objective function to minimize is a general framework for training. For example, several popular training methods([9],[14],[56]) can be formalized in this framework by choosing different types of loss functions. In the CLSP summer workshop 2011, other than the popular perceptron algorithm, we also used the minimum risk discriminative training algorithm developed by ([21]) for discriminative language model training, and made comparison between the two methods. Here we briefly describe the rationale of minimum risk training, more details can be found in ([21]). We consider the supervised case here first, just to illustrate how this algorithm works, modifications in the unsupervised case will be described in section 7.5.1.

Let x be any source sentence, and y be the corresponding target reference sentence. Given a translation system controlled by parameter set $\mathbf{\Theta}$, denote $\delta_{\mathbf{\Theta}}(x)$ to be the translation of x. Define a loss function $L(\delta_{\mathbf{\Theta}}(x), y)$ to measure how bad the translation is, then the goal of minimum risk training is to find the

optimal $\mathbf{\Theta}^*$ such that the Bayes risk is minimized:

$$\mathbf{\Theta}^* = \arg\min_{\mathbf{\Theta}} \sum_{x,y} p(x,y) L(\delta_{\mathbf{\Theta}}(x), y) \tag{7.1}$$

In reality, it is impossible to enumerate all possible translation pairs (x, y), therefore the true distribution $p(x, y)$ is unknown, and we must replace the true risk with the empirical risk. Suppose we are given a parallel development corpus $\mathcal{C} = \{(x_i, y_i), i = 1 \dots N\}$, in which x_i is the source sentence, y_i is the target reference sentence and N is the size of the corpus. Now the minimum risk training becomes

$$\mathbf{\Theta}^* = \arg\min_{\mathbf{\Theta}} \frac{1}{N} \sum_{i=1}^{N} L(\delta_{\mathbf{\Theta}}(x_i), y_i) \tag{7.2}$$

Usually, $\delta_{\mathbf{\Theta}}(x_i)$ is chosen to be the most likely output from the translation system, namely $\delta_{\mathbf{\Theta}}(x_i) = \arg\max_y p_{\mathbf{\Theta}}(y|x)$. However by doing so would make the objective function not differentiable. To overcome this difficulty, we relax the loss in (7.2) with the expectation of the loss, namely

$$\begin{aligned} \mathbf{\Theta}^* &= \arg\min_{\mathbf{\Theta}} \frac{1}{N} \sum_{i=1}^{N} E\left[L(\delta_{\mathbf{\Theta}}(x_i), y_i)\right] \\ &= \arg\min_{\mathbf{\Theta}} \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{y \in T(x_i)} p_{\mathbf{\Theta}}(y|x_i) L(y, y_i) \right) \end{aligned} \tag{7.3}$$

in which $T(x_i)$ is the set of all the possible translations of x_i generated by the translation system.

We used the open source translation software Joshua([31]) in all our experiments in which the Hiero translation model([57]) is implemented. For each source sentence x, the Hiero model generates a set of derivations which we denote by $D(x)$. We use the log-linear model to represent the probability of each derivation as the following:

$$p_{\lambda,\gamma}(d|x) = \frac{1}{Z(x)} \exp\left(\sum_{m=1}^{M} \lambda_m \psi_m(d) + \sum_{n=1}^{N} \gamma_n \phi_n(d) \right), \quad d \in D(x) \tag{7.4}$$

in which $\psi_m(d)$ and $\phi_n(d)$ are the m^th baseline feature and n^th discriminative language model feature on derivation d respectively, λ_m and λ_n are the m^th baseline feature and n^th discriminative feature weights respectively, M and N are the number of baseline features and discriminative language model features respectively, $Z(x) = \sum_{d' \in D(X)} \exp\left(\sum_{m=1}^{M} \lambda_m \psi_m(d') + \sum_{n=1}^{N} \gamma_n \phi_n(d') \right)$ is the normalization term. Since each derivation yields a deterministic output translation $y(d)$, we rewrite (7.3) as

$$\begin{aligned} \mathbf{\Theta}^* = \arg\min_{\mathbf{\Theta}} \frac{1}{N} \sum_{i=1}^{N} \sum_{d \in D(x_i)} \\ \frac{1}{Z(x_i)} \exp\left(\sum_{m=1}^{M} \lambda_m \psi_m(d) + \sum_{n=1}^{N} \gamma_n \ \phi_n(d) \right) L(y(d), y_i) \end{aligned} \tag{7.5}$$

In practice, we choose the negated BLEU score as the loss function, and the expectation of the loss is computed via semiring([58]) on the hypergraph. Optimization of the objective function is realized using gradient-based algorithm LBFGS.

7.5.1 Hallucinated Data Generation and Unsupervised Training

The minimum risk training introduced in section 7.5 is based on the assumption that the parallel development corpus is available. However it is usually too expensive to acquire large amount of parallel training data, and we hope to hallucinate source sentences from target side sentences which is widely available to inflate the training corpus.

Two methods for hallucinated data generation we explored are based on reverse translation and monolingual confusion grammar generation. The reverse translation method([20]) makes use of an existing reverse translation system from the target to the source side, translate the target side references and treat the outputs as the source input sentences. In this way we hallucinate a bilingual "parallel" training corpus which can then be used for discriminative training. This process actually completes a "round-trip" translation from target to target.

The monolingual confusion grammar generation method([59],[60],[20]) derives a synchronous monolingual grammar rule set from a synchronous bilingual grammar rule set. For example, if we have two synchronous bilingual grammar rules

$$X \quad \rightarrow \quad < f_1, e_1 > \qquad X \quad \rightarrow \quad < f_2, e_2 >$$

in which f and e are the foreign and target side rules respectively, then we can derive the following target side monolingual grammar rules by using f as pivot:

$$\begin{array}{llll} X & \rightarrow & < e_1, e_1 > & X \quad \rightarrow \quad < e_1, e_2 > \\ X & \rightarrow & < e_2, e_1 > & X \quad \rightarrow \quad < e_2, e_2 > \end{array}$$

Once we derive a whole set of monolingual grammar rules, we can build a target-to-target translation system for discriminative training, and all that we need is just the target side monolingual corpus. Of course, to derive the monolingual rules, we first need a small set of parallel corpus to extract the set of bilingual rules.

Now that we are able to hallucinate a "parallel" corpus, we can do discriminative training in an unsupervised way. Since the hallucinated data may not consistently have high quality, we hope that the training starting only from the target side corpus will have a low "round-trip" loss. Thus equation (7.2) is modified as

$$\begin{aligned} \mathbf{\Theta}^* &= \arg\min_{\mathbf{\Theta}} \frac{1}{N} \sum_{i=1}^{N} L(\delta_{\mathbf{\Theta}}(x_i), y_i) \\ &= \arg\min_{\mathbf{\Theta}} \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{x \in G(y_i)} p_{\mathbf{\Omega}}(x|y_i) L(\delta_{\mathbf{\Theta}}(x, y_i) \right) \end{aligned} \tag{7.6}$$

in which $p_{\mathbf{\Omega}}(x|y_i)$ gives the probability of translating the target side sentence y_i to the foreign side sentence x by a reverse translation system controlled by parameter Ω, and $G(y_i)$ is all the possible reverse translations. However in practice it is intractable to compute $p_{\mathbf{\Omega}}(x|y_i)$ for each $x \in G(y_i)$, and an approximation is to use the k-best from $G(y_i)$ and renormalize the probability([21]). In our experiments, we only used 1-best reverse translation since it is fast and using k-best is reported not to be very helpful in improving the performance([21]). That is to say, in our case $G(y_i) = \{x_{1-best}\}$ and $p_{\mathbf{\Omega}}(x_{1-best}|y_i)$ is set to 1.

7.5.2 Experimental Results

7.5.2.1 Training Methods

In our experiments, we used minimum risk as well as average perceptron for training. In minimum risk training, since we can optimize the objective function as given in (7.5) by computing the gradient with respect to each feature weight(for both baseline and discriminative features), we are able to determine the baseline and discriminative feature weights at the same time. However to use perceptron for discriminative training, we need to tune the baseline feature weights first(using MERT for example) in order to generate

the n-best lists for perceptron training, then fix the discriminative weights and re-tune the baseline weights. That is to say, to use perceptron for training the discriminative and baseline weights are determined in two steps.

7.5.2.2 Feature Selection

In our experiments the following features are used in our model:

1. Baseline features: regular Hiero features([57]) including the language model feature, three translation model features, three arity count features, two glue rule count features, and a word penalty feature.

2. Discriminative language model features: we used the target-rule bigram features([21]) as discriminative language model features. That is, on the target side of the Hiero synchronous rules, we replace the terminals with their dominant POS tags, then extract the most frequent bigram rules as the features. On average, we extract around 1000 features from the development set grammar.

7.5.2.3 Experimental Setup

We did our experiments on both Urdu-English and Chinese-English language pairs, each language pair includes a parallel training corpus and a parallel tuning set. The target language is English. The Urdu-English training corpus and tuning set contains 87711 and 980 sentences respectively, the Chinese-English training corpus and tuning set contains 2116920 and 4089 sentences respectively. The training corpora are divided into 10 slices with equal size. The evaluation set for Urdu-English and Chinese-English task has 883 and 1082 sentences respectively. The data usage for the two types of experiments corresponding to the two unsupervised training methods introduced in section 7.5.1 is as the following:

1. "Round trip" translation: we extracted the last 200 and 880 English sentences from each of the slices for both Urdu-English and Chinese-English task as the monolingual data(other numbers of sentences are also tried in some experiments but generated in the same way), and we use the tuning set as the bilingual data. The reverse translation system uses MERT for parameter tuning on the tuning set.

2. Monolingual confusion grammar generation: the confusion grammar is extracted from the combination of the tuning and evaluation set, and we still use the 2000 and 8800 sentences extracted from the slices as the monolingual data. These monolingual data is used only to tune the discriminative language model feature weights, and after the weights have been computed, we fix the discriminative feature weights and tune the baseline weights on the bilingual tuning set.

7.5.2.4 Results

Following the experimental setups introduced in the previous section, we did our experiments in supervised, semi-supervised and unsupervised settings. For Urdu-English task, table 7.6, 7.7 show the experimental results based on “round-trip” translation and confusion grammar generation using minimum risk training respectively, and table 7.8, 7.9 show the results using perceptron training. For Chinese-English task, table 7.10, 7.11 show the results based on “round-trip” translation and confusion grammar generation using minimum risk respectively, and table 7.12, 7.13 show the results using perceptron.

A few notes to the table:

1. The scores on the table are the BLEU scores on the evaluation set.

2. The number in the parenthesis stands for the number of training sentences. “semisup(x+y)” means the training data is composed of x bilingual sentence pairs and y hallucinated sentence pairs, “DLM” means discriminative language model features.

3. Since in minimum risk training we do not need to first tune and fix the baseline feature weights to generate the n-best list, the baseline weights are tuned on each given data set specifically, thus giving different baseline scores for each setting. However in perceptron training we need to fix a baseline system for the n-best list generation, thus the baseline score is consistent for all settings.

	Baseline	Baseline+DLM
sup(980)	**19.66**	19.64
semisup(980+2000)	18.24	**19.02**
semisup(980+8800)	18.34	**18.47**
unsup(2000)	18.98	**19.00**
unsup(8800)	**19.15**	18.84

Table 7.6: Urdu-English "round-trip" translation training results using minimum risk

	BLEU
baseline(980)	19.38
baseline+DLM(980+2000)	**19.53**
baseline+DLM(980+8800)	19.25

Table 7.7: Urdu-English confusion grammar training results using minimum risk

	Baseline+DLM
sup(1000)	19.13
unsup(1000)	19.17
semisup(1000+2000)	19.43
semisup(1000+8800)	19.46
unsup(2000)	19.37
sup(2000)	19.28
unsup(8800)	19.54
sup(8800)	19.50
unsup(87711)	**19.56**
sup(87711)	19.28

Table 7.8: Urdu-English "round-trip" training results using perceptron(baseline score is 18.67)

	Baseline+DLM
semisup(980+1000)	19.44
semisup(980+2000)	19.43
semisup(980+5000)	**19.66**
semisup(980+8000)	19.57

Table 7.9: Urdu-English confusion grammar training results using perceptron(baseline score is 18.67)

7.5.2.5 Observations and Analyses

From the results given in 7.5.2.4 and the process of our experiments, we have the following observations:

1. By incorporating the discriminative language models, the performance gets improved compared with the baseline system in general.

	Baseline	Baseline+DLM
sup(4089)	**28.60**	28.48
semisup(4089+2000)	28.06	**28.21**
semisup(4089+8800)	28.03	**28.28**
unsup(2000)	26.79	**27.94**
unsup(8800)	27.38	**27.42**

Table 7.10: Chinese-English "round-trip" training results using minimum risk

	BLEU
baseline(4089)	
baseline+DLM(4089+2000)	
baseline+DLM(4089+8800)	

Table 7.11: Chinese-English confusion grammar training results using minimum risk

	Baseline+DLM
sup(4000)	28.00
unsup(4000)	27.94
semisup(4000+2000)	28.02
semisup(4000+8800)	28.12
sup(2000)	27.99
unsup(2000)	**28.23**
sup(8800)	28.02
unsup(8800)	27.87

Table 7.12: Chinese-English "round-trip" training results using perceptron(baseline score is 25.00)

	Baseline+DLM
semisup(4089+1000)	
semisup(4089+2000)	
semisup(4089+5000)	
semisup(4089+8000)	

Table 7.13: Chinese-English confusion grammar training results using perceptron(baseline score is 25.00)

2. For the same amount of training data, the average performance given by the "rount-trip" and confusion grammar generation method is about the same.

3. Though increasing the amount of training data helps to improve the performance in many cases, it is not guaranteed that the performance gets improved monotonically with the increase of data size. For example, in table 7.6 and 7.7, the results using 8800 hallucinated sentences did not outperform those using 2000 hallucinated sentences.

4. The perceptron algorithm in general gives better performance than minimum risk, with the same amount of training data.

5. The training process of minimum risk is very slow. For example, given 12889 sentences as in the Chinese-English semi-supervised setting, it could take eight or more hours to finish one iteration, and usually we need four or more iterations to reach some mature results. This is because two types of iterations are involved in the minimum risk training. The first is the LBFGS internal iterations, but since we need to prune the hypergraph for practical decoding(otherwise there would be too many derivations making the decoding intractable), we need another iteration to check the convergence of the n-best list to reach reliable results. What is more, to compute the expectation of loss using semiring on each hypergraph also adds to the computational complexity.

By comparison, the perceptron training is much faster due to the simplicity of the algorithm and the fact that no decoding is required on the fly(the n-best list is generated by the baseline system beforehand). It took only less than an hour to finish training on the entire Urdu-English corpus(87711 sentences), which would probability take a week for the minimum risk algorithm to finish(and that is why we did not try large datasets for the minimum risk method). Therefore the perceptron is much more scalable on large datasets than minimum risk.

6. Contrary to our intuition, in table 7.6,7.10 the semi-supervised setting(with more data) did not outperform the supervised case, and we found out that this is due to the low quality of the hallucinated data. For example, if we replace the hallucinated 2000 sentences with the genuine sentences for the semi-supervised setting(4089+2000) in table 7.10, the BLEU scores become:

	Baseline	Baseline+DLM
sup(4089+2000)	28.72	28.75

which exceeded the supervised case with 4089 sentences. Therefore the quality of the hallucinated data has a significant impact on the performance, and we would like raise the data quality.

In all our experiments introduced above, the hallucinated data was generated by the reverse baseline translation system tuned using MERT. Although a classical method, it seems that the quality of the hallucinated data generated by MERT was not satisfying yet, and we wanted to see if minimum risk can do better than MERT on tuning the baseline systems. To do so, we tuned the Chinese-English forward system using both MERT and minimum risk, by using both 5 and 10 baseline features, and table 7.14 show the baseline system performance on the evaluation set:

	MERT	Min-Risk
5 features	25.00	**28.07**
10 features	22.04	**28.60**

Table 7.14: Comparison between the baseline systems tuned by MERT and minimum risk

It can be seen that minimum risk reached much better performance on the baseline system than MERT,

and we expect to hallucinate better training sentences by the stronger baseline. Hence we ran the Chinese-English "round-trip" experiments again using data generated by the baseline tuned using minimum risk, and the new results are given in table 7.15 and 7.16.

	Baseline	Baseline+DLM
semisup(4089+2000*)	28.45	**28.62**
semisup(4089+8800*)	27.99	28.39
unsup(2000*)	27.98	**28.21**
unsup(8800*)	28.06	**28.08**

Table 7.15: Chinese-English "round-trip" training results using minimum risk with newly hallucinated data(marked by *)

	Baseline+DLM
semisup(4000+*2000)	27.94
semisup(4000+*8800)	27.98
unsup(*2000)	28.08
unsup(*8800)	27.88

Table 7.16: Chinese-English "round-trip" training results using perceptron with newly hallucinated data(marked by *)

7.6 Conclusions and future work

We have seen in the reported experiments that perceptron training alone yielded no gains in BLEU score on the reranking task and hence the retuning step was essential. Perceptron models optimized towards both objectives (TER, BLEU) yielded gains with retuning, which is in a way non-intuitive. In many cases, we were able to get improvements when adding POS n-gram features on top of word n-gram features.

An important outcome of the experiments was the confirmation that we can achieve matching performance with hallucinated n-best lists. We are therefore confident that we will be able to apply large amounts of monolingual data to create more discriminative training data. We got interesting results with confusion grammars learned from parallel corpora which provided a fast way of generating hallucinated n-best lists. We also got encouraging results with some first (rather small-scale) experiments on monolingual round-trip data and we expect to see larger gains with more data.

We also explored minimum risk discriminative training in comparison with the average perceptron on both Urdu-English and Chinese-English task. Experimental results show that the discriminative language model in general helps to improve the system performance compared with the baseline. The results given by the minimum risk and perceptron algorithm are comparable to each other, but the former is much more expensive to train and is not suitable for scaling. However minimum risk reached much better performance than MERT on tuning the baseline system, generating hallucinated data with higher quality which improved the training performance.

In the future, we would like to scale up training to the full data sets that were created during this workshop. Especially, we would like to fully explore the potential of the Pivot method (confusion grammar) for creating training data as well as the data created from monolingual corpora. An interesting question to answer is whether we get different results with the Pivot data, with and without removing the intersection, since this might give insights on the role of paraphrases in discriminative training. We have seen some improvements when optimizing towards TER and we could try using a linear combination of the TER and BLEU metrics for training. Finally, the features used within the discriminative language model should be integrated with the decoder in order to use them directly during search.

Chapter 8

ASR Hallucinations

The main results of this chapter involve controlled experimentation under a number of conditions, to determine how much WER reduction can be achieved with these semi-supervised methods, versus fully supervised discriminative language model training on the same data. We carefully prepared a large discriminative language modeling corpus, and examined performance when training models on real n-best lists produced under standard conditions versus when training models on hallucinated n-best lists produced under various protocols for producing such lists. We find that methods for direct extraction of phrasal cohorts yield the largest WER reductions of our three examined methods, just over half the gain achieved with fully supervised methods, and that improvements can also be achieved by modeling confusability at the phone level. Given that these semi-supervised methods can be applied to arbitrarily large text corpora, this bodes well for additional system gains beyond what was achieved for these moderate sized training sets.

8.1 Data

For this paper, we focused on English conversational telephone speech (CTS), and used the IBM Attila speech recognition software library [49] to build a baseline system. The training data for acoustic models consists of about 2000 hours of speech segments from about 14500 telephone conversations, of which about 11000 conversations are from the Fisher corpus and 3500 from the Switchboard corpus. We utilized the NIST RT04 Fall development and evaluation test for benchmarking the performance of our systems. The development and evaluation sets consist of about 38K words and 37K words respectively, spanning about 2 hours.

The acoustic models contain 41 phones, a 3-state left-to-right HMM topology for phones, 4K clustered quinphone states, 150K Gaussians, a linear discriminant transform, and a semi-tied covariance transform. The acoustic features consist of 13 coefficient perceptual linear coding vectors with speaker-specific vocal tract length normalization (VTLN). This training approach is as described in [61], and we refer readers there for further details. The baseline 4-gram language model defined over 50K word vocabulary was estimated by interpolating the transcripts and similar data extracted from the web as described in [48].

In order to produce training data for discriminative language modeling, we decoded the training data using 20-fold cross validation. That is, while decoding a fold, we ensured that the transcripts from that fold were kept aside in estimating the language model used for decoding the fold. All three stages of the decoding were performed in this 20-fold cross-validation manner. Thus, we took more care to avoid biasing the decoder than the standard process of employing 20-fold cross-validation just in the last stage of decoding.

For the controlled studies of the current paper, we used up to eight of these folds to train our discriminative language models. In fully supervised conditions, the lattices output by the above cross-validation scenario were used to produce the n-best lists directly. To compare with these fully supervised conditions, we generated hallucinated n-best lists for the same utterances. To produce these simulated n-best lists, we followed another cross-validation procedure: For each of the k folds in the given training set, we trained confusion models (using one of our three methods detailed below) on the other $k-1$ folds, and used them to produce n-best lists for that fold. In such a way we produce alternative n-best lists for the same data as in the fully supervised scenario, to provide to the discriminative training algorithm. In the results presented below, we investigated

conditions with 2, 4 and 8 folds.

8.2 Generating confusions

In this section, we describe a variety of methods that we explored during development at the six week summer workshop. The most promising methods were then validated in a set of controlled experiments that will be presented in the following section.

8.2.1 Linguistically motivated phone confusions

Using the phoneme class lists that were included with the baseline ASR systems, we constructed feature vectors for each phone that were then used to calculate confusability. The class files are structured as follows:

Vowel	AA AE AH AO AW AX AY EH ER EY IH IY OW OY UH UW
Sonorant	AA AE AH AO AW AX AY EH ER EY IH IY L M N NG OW OY R W UH UW Y
Continuant	AA AE AH AO AW AX AY DH EH ER EY F HH IH IY L NG OW OY R S SH TH UH UW V W Y Z ZH
Nasal	M N NG
Retroflex	R ER
Lateral	L

Each class (vowel, continuant, etc.) is taken to be a single feature, and each phoneme is assigned a vector indicating its class memberships. The vector consists of + and -, minus indicating that the phoneme belonged to a particular class, resulting in structures as follows:

```
 S   ----------+-+--++-----------
 T   ------------+-----------+---
UW   ---+----+----+-+-+--------++
 N   ---+---+----+---------------
```

Given such data structures, there are many ways to calculate similarity and confusability. In a preliminary investigation, we generated confusions based on how many of the same classes were assigned + for two phonemes. Alternatively, we could have defined similarity based on both the - and + symbols, although the vectors seem to be sparser in +, leading to less spurious overlap.

This approach has its pros-and-cons. On the positive side, this can be applied for any of the baseline ASR system, since it relies on the existing phone class definitions. Several vowel classes are naturally distant from consonant classes, and many natural confusions fall out of the approach, e.g., s↔z and th↔dh. However, schwa has it's own class, the reduced vowel, which makes it very dissimilar to everything else, which is contrary to what would be expected. Rather than pursue this approach to the point where it would become a viable alternative, we focused instead on data driven methods for deriving confusions.

8.2.2 Low level confusion models

The aim of the low level confusion models is to capture acoustic and phonetic confusions from the Automatic Speech Recognizer (ASR). The confusions can be inherent in the ASR itself and/or due to pronunciation variabilities that occur from training and test data mismatch, and/or speaker particularities. As our current work has been employed with training and test data from the same domain, we have focused on capturing the ASR specific confusions. Our low level confusion model considers the acoustic confusions that come from the difficulty in discriminating phones that overlap in their spectral distributions, and the phonetic confusions derived from the simplistic configuration of the pronunciation lexicon, which is far from accurately capturing the pronunciation variability of the English conversational speech, as reported in [62].

8.2.2.1 CI phone confusion model

There are several previous studies on predicting and simulating the errors of the ASR. Many of them model confusions at the Context Independent (CI) phone level. Jyothi and Fosler combine data-driven phonetic

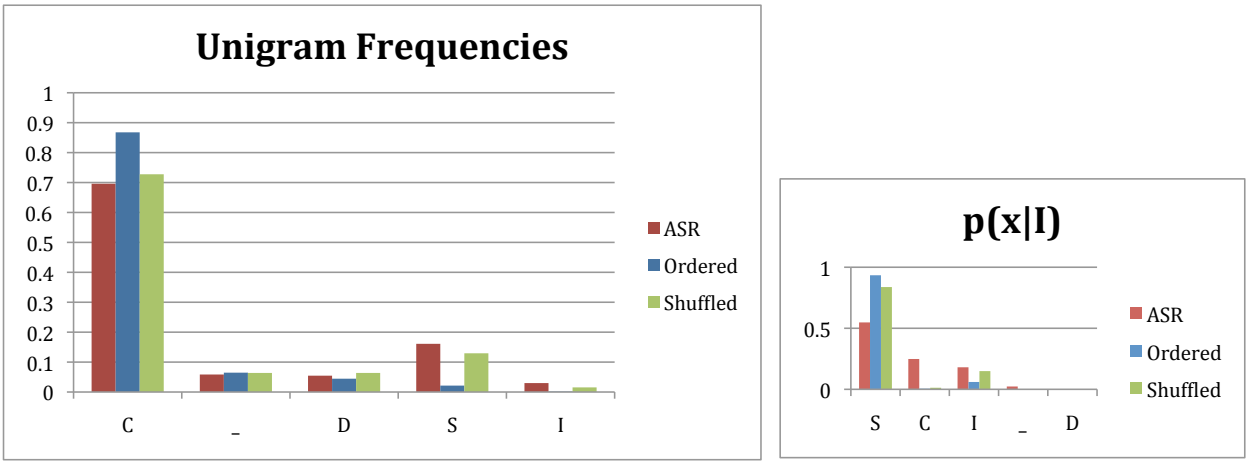


Figure 8.1: Patterns of correct (C), deletion (D), insertion (I), substitution (S) for the 1-best output of a) actual ASR; b) phone confusion generated lists; and c) randomly shuffled phone confusion lists. Underscore represents sentence boundary.

distances with phonetic distances based on the AMs [63, 22]. Kurata et. al. use a simple approximation of the Bhattacharyya distance to define the distance between phones [23, 24], and Hershey and Olsen propose different approximations to measure the distance between Gaussian Mixture Models (GMMs), and they also present an extension to measure the distance between Hidden Markov Models (HMMs) [64, 65].

Following this vein, we also built a CI phone confusion model. Using the EM algorithm proposed by Oncina and Sebban [66] we computed the conditional probabilities of the edit operations (insertions, deletions, and substitutions) between phones. The Expectation step accumulates the expected number of times an edit operation is used over the terminated edit sequences to map an input sequence X into an output sequence Y. The Maximization step is just a normalization step.

In some preliminary experiments we noticed that the distribution of the edit operations in the hallucinated nbest lists generated with the CI phone confusion model significantly diverged from the distribution shown by the ASR output. Figure 8.1 presents two representative graphs showing that the kinds of errors produced by the direct use of phone confusion models diverges from ASR. Interestingly, random shuffling of the output confusions does a better job of matching ASR error patterns. For this work, in order to keep the expected counts of insertions, deletions and substitutions constant over the EM iterations we introduced a marginalization step after the Expectation step. See Chapter 9 for another approach to this.

The resulting confusion model is a memoryless single state Finite State Transducer (FST) with the arcs representing the edit operations, as shown in Fig. 8.2.

8.2.2.2 Experimental Results

We used one of the folds from the Switchboard corpus to train the CI phone confusion models. The fold contains around 50,000 sentences. The training data consist of sentence pairs with each pair containing the reference phone sequence and 1-best phone sequence. The reference phone sequences were generated from the pronunciation lexicon, and the 1-best phone sequences from the pronunciation lexicon as well as from the lattices that the ASR outputs. We also analyzed another alternative; generating the phone confusions using local alignments instead of global alignments. We aligned word pairs instead of whole sentence pairs.

We picked a different fold from Switchboard, with similar number of sentences, to generate the hallucinated nbest lists (specifically 300best lists) that were later used to train the Discriminative Language Model (DLM). Fig 8.3 illustrates the pipeline to generate the hallucinated nbest lists. The input word sequence is composed with the pronunciation lexicon from the training step of the ASR, the output is composed with the confusion model generating a lattice of phones, this lattice is composed with the pronunciation lexicon from the decoding step of the ASR, and finally the output word lattice is rescored with the generative LM. None of the DLM trained with the hallucinated nbest lists generated with the different CI phone confusion models

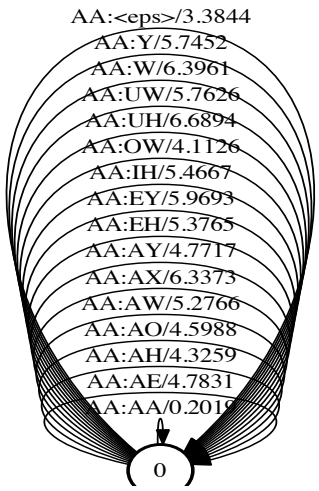


Figure 8.2: CI phone confusion model.

show any significant improvement over the baseline system. The CI phone confusion model generated from the local alignments showed a slight improvement, although it was insignificant. The CI phone confusion

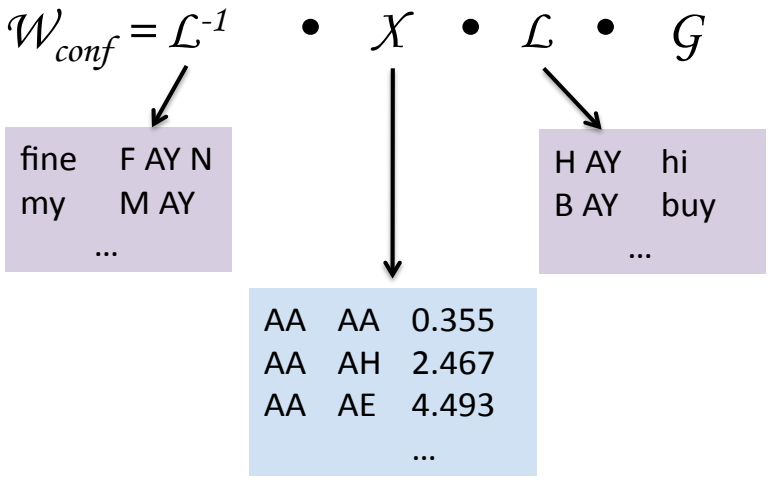


Figure 8.3: Pipeline for the generation of the hallucinated nbests with the CI phone confusion model.

model seems to be too coarse to accurately model the confusions from the English conversational ASR. In other languages where the correspondence between graphemes and phonemes is almost one-to-one, and there are not so many coarticulation effects, the CI phone confusion model may be sufficient, but in English the pronunciation of a phone is highly dependent on its neighboring phones, so a confusion model that captures the context should be more effective.

8.2.2.3 CD phone confusion model

The clustered or context dependent (CD) phones are the most common units modeled by the AMs. They represent the allophones of the phonemes; the different acoustic realizations of a phoneme given the context in which it appears. Therefore, defining a confusion model at clustered phone level may be a good solution to capture the phone confusions taking into account the context. The confusions between clustered phones come from the overlap of their spectral distributions as well as the confusions introduced by the pronunciation lexicon due to its poor modeling of the pronunciation variations that English conversational speech shows. Implicitly the AMs already capture the lack of accuracy of the pronunciation lexicon [62]. As a result the

observation or spectral distributions for the clustered phones will be broader, and they will be likely to present more overlapping between them. This overlap can be quantified measuring the distance between the spectral distributions, parameterized as GMMs, of clustered phones. Kullback-Leibler and Bhattacharyya distance are popular distance metrics to measure the similarity between a pair of probability distributions. For the particular case of GMMs there is not analytical closed-form solution for these distances, but several approximations have been proposed in the literature [64]. Our approach to build the clustered phone confusion model summarizes as follows. Clustered phones are modeled with a linear left-to-right 3 state topology with each state representing the spectral distribution of the initial, middle and final part of the clustered phone. As the most stable part of the sound unit is the middle one, we measured the similarity between clustered phones computing the Bhattacharyya distance between GMMs representing the middle state. For each clustered phone, only the top 20 confusions were included in the confusion model. The confusion distributions for the initial and final states were set to uniform distributions.

The pipeline to generate the hallucinated nbest lists is the composition of the input word sequence, the FST that inserts the word boundary marks between words (the clustered phones are modeled taking into account the word boundaries), the pronunciation lexicon used in the training of the AMs, the FST representation of the CD decision tree, the clustered phone confusion model, the inverted version of the CD decision tree, the pronunciation lexicon that the ASR uses during the decoding, the FST to delete the word boundary marks, and finally the LM, as shown in Fig 8.4.

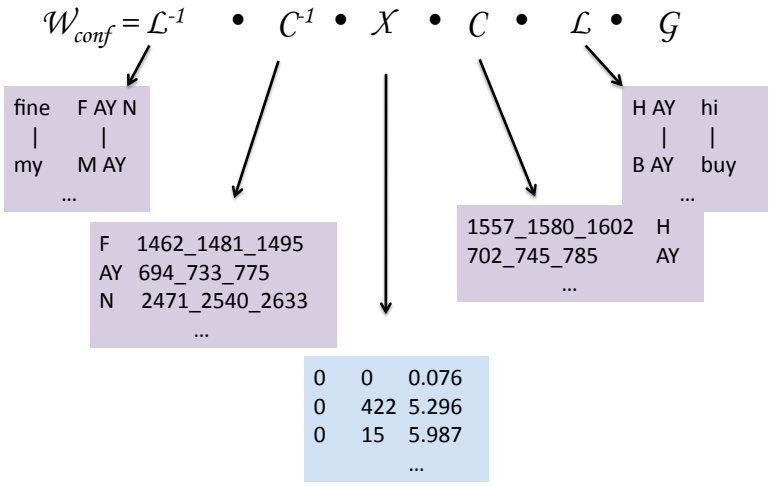


Figure 8.4: Pipeline for the generation of the hallucinated nbests using the CD phone confusion model.

8.2.2.4 Experimental Results

This approach was evaluated using the same data as in the CI phone confusion model experiments. As in the previous case we generated the hallucinated 300best lists for one of the Switchboard folds, and trained a DLM with the resulting 300best lists. Even if we are now capturing short duration context information we did not get any significant improvement over the performance of the baseline ASR.

8.2.2.5 Summary

In summary, the low level confusion models have not shown to be appropriate to generate the hallucinated nbest lists, at least for the English conversational ASR. In languages with one-to-one letter to phone mapping and with no so many coarticulation effects, or in domains where the speaking style of the speakers is not so variable, the CI or CD phone confusion models may work well, but for the English conversational domain looks not be enough to capture robust confusions. We may have to combine low level and high level confusion models.

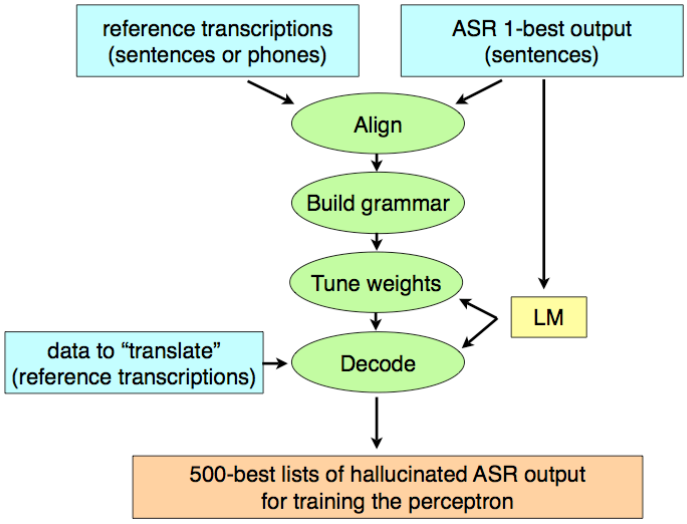


Figure 8.5: "MT" pipeline for hallucinating ASR n-best lists.

8.2.3 Translating transcripts to ASR output

Above we discussed methods for generating hallucinated ASR n-best lists using phone-based FSTs. We now turn to generating confusions at the word level. In this section, we discuss using standard phrase-based statistical machine translation techniques to "translate" from reference transcriptions to ASR-like output.

Figure 8.5 presents the basic "MT" pipeline used to hallucinate ASR output. There are three input data sources required: 1) a parallel training corpus consisting of source sentences of human-generated reference transcriptions and target sentences of true ASR output; 2) training data on which to build a language model for the target language output; and 3) data in the source language to be "translated" into the target language. We generate word-level alignments of the sentences in the parallel corpus, using a standard word alignment package such as Giza++ or the Berkeley aligner. From these alignments, we build a phrase table and tune the weights of the feature functions for the phrase translation rules using Moses. Finally, we decode the data to be translated to generate the hallucinated ASR n-best lists. These n-best lists are then used as training data for the perceptron.

8.2.3.1 Preliminary experiments

In the parallel corpus used to train the system, the source language sentences are human-generated references transcriptions of speech. These reference transcriptions can be either word sequences or phone sequences created by concatenating the canonical pronunciations of the component words. The target language (i.e., the "language" we want to generate) is true ASR output. In the experiments presented here, we use only the top candidate from the ASR n-best lists as the target side of our parallel corpus. In these preliminary experiments, the translation model was trained on the 43,000 sentences from the Switchboard 19 fold of the CTS corpus.

The language model used in our two preliminary experiments was built on the target side (i.e., the ASR output side) of the parallel training corpus. In order to compare the perceptron results produced using the MT-derived n-bests lists with those produced using FST-based n-best lists, we translated the roughly 50,000 reference sentences for the Switchboard 18 fold of our ASR training corpus.

Giza++ was used to train the word alignments. Grammar extraction, tuning, and decoding were performed with Moses using default settings with a few exceptions. Since there is no reordering of words in this kind of "translation", it was not necessary to build a reordering model. We also set the distortion limit of the decoder to zero in order to prohibit reordering during decoding. In the phone-to-word models, we increased

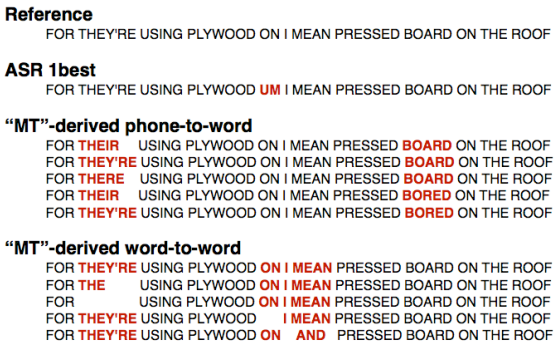


Figure 8.6: Examples of "MT"-hallucinated ASR output.

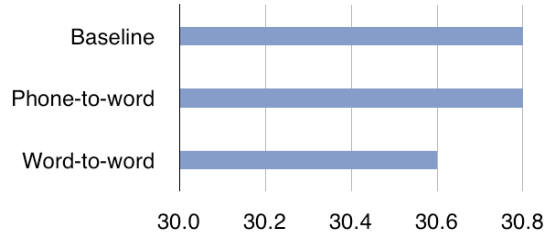


Figure 8.7: Preliminary perceptron WER results.

the default maximum phrase size to allow much longer phrases in the phrase table, since a single word is usually made up of multiple phonemes.

Figure 8.6 provides examples of the hallucinated output generated by the first two experimental "translation" models: one in which the source side contained reference phone strings (*phone-to-word*) and one in which the source side of the parallel corpus consisted of reference word strings (*word-to-word*) and . We observe that the variation in the phone-to-word output is mostly limited to pure homophone substitution, while the word-to-word output include more complex deletions, insertions, and substitutions based on phonetic similarity.

Figure 8.7 shows the word error rates obtained using the output of these two systems as training data for the perceptron re-ranker. We see that the phone-to-word hallucinated n-best lists do not improve WER, while the word-to-word hallucinated lists improve WER by 0.2. In the remainder of this section we discuss modification made to the word-to-word system in order to further decrease WER.

8.2.3.2 Improving hallucinations

Our first attempts at generating better n-best lists focused on increasing the quantity of input data used to train the "translation" model, the language model, and the perceptron itself. First we expanded the parallel corpus to include more of the other folds in the CTS dataset. Given that we are performing a kind of monolingual translation, we experimented with training the LM on other English data, including the reference transcriptions and other in-domain English language data collected from the web. Once a translation model is built, it is not necessary to limit the hallucination of ASR n-best lists to actual ASR data. Any English language text can be "translated" into ASR-like output that can be used to train the perceptron, which allows us to produce a much larger set of -n-best lists for the perceptron.

Figure 8.8 shows the WER produced under variations of these different data conditions. The small models were built using the 43,000 sentences of Switchboard 19; the medium models were built on roughly 200,000 sentences, and the large models were built using roughly 1 million sentences. Increasing the size of the parallel corpus and the amount of data used to train the language model improves WER. We also see that using

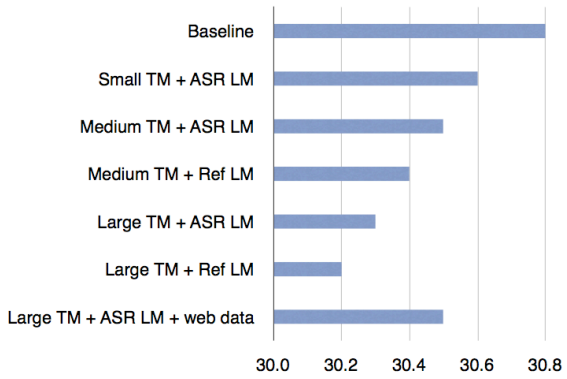


Figure 8.8: Further perceptron WER results.

a language model built on the reference transcriptions rather than the ASR output results in a decreased WER. Somewhat surprisingly, increasing the amount of data used to train the perceptron by "translating" large amounts of in-domain data collected from internet sources did not improve the performance of the perceptron.

8.2.3.3 Future improvements

We plan to continue to explore ways of modifying the three input data sources. Given the weak performance of the perceptron when trained on n-best lists generated from very large amounts of in-domain web data (shown in the final row of Figure 8.8), we will try training the perceptron on n-best lists generated exclusively from reference transcriptions from additional folds of the CTS corpus. We also plan to expand the parallel corpora used to train the "translation" model by including more candidates from the true ASR n-best lists, by taking the top k elements or sampling from the n-best list.

The processes in the MT pipeline can be customized to the specific task at of monolingual, monotonic "translation". For instance, existing word alignment packages allow for re-ordering of elements, which is not ideal for our data. Altering the alignment HMM to prohibit backward distortions might improve word alignments, which in turn might improve translations and perceptron performance. We also plan to explore altering the evaluation metric against which the feature function weights of the grammar are tuned. ASR is evaluated in terms of WER, and ASR models are tuned to minimize WER. Machine translation, however, is typically evaluated and tuned according to the BLEU metric. Our future work will include modifying the minimum error rate training phrase of the "MT" pipeline to tune to WER rather than BLEU.

8.2.4 Phrasal cohorts

The approach described here bears some similarity to the MT-based confusion generation, but has the advantage of being more straightforward. As we said, the confusion generation problem for discriminative LM(DLM) can be thought of as a translation problem, but much simpler than the real translation between different languages. The key observation here is that the ASR confusions do not involve any reordering, the entire recognition pipeline is a time-synchronized process. Unlike in MT, where the translation rules have to be extracted in more complex ways, it is very easy to identify the mistakes that an ASR system makes from simple string alignments.

In this part of the report, we will present a method to extract phrase-level confusions directly, and use these confusions for training DLM. We will also extend the method to the *unsupervised* case: The standard confusion extraction requires at least some labeled data(transcribed speech) in order to learn the transformation from the clean reference text to the erroneous ASR output. However, labeled data are usually limited and expensive to acquire, we will assume a scenario where the DLM has to be trained on un-paired speech and text data, thus the confusions have to be learned on *un-transcribed* speech only.

p	$C(p)$
that to you	that too you
<s>she	<s>and she, <s>oh she
to you	to be you

Table 8.1: Examples of cohort

8.2.4.1 Cohort

Central to our methodology is the definition of *cohort*. The cohort for a phrase p, denoted as $C(p)$, is the set of the phrases that are potential erroneous outputs where the correct transcription is p. Table 1 contains a few examples of the cohorts that we extracted.

Note that the phrases in Table 1 contain identical left and right contexts that appear on both sides of the translation rules, which we will call *pivots*, thus the confusions that we learned can be thought of as context-dependent. It's not necessary to constrain the pivots to be a single word, we can also have multiple words or phone-level contexts serving as pivots. If we remove the pivots, we then get a set of context-independent rules. In either case, the key to extracting cohorts is identifying pivots, which can be easily obtained from Levenshtein string alignments.

The extracted cohorts can be used to hallucinate n-best lists from the clean text without the corresponding acoustics. Although the DLM is trained in an unsupervised manner in the sense that it does not require transcribed speech to generate the real n-best list, acquiring cohorts may or may not require transcribed data.

8.2.4.2 Supervised cohort acquisition

In the supervised scenario, we assume there's a set of *transcribed* speech data where we can learn phrase-level transformations. By aligning the reference string with the ASR hypothesis based on edit distance, we can easily find out the errors that the ASR system produces. Below is the alignment between the reference and the 1-best ASR output.

Reference: <s>What kind of *a company is it* <s>
1-best: <s>What kind of *the company that* <s>

Highlighted in red are the pivot words. The phrases between the pivots form a phrase-level transformation, where "a company is it" gets falsely recognized as "the company that". Such alignments can be done between the reference text and each one of the n-best hypothesis, where n can be a parameter to tune. We can also assign weights to the translation rule based on the number of times such errors appear in the n-best hypotheses. In the example above, let n be 10, if "a company is it" becomes "the company that" in all of the 10-best hypotheses, then the rule has probability 1, if 5 of them make this error, the probability is 0.5.

With the set of weighted translation rules obtained from the transcribed speech, we can hallucinate n-best lists from the text only. We'll discuss n-best generation in Section 8.2.4.4.

8.2.4.3 Unsupervised cohort acquisition

Transcribed speech data are generally expensive to acquire. For a lot of tasks, we might not have enough such data to learn confusions. If we had, the standard supervised DLM using the available transcribed resources may have already set a strong baseline, and it can be hard to expect further gains from using extra text data with hallucinated confusion sets. A more promising scenario will be trying to make use of the abundant un-paired speech and text data. In this section, we show that phrasal cohorts can also be extracted from un-transcribed speech data. Despite being very coarse in the way such cohorts are extracted, we do not seem suffer too much performance loss as measured by word error rate.

Unsupervised cohort extraction has been explored before, but at the single word level. The model used was also different, and intuition was trying to build a locally normalized model to disambiguate word-level confusions. Therefore, what described here can be seen as an extension to the more general phrase-level confusions.

The acquisition of cohorts from un-transcribed speech is almost no different from the supervised case as described in previous section. The only important change is that the obtained translation rules become symmetric: If there is a rule of "$A \rightarrow B$", there will also be a rule "$B \rightarrow A$". Such symmetry comes from the fact that in the ASR output without corresponding references, we can not determine whether A or B is the correct phrase, all we know is that A and B are competing with each other. By aligning all pairs of hypotheses in the n-best list, we'll be able to learn a set of such competitions. If either A or B turns out to be the correct transcription, at least one of the rules that we extract will be legit, although the transformation in the other direction shouldn't have been learned as a rule. If neither of the phrases is correct, then both rules are fake.

Another distinction in the unsupervised setting is that it's not straightforward to assign weights to the extracted symmetric rules. The posterior probabilities of the n-best hypotheses can be an important source of information, but the best methodology remains to be found. As an interesting direction for future research, it is not pursued in this workshop. Therefore, the cohorts we extracted in the unsupervised scenario are all unweighted.

As we can see, such symmetric definition of cohorts brings a lot of bogus confusions which may be harmful to the hallucination of the n-best list. Also the lack of transformation probabilities may also corrupt the statistics of the hallucinated data. We will show with empirical results that such impacts exist but are limited, and gains can still be achieved by building unsupervised DLM with cohorts learned also from un-transcribed data.

8.2.4.4 N-best generation using cohorts

The set of cohorts essentially defines a phrase table which describes possible transformations from the reference to the ASR output. These translation rules can be applied easily to the text and generate n-best output. Figure 8.9 shows an example of the constructed confusion network, using the cohorts in Table 1.

Figure 8.9: N-best generation

The green sentence comes from the text corpus, and thus is the reference text for the hallucinated output. The red words are the simulated errors. From the confusion network, we can extract n-best hypotheses for discriminative training. Note that the transitions through the red words can be associated with costs, a LM can also be used to score each path in the confusion network, making the optimal combination weight an extra hyper-parameter to tune. To simplify the experiments, we will not combine scores: For supervised cohorts, we use translation costs only; For unsupervised cohorts, where the translation costs are not easily attainable, we use LM only.

8.2.4.5 Experiments & Results

All DLM are trained on partition 18, which contains 70hrs of speech and 800k words of transcripts. The basic average perceptron are used for all models with only n-gram features up to trigram.

For the supervised DLM, we have access to both text and speech of partition 18.

For unsupervised DLM, we assume we only have text for partition 18, thus the n-best list can only be hallucinated. In order to extract cohorts, we use partition 19, which contains 63hrs of speech and 860k words, and we either have access to both speech and text or speech only.

The word error rates are reported on partition 16, which contains 15hrs of speech and 215k words.

The results are shown in Figure 8.10. The supervised DLM brings 1% absolute WER improvement over the baseline, from 30.9% to 29.9%. The unsupervised DLM using supervised cohorts achieves half of the gains at 30.4%, and unsupervised cohort extraction does not seem to degrade performance by much.

Model	WER
Baseline	30.9
Un-supervised cohorts + DLM training	30.5
Un-supervised DLM training	30.4
Supervised DLM training	29.9

Figure 8.10: Word Error Rates

8.2.4.6 Future work

Despite the positive results, the approach we described here is very straightforward and maybe coarse from place to place. The way we extract cohorts and apply them for n-best generation has a lot of room for refinement. The cohort approach is promising in the sense it does not require any labeled data(transcribed speech) therefore the resources that can be utilized are abundant. The immediate next step will be scale up the experiments and see whether the improvement that we showed here can sustain. We also hope to find out whether adding more data (text and/or speech) can improve performance. As more data coming in, the coarseness in our method may start to deteriorate results, thus making refining the cohorts a very interesting research direction to explore.

8.3 Final Controlled Experiments

We present three methods for generation of simulated ASR n-best lists from text to serve as training material for discriminative language models. The first is based on finite-state transducers that model phone confusion in the output of the baseline system. The other two methods consider confusability at the level of words and phrases directly, using techniques developed for machine translation.

8.3.1 Finite-state transducers for modeling phone confusion

Our first proposed method for generation of simulated n-best lists is to model confusions at the phone level using a process that can be thought of as a simulation of an ASR system without actual speech. The more faithful our simulation is to the targeted ASR system, the more likely it is that the resulting simulated n-best lists will be effective training material for discriminative language modeling.

Given a sequence of words S and a weighted finite-state transducer X that maps phone sequences to confusable phone sequences, a list of phonetically confusable word sequences can be generated using a pronunciation dictionary L and a generative language model G (also encoded as finite-state transducers) through the composition $S \circ L \circ X \circ L^{-1} \circ G$. The initial sequence of words is composed with the pronunciation dictionary, resulting in a corresponding lattice of phone sequences (since words in the dictionary may have multiple corresponding pronunciations). Composing this lattice with a phone confusion transducer produces a lattice encoding several confusable phone sequences. An inverse pronunciation dictionary is then used to create a lattice of words from the lattice of phones, and finally the generative n-gram language model is composed with this lattice of words. A list of word strings confusable with the input string S, scored according to phonetic confusability and the n-gram language model, can then be generated from this resulting lattice. Since the same dictionary and generative language model used in an ASR system can be used in this approach, whether or not the resulting lists are similar to what the ASR system would produce depends crucially on the phone confusion transducer X.

To create a phone confusion transducer, we use real n-best lists produced by the ASR system we want to simulate, coupled with the reference transcription corresponding to each n-best list. Using the dictionary L, the reference strings and each of the strings in the n-best lists are mapped to corresponding sequences of phones. If multiple pronunciations are listed for words in L, only one is used, so that each word sequence

is mapped to exactly one phone sequence. For each n-best list, we then produce Levenshtein alignments between the reference phone sequence and each n-best hypothesis phone sequence. Based on each of these individual alignments of pairs of phone strings, each phone in a reference string corresponds to the same phone in the hypothesis string (phone identity), to a different phone in the hypothesis string (a substitution), or to no phone (ϵ) in the hypothesis string (a deletion). Additionally, a phone in the hypothesis string might not correspond to any phones in the reference string, in which case ϵ in the reference string is aligned with the phone in the hypothesis string (an insertion). Each alignment between two phone sequences is then composed of a sequence of such phone-to-phone (or ϵ-to-phone, or phone-to-ϵ) mappings, represented as X:Y, where X and Y can be any phone or ϵ. Using these sequences, we estimate an n-gram model of phone identity, substitutions, insertions and deletions, following [67]. We encode this n-gram model as a weighted finite-state acceptor with X:Y pairs in the transition arcs, and by separating these pairs so that X is an input symbol and Y is an output symbol, we obtain a phone confusion transducer X. In our experiments, we used 5-gram phone confusion models.

One practical difficulty with the computation of $S \circ L \circ X \circ L^{-1} \circ G$ for a given text string S is that X allows nearly arbitrary deletions, insertions and substitutions in nearly any context. While the vast majority of deletions, insertions and substitutions have extremely low probability in most contexts, they are still encoded in X, making any composition involving X and L^{-1} infeasible. We approach this problem with two strategies. First we prune low probability transitions from X, reducing the size of the transducer by 60% (the amount of pruning was determined empirically based on performance on a small held-aside set). The second strategy is to modify the original composition slightly: given a string S, we first compute $S \circ L \circ X$, then find the 300 best paths P (which correspond to the highest scoring confusable phone sequences according to X), and finally proceed with $P \circ L^{-1} \circ G$ (the number of confusable phone sequences to keep was also determined empirically). Extraction of n-best paths from the resulting transducer produces the simulated n-best lists.

8.3.2 Machine translation methods

Above we discussed methods for generating hallucinated ASR n-best lists using phone-based finite-state transducers. We now turn to generating confusions at the word level. In this section, we discuss using standard phrase-based statistical machine translation techniques (MT) to "translate" from reference transcriptions to what those strings might be as ASR output.

Figure 8.11 presents the basic "MT" pipeline used to hallucinate ASR output, slightly modified from Figure 8.5. There are three input data sources required: 1) a parallel training corpus consisting of source sentences of human- generated reference transcriptions and target sentences of true ASR output; 2) data on which to build a language model for the target language output; and 3) data in the source language to be "translated" into the target language. We first generate word-level alignments of the sentences in the parallel corpus. From these word alignments, we build a phrase table and tune the weights of the feature functions for the phrase translation rules. Using the phrase table and associated weights, we decode the data to be "translated" to generate the hallucinated ASR n-best lists that can then be used as training data for the perceptron.

In the experiments presented here, we use only the top candidate from the ASR n-best lists as the target side of our parallel corpus. The language model for translation was built on the source side (i.e., the reference transcriptions) of the parallel training corpus. Giza++ ((http://code.google.com/p/giza-pp/) was used to train the word alignments. Grammar extraction, tuning, and decoding were performed with Moses (http://www.statmt.org/moses) using default settings with a few exceptions: since this kind of "translation" is monotonic, it was not necessary to build a reordering model; we also set the distortion limit of the decoder to zero in order to prohibit reordering during decoding.

8.3.3 Phrasal cohort methods

The approach described in this section bears some similarity to the MT-based confusion generation. As above, we think of the confusion generation problem as a kind of translation, but one that is in many ways simpler than translation between two natural languages. This allows us to use a more straightforward approach than in a full MT system, rather than configuring existing MT software for this task. The key

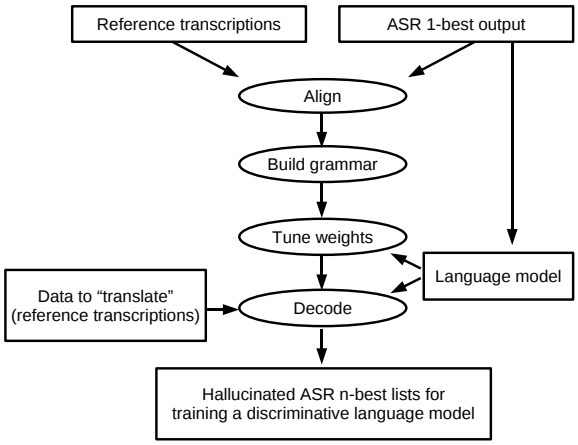


Figure 8.11: "MT" pipeline for hallucinating ASR n-best lists.

observation here is that ASR confusions do not involve any reordering: the entire recognition pipeline is a time-synchronized process. Unlike in MT, where the translation rules have to be extracted in more complex ways, it is easy to identify the mistakes that an ASR system makes from simple string alignments.

Central to our methodology is the definition of *cohort*. The cohort for a phrase p, denoted as $C(p)$, is the set of the phrases that are potential erroneous outputs where the correct transcription is p. Table 8.1 contains a few examples of the cohorts that we extracted from our baseline ASR system. Note that the phrases in Table 8.1 contain identical left and right contexts, which we call *pivots*, thus the confusions that we learned can be thought of as context-dependent. It is not necessary to constrain the pivots to be single words; we could also have multiple words or phone-level contexts serving as pivots. If we remove the pivots, we then get a set of context-independent rules. In either case, the key to extracting cohorts is identifying pivots, which can be obtained from Levenshtein string alignments.

To extract cohorts, we assume there is a set of *transcribed* speech data where we can learn phrase-level transformations. By aligning the reference string with the ASR hypothesis based on edit distance, we can easily find the errors that the ASR system produces. Below is the alignment between the reference and the 1-best ASR output.

Reference:	<s>What kind of *a company is it* </s>
1-best:	<s>What kind of *the company that* </s>

The pivot words are underlined. The phrases between the pivots form a phrase-level transformation, where "a company is it" is incorrectly recognized as "the company that". Such alignments can be done between the reference text and each one of the n-best hypothesis, where n can be a parameter to tune. We can also assign weights to the translation rule based on the number of times such errors appear in the n-best hypotheses. In the example above, let n be 10, if "a company is it" becomes "the company that" in all of the 10-best hypotheses, then the rule has probability 1, if 5 of them make this error, the probability is 0.5.

With the set of weighted translation rules obtained from the transcribed speech, we can hallucinate n-best lists from the text only. The set of cohorts essentially defines a phrase table which describes possible transformations from the reference to the ASR output. These translation rules can be applied easily to the text and generate n-best output. Application of these rules to a particular input string produces a confusion network. Each path through this network is a confusable string, which is scored according to the translation rule probabilities. Although it is possible to take generative n-gram language modeling scores into account in this framework, in this paper we used only scores derived from the translation rules.

8.3.4 Experimental Results

In our experiments, we used the perceptron algorithm to train discriminative language models (DLM) for re-ranking n-best list output (100-best lists, more precisely) from the baseline ASR system. We follow the

		dev	eval
1-best ASR		22.8	25.7
Real n-best lists	2 folds	22.3	–
	4 folds	21.8	–
	8 folds	21.7	24.8
Phone model lists	2 folds	22.4	–
	4 folds	22.3	–
	8 folds	22.3	25.3
MT model lists	2 folds	22.5	–
	4 folds	22.5	–
	8 folds	22.4	25.4
Cohort model lists	2 folds	22.3	–
	4 folds	22.4	–
	8 folds	22.2	25.2

Table 8.2: WER results using various n-best lists.

approach outlined in [2, 3], using unigrams, bigrams and trigrams as features in the model, and using the averaged perceptron for evaluation on held-aside, development and evaluation sets. The perceptron is an on-line algorithm, and after every epoch over the training set as a whole, performance was evaluated on a small held-aside set. Training stopped when performance on the held-aside set failed to improve for 5 epochs, and the averaged perceptron model for the lowest held-out WER epoch was output.

One difference from the methods in [2, 3] is that we did not use the baseline ASR system score as a feature during training of the n-gram DLM. Rather, we empirically optimized the scaling parameter for the baseline ASR score when adding it to the score of the DLM, after training the DLM independently. For the current results, we optimized the mixing factor on the development set, and used that mixing parameterization with both development and evaluation sets.

Table 8.2 presents our results obtained using 100-best lists hallucinated from text using each of the methods described in section 8.3. We show results obtained on development data with varying amounts of training data to illustrate the effect of dataset size, but test each method on the evaluation set using only the configuration with lowest WER on the development set, which, not surprisingly, is the configuration using the larger portion of the data for each method. For comparison, we also show the error rates for the 1-best result of the baseline ASR system, and for fully supervised DLMs trained on real n-best lists produced by the baseline ASR system. Using real n-best lists, we obtain an absolute improvement of 0.9% in word error rate over the baseline, or a 3.5% relative improvement. Our approach based on phrasal cohorts produced an absolute improvement of 0.5% over the 1-best baseline, which is statistically significant ($p < 0.001$) and more than half of the improvement obtained using real n-best lists. Our phone confusion approach produced an improvement of 0.4% ($p < 0.005$). Because these two approaches work at different levels (phones vs. words and phrases), it is possible that there might be ways to leverage both approaches for further improvements. The improvement over the baseline obtained using our MT-based approach was smaller, at 0.3%, but still statistically significant at the $p < 0.05$ level.

8.4 Conclusion

We presented three approaches for generation of hallucinated n-best lists from text that mimic ASR n-best lists produced from speech, and showed that discriminative language models trained on these hallucinated n-best lists can improve ASR word error rates from a strong baseline. One advantage of the use of hallucinated n-best lists is that it allows training of discriminative language models using arbitrary text, instead of transcribed speech.

In contrast to previous work, we compared the use of n-best lists produced using phone-level and word/phrase-level confusion modeling. We showed that approaches based on either level of confusion can be beneficial in discriminative language modeling, suggesting that an approach that works at both levels might produce even further gains.

Chapter 9

Semi-supervised Discriminative Language Modeling for Turkish ASR

We present our work on semi-supervised learning of discriminative models where the negative examples for sentences in a text corpus are generated using confusion models for Turkish at various granularities, specifically, word, sub-word, syllable and phone level. We experiment with different language models and various sampling strategies to select competing hypotheses for training with a variant of the perceptron algorithm. We find that morph-based confusion models with sample selection from simulated hypotheses to match the error distribution of baseline ASR system gives the best performance. We also observe that substituting half of the supervised training examples with examples obtained in a semi-supervised manner gives similar results.

9.1 Introduction

In automatic speech recognition (ASR), language models assign weights to word sequences to discriminate between acoustically similar sequences. Discriminative training of language models has been shown to improve the speech recognition accuracy by resolving acoustic confusions more effectively [3]. In discriminative language modeling (DLM), a speech recognizer is employed to generate a set of competing hypotheses for an utterance. Given the correct transcription of an utterance and the set of competing hypotheses (confusion set), discriminative learning techniques can be applied to make use of positive and negative examples to reward features in the correct transcription and penalize features in the competing hypotheses.

However, this approach requires a large amount of transcribed speech data. Several approaches have been proposed to overcome the necessity of supervised learning for DLM. For instance, Xu et al. propose a self-supervised discriminative training method, in which an exponential language model is trained using only untranscribed speech and a large text corpus [25]. The discriminative training criterion is based on maximization of the likelihood ratio between words w in the text corpus and other words that w competes with (*cohorts*) in the first-pass ASR output lattices for untranscribed speech utterances. In another work, Kurata et al. propose to generate the probable n-best lists that an ASR system may possibly output, for an input hypothetical utterance given a word sequence [23]. They call this process *Pseudo-ASR* since they use phoneme similarities estimated from an acoustic model to generate the competing hypotheses. The discriminative training of the model is based on the generalized probabilistic descent (GPD) algorithm and more recently they applied discriminative reranking using the perceptron algorithm [24]. In another study, Tan et al. propose a system for channel modeling of ASR for simulating the ASR corruption using a phrase-based machine translation system trained between the reference phoneme and output phoneme sequences from a phoneme recognizer [19]. Jyothi et al. have also modeled the phonetic confusions using a confusion matrix that takes into account word-based phone confusion log likelihoods and distances between the phonetic acoustic models [22]. The confusion matrix is used to generate confusable word graphs for training a discriminative language model using the perceptron algorithm.

In this paper, we investigate various confusion models to employ in semi-supervised learning of discrim-

input set of training examples $\{y_i : 1 \leq i \leq n\}$
input number of iterations T
$\bar{\alpha} = 0$, $\bar{\alpha}_{sum} = 0$
for $t = 1 \ldots T$, $i = 1 \ldots n$ **do**
$z_i = \text{argmax}_{z \in \mathbf{GEN}(y_i)} \bar{\alpha} \cdot \mathbf{\Phi}(z)$
$\bar{\alpha} = \bar{\alpha} + \Delta(y_i, z_i)(\mathbf{\Phi}(y_i) - \mathbf{\Phi}(z_i))$
$\bar{\alpha}_{sum} = \bar{\alpha}_{sum} + \bar{\alpha}$
end forreturn $\bar{\alpha}_{avg} = \bar{\alpha}_{sum}/(nT)$

Figure 9.1: The WER-sensitive perceptron algorithm

inative reranking models for Turkish. Since Turkish is an agglutinative language with a highly productive morphology, the number of distinct words is considerably high compared to other languages. Hence, we experiment with confusion models at various granularity, namely, word, sub-word (morph), syllable and phone level. The confusion models are trained over transcribed speech which is generally available for acoustic modeling. However, for discriminative training, the negative examples corresponding to sentences from a text corpus are generated using confusion and language models. We experiment with various language models to choose among possible confusions. For discriminative training of models, we use a variant of the perceptron algorithm, *WER-sensitive perceptron* which has been shown to perform better for reranking ASR hypotheses [68]. Rather than just using the top simulated hypotheses output by the model, we also experiment with different sampling strategies. The strategy that simulates the word error distribution in the ASR output gives the best improvement.

The paper is organized as follows. In section 2, we introduce the discriminative training of reranking models. Following that, in section 3, we describe the system. In section 4, we present the experimental setup and the results, before concluding with section 5.

9.2 Semi-supervised Discriminative Reranking

We use a variant of the averaged perceptron algorithm, the *WER-sensitive perceptron* [68], to estimate the parameters of reranking model discriminatively as shown in Figure 9.1. The standard perceptron algorithm is trained to minimize the number of misclassifications, whereas the WER-sensitive perceptron algorithm is trained to minimize a loss function which is defined using edit distances of hypotheses with the reference transcription. Hence, discriminative training of feature parameters runs on the criteria of minimizing the WER rather than the number of misclassifications.

The algorithm estimates a parameter vector $\bar{\alpha} \in \mathbb{R}^d$ using a set of training examples. In the conventional DLM, the function **GEN** generates a set of hypotheses for an acoustic input x using a baseline speech recognizer. In this work, we aim to generate a confusion set of sentences for an input sentence y as similar as possible to what we would get from a recognizer if we had the acoustic utterance for that sentence. Hence, learning is semi-supervised in the sense that training does not directly require transcribed speech examples but we use transcribed speech to build confusion models to generate the training examples. The representation $\mathbf{\Phi}$ maps each y to a feature vector $\mathbf{\Phi}(y) \in \mathbb{R}^d$. This work uses morph unigram counts as features. The function $\Delta(y_i, z_i)$ is defined as the edit distance between z_i and y_i.

The learned averaged parameter vector $\bar{\alpha}_{avg}$ can be used for mapping an unseen acoustic input x to an output y by searching for the best scoring hypothesis:

$$y = \underset{\tilde{y} \in \mathbf{GEN}(x)}{\text{argmax}} \{\lambda \log P(\tilde{y} \mid x) + \bar{\alpha}_{avg} \cdot \mathbf{\Phi}(\tilde{y})\}$$

where $\mathbf{GEN}(x)$ is a set of hypotheses output from the baseline recognizer with the recognition score $\log P(\tilde{y} \mid x)$, and λ is a scaling factor optimized on a held-out set.

The WER-sensitive perceptron gives significantly better results (0.4%) than the standard perceptron when trained on the real ASR hypotheses while omitting the baseline recognition score. We also observe that the WER-sensitive perceptron is oblivious to the omission of the baseline recognition score in training.

This is important since the scores output from the confusion models do not match the ASR baseline scores, and it would result in a mismatch between training and testing.

9.3 System Description

The flow of our system starts with the generation of a confusion graph for a given word sequence. We then extract n-best hypotheses ($n = 1000$) out of that graph and apply different sampling schemes in order to select instances with different properties from a wide range of samples throughout the 1000 hypotheses. In the final step, we use sampled hypotheses ($n = 50$) to train the language model discriminatively.

9.3.1 Confusion Graph Generation

$$\mathcal{C}(\mathcal{W}) = (prune(\mathcal{W} \circ \mathcal{L}_{\mathcal{W}} \circ \mathcal{CM}) \circ \mathcal{L}_{\mathcal{M}}^{-1} \circ \mathcal{G}_{\mathcal{M}})$$

Confusion graph $\big(\mathcal{C}(\mathcal{W})\big)$ generation happens in two stages. In the first stage, the given word sequence in the lattice $\mathcal{W}$ is first composed with the lexicon $\mathcal{L}_{\mathcal{W}}$ and then with the confusion model $\mathcal{CM}$. Based on the granularity of the $\mathcal{CM}$, composition with the $\mathcal{L}_{\mathcal{W}}$ transforms $\mathcal{W}$ into phone, syllable, morph or word-based lattice. Later composition with the $\mathcal{CM}$ simulates the noisy channel property of ASR as each inserted confusion unit adds noise over the reference unit. Based on the length of the word sequence and the number of possible confusions, the resulting lattice might be very large. Hence, we prune that lattice by using only those arcs that form its 1000-best paths. Further investigation showed that this filtering has no negative effect on the results.

In the second stage, if the resulting lattice from the first stage is not morph-based, it is first composed with the inverse morph lexicon $\mathcal{L}_{\mathcal{M}}$ since the language model transducer $\mathcal{G}_{\mathcal{M}}$ is morph-based. $\mathcal{G}_{\mathcal{M}}$ scores the sequences in the lattice based on their likelihood seen in a typical sentence in the language being modeled.

9.3.1.1 Confusion Models

We use five different $\mathcal{CM}$s. While two of them are phone-based, the other three are syllable, morph, and word-based models. One of the phone-based $\mathcal{CM}$s is estimated using the acoustic similarities of phones by calculating the Bhattacharrya distance (bh) between the representative Gaussians of each phone in the acoustic model of the ASR, as proposed in [23].

The other four $\mathcal{CM}$s are estimated with the edit distance (ed) method, where we align the reference sentence with its recognized transcription and extract cases of substitutions, insertions, and deletions by using one of the optimal alignments. Unlike what was done in [22], we didn't use any special cost function to get the best alignment.

9.3.1.2 Language Models

We use three different LM approaches: GEN-LM, ASR-LM and NO-LM. While GEN-LM is estimated from Turkish newswire data set collected from the Internet, ASR-LM is derived from ASR n-bests. Since our ultimate goal is to simulate the ASR output, using the ASR-based LM is more intuitive as it is supposed to give higher scores to those alternatives that resemble the ASR output most. As a third approach, called NO-LM, we choose not to apply any language model, which means that only the scores of the $\mathcal{CM}$ are used to pick the n-best out of the lattice at the end.

9.3.2 Sampling Schemes

The top 50 simulated hypotheses from the confusion graph have very different word error (WE) distribution than the ASR WE distribution, containing less WEs. Hence we sample 50 instances from the top 1000 in order to match to the WE distribution of the ASR. We use two specific sampling schemes.

The first sampling scheme, called the Uniform Sampling (US), follows the method applied in [69]. We select instances in uniform intervals from the WER-ordered list, hoping that they contain more WEs. In this scheme, denoted US-k, the best and the worst hypotheses are always selected.

The second approach is where we specifically sample instances according to the actual WE distribution obtained from the ASR output. We learn how frequently each unique word error occurs in the ASR 50-best, and try to simulate this distribution by picking samples from the artificially generated 1000-best in proportional numbers. We call this method ASRdist-50.

Note that, both sampling approaches sort the n-best list in ascending number of word errors before doing sampling.

9.4 Experiments

9.4.1 Experimental Setup

In this study, DLM is applied on a Turkish broadcast news transcription task [70]. The data set we use in our experiments consists of approximately 194 hours of speech recorded from news channels. We divide the data into disjoint training (188 hours), held-out (3.1 hours) and test (3.3 hours) subsets, containing a total of 105355, 1947 and 1784 utterances, respectively. The training subset is further divided into 12 equi-size bins. The bins 1-6 are used to obtain the n-best lists from which the confusion models are learned. As most of the confusions happen with very low probability, to reduce the computational cost, we filtered out all confusions with probability less than 0.01 in all models. After pruning, the phone-based models contain over a hundred parameters, whereas the syllable, morph, and word-based models contain 28K, 137K, and 362K parameters, respectively. These models are then used to generate the artificial n-bests from the reference transcriptions of bins 7-12.

We use the SRILM toolkit for building language models and the OpenFST library for finite-state operations. The speech recognition system is based on the statistical morph-based ASR system of [70]. Using this setup, the generative baseline and oracle WER on the held-out set are 22.9% and 14.2% and on the test set are 22.4% and 13.9%, respectively. When we use ASR 50-best from bins 7-12 for discriminative training, WERs drop to 22.1% and 21.8% on the held-out and the test sets, respectively.

9.4.2 Results

In our experiments, we use five different $\mathcal{CM}$s with three different LM approaches. Table 9.1 shows the WER improvement on the held-out set for these 15 different configurations. The results are obtained with the ASRdist-50 sampling scheme.

Table 9.1: WER (%) on held-out for different CMs and LMs w/ ASRdist-50 (held-out baseline: 22.9%; real (ASR) 50-best: 22.1%)

CMs	GEN-LM	ASR-LM	NO-LM
phone-based (bh)	22.8	22.7	N/A[1]
phone-based (ed)	22.6	22.7	N/A[1]
syllable-based	**22.5**	**22.4**	22.6
morph-based	22.6	**22.4**	**22.5**
word-based	22.6	**22.5**	22.7

Based on the NIST MAPSSWE test, phone-based models yield significantly smaller WER improvements compared to syllable-, morph- and word-based models, whereas the difference between these three models is not significant. When compared with the baseline on the held-out set, configurations given in bold in Table 9.1 are significantly better at $p < 0.005$. For the comparison of LMs, even though there is no significant difference between them, configurations with the ASR-LM result in significantly better WER improvement over the baseline. The best configuration uses the morph-based $\mathcal{CM}$ and the ASR-LM and is significantly better than the baseline at $p < 0.001$ on the held-out set.

With this best configuration, a comparison of the different sampling schemes over the test set is shown in Table 9.2. While using Top-50 and US-50 sampling strategies is not significantly different than using no sampling strategy, improvement by the ASRdist-50 is significant at $p = 0.006$. This supports our assumption,

[1]We didn't run these configurations because not using a LM with phone-based CMs takes too much computational time.

Table 9.2: Sampling from the 1000-best list w/ morph-based $\mathcal{CM}$ and ASR-LM (test set baseline: 22.4%; real (ASR) 50-best: 21.8%)

Sampling Strategy	WER on test set	KL-distance
NoSampling	22.1	0.38
Top-50	22.1	0.43
US-50	22.0	0.27
ASRdist-50	21.8	0.23

Table 9.3: Test set results for combining real and simulated n-best lists (test set baseline: 22.4%)

bins 1-6	bins 7-12	WER (%)
Real (ASR)	Real (ASR)	21.5
Real (ASR)	Simulated (ASRdist-50)	21.6

that is, the more simulated n-bests resemble the ASR output in terms of WE distribution, the better WER improvement we get. Figure 9.2 shows how the WE distribution of the ASRdist-50 is more similar to ASR's compared to Top-50's. We also measure this similarity in terms of KL-distance given in Table 9.2 for each sampling strategy. To test our assumption even further, we calculate the correlation between the KL-distance and WER improvement over all experiments to see whether the drop in KL-distance correlates with the drop in WER and find out that there is a correlation of 0.84, which further supports our assumption.

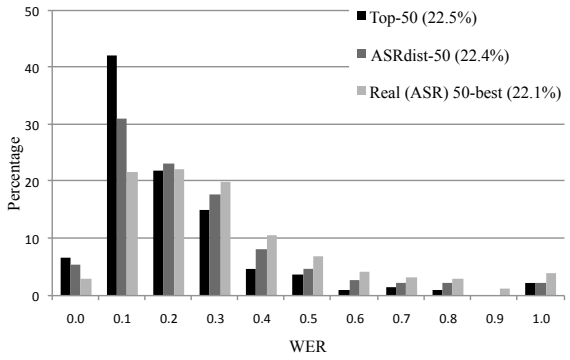


Figure 9.2: Word error distribution on the held-out set

We also test if our artificially generated n-bests give similar results with the real ASR n-bests. Table 9.3 shows that when we combine the real n-bests from the bins 1-6 with the simulated ones from the bins 7-12, we get similar WER improvement with respect to the real n-bests from the bins 1-12.

9.5 Conclusion

In this study, we examined the semi-supervised learning of discriminative language models for a Turkish ASR system. We experimented with different confusion and language models. We observe that phone-based confusion models are outperformed by syllable-, morph- and word-based models whereas there is no significant difference among these three. In case of difference between LMs, ASR-LM gives better WER improvement with respect to the baseline. Moreover, we get the best WER improvement by trying to match the ASR WE distribution. Finally, when we substitute half of the real n-best lists in our data set with the simulated ones, we achieve almost the same result as the whole set of real n-bests would.

Bibliography

[1] N. Smith and J. Eisner, "Contrastive estimation: Training log-linear models on unlabeled data," in *Proc. ACL*, 2005.

[2] B. Roark, M. Saraçlar, M. J. Collins, and M. Johnson, "Discriminative language modeling with conditional random fields and the perceptron algorithm," in *Proc. ACL*, 2004, pp. 47–54.

[3] B. Roark, M. Saraçlar, and M. Collins, "Discriminative n-gram language modeling," *Computer Speech and Language*, vol. 21, no. 2, pp. 373 – 392, 2007.

[4] M. Collins, M. Saraçlar, and B. Roark, "Discriminative syntactic language modeling for speech recognition," in *Proc. ACL*, 2005, pp. 507–514.

[5] I. Shafran and K. Hall, "Corrective models for speech recognition of inflected languages," in *Proc. EMNLP*, 2006.

[6] N. Singh-Miller and M. Collins, "Trigger-based language modeling using a loss-sensitive perceptron algorithm," in *Proc. ICASSP*, 2007.

[7] E. Arisoy, M. Saraçlar, B. Roark, and I. Shafran, "Syntactic and sub-lexical features for turkish discriminative language models," in *Proc. ICASSP*, 2010.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[9] M. Collins, "Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms," in *Proc. EMNLP*, 2002, pp. 1–8.

[10] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer, "Online passive-aggressive algorithms," *JOURNAL OF MACHINE LEARNING RESEARCH*, vol. 7, pp. 551–585, 2006.

[11] David Chiang, Yuval Marton, and Philip Resnik, "Online large-margin training of syntactic and structural translation features," in *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, Honolulu, Hawaii, October 2008, pp. 224–233, Association for Computational Linguistics.

[12] David Chiang, Kevin Knight, and Wei Wang, "11,001 new features for statistical machine translation," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, Boulder, Colorado, June 2009, pp. 218–226, Association for Computational Linguistics.

[13] David McAllester, Tamir Hazan, and Joseph Keshet, "Direct loss minimization for structured prediction," in *The 24th Annual Conference on Neural Information Processing Systems (NIPS)*, 2010.

[14] Franz Josef Och, "Minimum error rate training for statistical machine translation," in *Proceedings of the 41st Annual Meeting of the Association of Computational Linguistics (ACL)*, 2003.

[15] E. Arısoy, B. Roark, I. Shafran, and M. Saraçlar, "Discriminative n-gram language modeling for Turkish," in *Proc. Interspeech*, Brisbane, Australia, 2008, pp. 825–828.

[16] Maider Lehr and Izhak Shafran, "Discriminatively estimated joint acoustic, duration and language model for speech recognition," in *Proc. ICASSP*, 2010.

[17] H. Printz and P. Olsen, "Theory and practice of acoustic confusability," *Computer Speech and Language*, vol. 16, no. 1, pp. 131–164, 2002.

[18] J.-Y. Chen, P.A. Olsen, and J.R. Hershey, "Word confusability - measuring hidden Markov model similarity," in *Proc. Interspeech*, 2007.

[19] Q.F. Tan, K. Audhkhasi, P. Georgiou, E. Ettelaie, and S. Narayanan, "Automatic speech recognition system channel modeling," in *Proc. Interspeech*, 2010, pp. 2442–2445.

[20] Zhifei Li, Ziyuan Wang, Sanjeev Khudanpur, and Jason Eisner, "Unsupervised discriminative language model training for machine translation using simulated confusion sets," in *Coling 2010: Posters*, Beijing, China, August 2010, pp. 656–664, Coling 2010 Organizing Committee.

[21] Zhifei Li, Ziyuan Wang, Jason Eisner, Sanjeev Khudanpur, and Brian Roark, "Minimum imputed-risk: Unsupervised discriminative training for machine translation," in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, Edinburgh, Scotland, UK., July 2011, pp. 920–929, Association for Computational Linguistics.

[22] P. Jyothi and E. Fosler-Lussier, "Discriminative language modeling using simulated ASR errors," in *Proc. Interspeech*, 2010, pp. 1049–1052.

[23] G. Kurata, N. Itoh, and M. Nishimura, "Acoustically discriminative training for language models," in *Proc. ICASSP*, 2009.

[24] G. Kurata, N. Itoh, and M. Nishimura, "Training of error-corrective model for ASR without using audio data," in *Proc. ICASSP*, 2011, pp. 5576–5579.

[25] P. Xu, D. Karakos, and S. Khudanpur, "Self-supervised discriminative training of statistical language models," in *Proc. IEEE ASRU*, 2009, pp. 317–322.

[26] Google, "Protocol buffers," `http://code.google.com/apis/protocolbuffers/`.

[27] Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky, "Distant supervision for relation extraction without labeled data," in *ACL-IJCNLP*, 2009.

[28] Ryan McDonald, Keith Hall, and Gideon Mann, "Distributed training strategies for the structured perceptron," in *HLT-NAACL*, 2010.

[29] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *CACM*, vol. 51:1, 2008.

[30] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Christopher J. Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst, "Moses: Open source toolkit for statistical machine translation," in *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, Prague, Czech Republic, June 2007, pp. 177–180, Association for Computational Linguistics.

[31] Zhifei Li, Chris Callison-Burch, Chris Dyer, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese, and Omar Zaidan, "Joshua: An open source toolkit for parsing-based machine translation," in *Proceedings of the Fourth Workshop on Statistical Machine Translation*, Athens, Greece, March 2009, pp. 135–139, Association for Computational Linguistics.

[32] Zhifei Li, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Ann Irvine, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Ziyuan Wang, Jonathan Weese, and Omar Zaidan, "Joshua 2.0: A toolkit for parsing-based machine translation with syntax, semirings, discriminative training and other goodies," in *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, Uppsala, Sweden, July 2010, pp. 139–143, Association for Computational Linguistics.

[33] Jonathan Weese, Juri Ganitkevitch, Chris Callison-Burch, Matt Post, and Adam Lopez, "Joshua 3.0: Syntax-based machine translation with the thrax grammar extractor," in *Proceedings of the Sixth Workshop on Statistical Machine Translation*, Edinburgh, Scotland, July 2011, pp. 478–484, Association for Computational Linguistics.

[34] Chris Dyer, Adam Lopez, Juri Ganitkevitch, Johnathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik, "cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models," in *Proceedings of the ACL 2010 System Demonstrations*, 2010, ACLDemos '10, pp. 7–12.

[35] Hieu Hoang, Philipp Koehn, and Adam Lopez, "A unified framework for phrase-based, hierarchical, and syntax-based statistical machine translation," in *Proceedings of the International Workshop on Spoken Language Translation (IWSLT)*, December 2009, pp. 152–159.

[36] Wei Wang, Kevin Knight, and Daniel Marcu, "Binarizing syntax trees to improve syntax-based machine translation accuracy," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007, pp. 746–754.

[37] Andreas Zollmann, Ashish Venugopal, Matthias Paulik, and Stephan Vogel, "The syntax augmented MT (SAMT) system at the shared task for the 2007 ACL workshop on statistical machine translation," in *Proceedings of the Second Workshop on Statistical Machine Translation*, Prague, Czech Republic, June 2007, pp. 216–219, Association for Computational Linguistics.

[38] David Chiang, Adam Lopez, Nitin Madnani, Christof Monz, Philip Resnik, and Michael Subotin, "The hiero machine translation system: Extensions, evaluation, and analysis," in *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, Vancouver, British Columbia, Canada, October 2005, pp. 779–786, Association for Computational Linguistics.

[39] Liang Huang and David Chiang, "Forest rescoring: Faster decoding with integrated language models," in *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, Prague, Czech Republic, June 2007, pp. 144–151, Association for Computational Linguistics.

[40] Mark Hopkins and Jonathan May, "Tuning as ranking," in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, Edinburgh, Scotland, UK., July 2011, pp. 1352–1362, Association for Computational Linguistics.

[41] Daniel Cer, Daniel Jurafsky, and Christopher Manning, "Regularization and search for minimum error rate training," in *Proceedings of the Third Workshop on Statistical Machine Translation*, Columbus, Ohio, June 2008, pp. 26–34, Association for Computational Linguistics.

[42] George Foster and Roland Kuhn, "Stabilizing minimum error rate training," in *Proceedings of the Fourth Workshop on Statistical Machine Translation*, Athens, Greece, March 2009, pp. 242–249, Association for Computational Linguistics.

[43] Z. Li and S. Khudanpur, "Large-scale discriminative n-gram language models for statistical machine translation," in *Proc. of the Eighth Conference of the Association for Machine Translation in the Americas (AMTA-2008)*, 2008.

[44] Jeff Bilmes and Amar Subramanya, "Parallel graph-based semi-supervised learning," in *Scaling Up Machine Learning*, Ron Bekkerman, Mikhail Bilenko, and John Langford, Eds. Cambridge University Press, 2011, Forthcoming.

[45] D. Karakos, M. Dredze, K. Church, A. Jansen, and S. Khudanpur, "Estimating document frequencies in a speech corpus," in *Proc. of the 2011 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU-2011)*, 2011.

[46] A. K. McCallum, "MALLET: A machine learning for language toolkit," http://mallet.cs.umass.edu, 2002.

[47] I. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors: A multilevel approach," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007.

[48] I. Bulyko, M. Ostendorf, M. Siu, T. Ng, A. Stolcke, and Ö. Çetin, "Web resources for language modeling in conversational speech recognition," *ACM Transactions on Speech and Language Processing (TSLP)*, vol. 5, no. 1, pp. 1–25, 2007.

[49] S. F. Chen, B. Kingsbury, L. Mangu, D. Povey, H. Soltau, and G. Zweig, "Advances in speech transcription at IBM under the DARPA EARS program," *IEEE Trans. on Audio, Speech and Language Processing*, vol. 14, no. 5, 2006.

[50] H. Kuo, E. Fosler-Lussier, H. Jiang, and C. Lee, "Discriminative training of language models for speech recognition," in *Proc.of International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2002, vol. 1, pp. 325–328.

[51] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[52] H. Schwenk, "Continuous space language models," *Computer Speech and Language*, vol. 21, pp. 492–518, 2007.

[53] R. Collobert and ÂăJ. Weston, "A unified architecture for natural language processing: deep neural networks with multitask learning," in *Proc.of ICML 2008*, 2008, pp. 160–167.

[54] Hua Wu and Haifeng Wang, "Pivot language approach for phrase-based statistical machine translation," in *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, Prague, Czech Republic, June 2007, pp. 856–863, Association for Computational Linguistics.

[55] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu, "BLEU: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association of Computational Linguistics (ACL)*, 2002.

[56] David A. Smith and Jason Eisner, "Minimum risk annealing for training log-linear models," in *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, Sydney, Australia, July 2006, pp. 787–794, Association for Computational Linguistics.

[57] David Chiang, "Hierarchical phrase-based translation," *Computational Linguistics*, vol. 33, no. 2, 2007.

[58] Zhifei Li and Jason Eisner, "First- and second-order expectation semirings with applications to minimum-risk training on translation forests," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, Singapore, August 2009, pp. 40–51, Association for Computational Linguistics.

[59] Colin Banrd and Chris Callison-Burch, "Paraphrasing with bilingual parallel corpora," in *43rd Annual Meeting of the Association of Computational Linguistics (ACL)*, 2005.

[60] Nitin Madnani, Necip Fazil Ayan, Philip Resnik, and Bonnie J. Dorr, "Using paraphrases for parameter tuning in statistical machine translation," in *Proceedings of the Second Workshop on Statistical Machine Translation*, Prague, Czech Republic, June 2007, pp. 120–127, Association for Computational Linguistics.

[61] M. Lehr and I. Shafran, "Learning a discriminative weighted finite-state transducer for speech recognition," *IEEE Transaction on Audio, Speech and Language Processing*, vol. 19, no. 5, pp. 1360–1367, 2011.

[62] D. McAllester, L. Gillick, F. Scattone, and M. Newman, "Fabricating conversational speech data with acoustic models: a program to examine model-data mismatch," in *Int'l Conf. on Spoken Language Processing*, 1998.

[63] P. Jyothi and E. Fosler-Lussier, "A comparison of audio-free speech recognition error prediction methods," in *Interspeech*, 2009.

[64] P. A. Olsen and J. R. Hershey, "Bhattacharyya error and divergence using variational importance sampling," in *Interspeech*, 2007.

[65] J. R. Hershey and P. A. Olsen, "Variational Bhattacharyya divergence for hiden markov models," in *Interspeech*, 2008.

[66] J. Oncina and M. Sebban, "Learning stochastic edit distance: Application in handwritten character recognition," *Journal Pattern Recognition*, vol. 39-9, pp. 1575–1587, 2006.

[67] A. Ghoshal, M. Jansche, S. Khudanpur, M. Riley, and M. Ulinski, "Web-derived pronunciations," in *Proc. ICASSP*, 2009.

[68] H. Sak, M. Saraçlar, and T. Güngör, "Discriminative reranking of ASR hypotheses with morpholexical and n-best-list features," in *Proc. ASRU*, 2011.

[69] E. Dikici, M. Semerci, M. Saraçlar, and E. Alpaydın, "Data sampling and dimensionality reduction approaches for reranking ASR outputs using discriminative language models," in *Proc. Interspeech*, 2011, pp. 1461–1464.

[70] E. Arısoy, D. Can, S. Parlak, H. Sak, and M. Saraçlar, "Turkish broadcast news transcription and retrieval," *IEEE Transactions on Audio, Speech & Language Processing*, vol. 17, no. 5, pp. 874–883, 2009.

Feeling MT

by Daniel M. Bikel
August 9th, 2011

(I don't want to be alone, want to be alone…)

I don't have the words
To tell you how much it truly hurts
I don't know where to start
To tell you the secrets of my aching heart
'Cause I'd need a machine
To say what I mean,
That I'm missing you

'Cause I'm feeling

MT
Not a lot of ways to say it
I'm feeling
MT
I'm trying to rephrase it
MT, oh, I'm feeling BLEU

I'm talking' 'bout you,
Dr. Daniel Marcu and your weaver's web
You have the right,
like Kevin Knight,
to say how you feel
And I wouldn't be surprised
if you opened your eyes
and said "I'm alone"

'Cause you're feeling

MT
Not a lot of ways to say it
You're feeling
MT
I'm trying to rephrase it

[Bridge:]
Where is Joshi, now?
His wise words will tell us how
I don't really care
That Phil Resnik has Steven Pinker's hair
I don't really see
How Mitch Marcus can
give another tree
I don't really care
I don't really care, oh no, oh no

I have concern,
like Philipp Koehn,
I'll fall off the beam
Like Chris Callison-Burch
I cannot search
for the one for me

I'm falling into shock
that Franz Och
knows just what I mean
He leaves me all alone
Now I'm on my own
I'm feeling so unseen

CHORUS

(… like David Chiang's probabilistic, synchronous, context-free, tree adjoining, tree substitution, tree insertion, hierarchical grammar...)

LAST CHORUS

Jabberwocky Confusion

Using confusion models developed for ASR modeling during this project, we generated a "confused" version of the poem Jabberwocky by Lewis Carroll. One confusion model used was a weighted finite-state transducer phone confusion model (fst); the other was a machine translation system built to translate from canonical pronunciations of the reference to word strings output by the ASR system (mt). Both are robust to out-of-vocabulary words in the reference text. We reserved poetic license by selecting which confusion model output to include in the final version, and indicate which model each line was taken from. After the command performance of "Feeling MT" by Daniel M. Bikel at the final banquet of the workshop, both the original poem (shown on the left) and the confusion poem (shown on the right) were recited by B. Roark.

Original Poem	**Confusion**	
Twas brillig, and the slithy toves	was braille it and sly see toes	(fst)
Did gyre and gimble in the wabe:	ire and kimble in that way	(fst)
All mimsy were the borogoves,	aw limbs were borough goes	(fst)
And the mome raths outgrabe.	and um rath out great	(fst)
"Beware the Jabberwock, my son!	be way the japanese bur[ning]- walk my son	(mt)
The jaws that bite, the claws that catch!	the jaws that by the clause that catch	(mt)
Beware the Jubjub bird, and shun	be way the jew but jew but bird and one	(mt)
The frumious Bandersnatch!"	the roomie a band other snatch	(mt)
He took his vorpal sword in hand:	took his oral sword in hand	(fst)
Long time the manxome foe he sought –	longtime some so he sought	(fst)
So rested he by the Tumtum tree,	so in he by the two two	(mt)
And stood awhile in thought.	and stood awhile in thought	(mt)
And, as in uffish thought he stood,	and as in uh fish thought he stood	(mt)
The Jabberwock, with eyes of flame,	the japanese bur walk with eyes of flame	(mt)
Came whiffling through the tulgey wood,	came whiff a ling to italy would	(fst)
And burbled as it came!	and herbal as it came	(fst)
One, two! One, two! And through and through	won two won two and through and through	(mt)
The vorpal blade went snicker-snack!	the or people blade went nick other snack	(mt)
He left it dead, and with its head	he left it dead and with it's head	(mt)
He went galumphing back.	he went gal um thing back	(mt)
"And, has thou slain the Jabberwock?	and has thou some lane the japanese bur walk	(mt)
Come to my arms, my beamish boy!	come to my arms my be missionaries boy	(mt)
O frabjous day! Callooh! Callay!'	oh from ab- toxicologist day cal you calais	(mt)
He chortled in his joy.	he chore coddled in his joy	(mt)
Twas brillig, and the slithy toves	was braille it and sly see toes	(fst)
Did gyre and gimble in the wabe:	ire and kimble in that way	(fst)
All mimsy were the borogoves,	aw limbs were borough goes	(fst)
And the mome raths outgrabe.	and um rath out great	(fst)