Ranking and Linear Ordering for System Combination in Machine Translation

Sushant Narsale
Johns Hopkins University
sushant@jhu.edu

Santosh Bahir
Johns Hopkins University
santoshbahir@gmail.com

Abstract

System Combination refers to the method of combining output of multiple MT systems, to produce a output better than each individual system. There have been different approaches for System Combination of Machine Translation (MT) systems to improve translation quality. Currently, there are several approaches to machine translation which can be classified as phrase-based, hierarchial, syntax-based (Hildebrand and Vogel, 2008) which are equally good in their translation quality even though the underlining frameworks are completely different. The motivation behind System Combination arises from this diversity in the state-of-art MT systems, which suggests that systems with different paradigms make different errors, and can be made better by combining strength of these systems. We report results on three approaches to system combination, two based on ranking of translated sentences and one that treats the aligning of systems for combination as a Linear Ordering Problem.

1 Introduction

The basic setup for system combination consists of:

1. M machine translation systems
2. N-best lists of translated sentences(refered to as hypothesis henceforth)[1] for each system.

[1]We will use hypothesis in scenario where there are more than one translations and translation to refer a single best output

In this project we introduce 3 distinct approaches to combining $M * N$ hypothesis to produce a single translation which is better than the individual hypothesis. In the first approach we generate number of features for each hypothesis and train a SVM based ranker to score each translation based on the features. All the hypothesis are then converted in confusion networks which are combined in the order predicted by the ranker to generate a single confusion network. We also use a language model to rescore the paths in this confusion networks, thereby reducing the likeliness of ungrammatical sentences. The best socre path is then picked from the resulting confusion network and used as the final translation. In our second approach we train a log-linear model by using MERT, to obtain weights for each feature. These weights are then used to score each hypothesis and the confusion network combination cycle explained earlier is followed to generate a translation. In the third approach, we first create one confusion network per system by combining confusion networks of N-best outputs generate by individual systems. This confusion network is used as the representative of the system. We use perceptron algorithm to learn Linear ordering problem over the systems, and find the best permutation (or order) in which the systems should be combined.

2 Previous work and Motivation

The task of system combination is to combine $M*N$ outputs and produce a single-best output. Current approaches to system combination can be fall under two categories:

1. Hypothesis Selection or ranking based approaches (Hildebrand and Vogel, 2008) (Hildebrand and Vogel, 2009)

2. Confusion Network based methods (Rosti et.al, 2009) (Karakos and Khudanpur, 2008)

The Hypothesis Selection (Hildebrand and Vogel, 2008) (Hildebrand and Vogel, 2009) merges N-best list from all the M systems and generates features for each hypothesis. A log-linear model is trained on these features, weights obtained are used for ranking each hypothesis and the translation with the best score is returned. We extend this approach in two ways, Firstly we use a SVM based ranking method and instead of returning the 1-best hypothesis we use techniques from Confusion network based methods, to combine all the hypothesis in the order of there scores.

Confusion Network methods (Rosti et.al, 2009) (Karakos and Khudanpur, 2008) have more elaborate and complicated pipelines. These methods generate a confusion network for every hypothesis, these confusion networks are then aligned either using ITG (Wu, 1997) alignments, or to minimize edit distance. We follow the JHU combination (Karakos and Khudanpur, 2008) pipeline for our work. In JHU combination process, confusion networks for all the N hypothesis generated by a system are aligned using edit distance to generate one confusion network per system. These per-system confusion networks are then aligned using ITG to generate one confusion network for every source sentence. The paths in the confusion network are then rescored using language models and the best path from resulting confusion network is treated as the best translation. However the order in which the confusion networks has a major impact on the quality of the final translation as each alignment requires multiple additions, deletions and insertions. In this project we try to formalize a method based on Linear Ordering Problem, to decide the order in which systems should be combined.

3 Ranking Based System Combination

In this section we describe 2 different approaches to ranking of hypothesis from all systems. The setup and pipeline is as follows: We merge the N-best lists from all the M systems for each source sentence, thus for every source sentence S we have $M * N$ hypothesis h. We then generate 15 features (explained in section 3.2) for every hypothesis and run a ranking algorithm to generate scores which are indication of the quality of each hypothesis. Hypothesis for each source sentence are used generate $M * N$ confusion networks. These confusion networks are then aligned in the order of scores obtained from ranking algorithm to generate one confusion network for every S. The resulting confusion networks are rescored with a language model and the resulting best path through the confusion network is taken as the translation for S. [2]

3.1 DataSet and generation of training data

We used GALE speech data for training and test. The broadcast news part of data was used to train the ranking algorithms, the training data consists of 613 source sentences we used 20-best from 4-systems for a total of 80 hypothesis per source sentence. The 613 sentences were divided into two parts of 100 and 513 sentences each and the smaller portion(100 sentences) was used as dev data. The test data consists of 267 source sentences form the broadcast channels portion of GALE speech data. For test data as well we used 20-best from 4 systems. For rescoring of confusion networks and calculating perplexity of sentences we used language model built fom English Gigaword Corpus.

Systems

We used output from four different Arabic-English MT systems part of the GALE project. These four systems 1-4 are ordered by their TER and BLEU scores on our test data table 1.

TER for evaluation and ranking

For generating training data we evaluated every hypothesis using TER and ranked them in the decreasing order of $100-TER$[3]. Thus we trained the ranker to rank documents so as to optimize the TER scores.

[2]For those who are not familiar with MT pipelines in general, Rescoring confusion networks with big Language Models plays an impotant role in the overall quality of the final output. This kind of rescoring reduces weights on paths which are ungrammatical, making them less likely to be generated.

[3]Smaller TER scores indicate better translation

Table 1: Scores for individual systems

System	TER	BLEU
1	51.88	27.37
2	52.85	28.01
3	53.12	26.97
4	53.16	26.91

3.2 Features

We generate 15 features for each hypothesis, these can be categorized in 3 categories:

1. Language Model based features
2. Word and N-gram agreement features
3. Length features

All the features we calculated are based on the individual hypothesis translations and include no information or scores from individual systems. There are two reasons for this, one the scores of individual systems are not comparable and we didn't have an appropriate normalization method. Second it makes our model independent of systems combined. Following is the complete list of features:

Position Dependent Word Agreement

The position dependent word agreement for a hypothesis is defined over every word in the hypothesis. For each word w in position i we calculate the percentage of hypothesis for the same source sentence which have w at position i. Here we try to capture how many hypothesis translations of a source sentence agree with the translation under consideration in terms of words and their positions. We sum over the scores for each word and normalize by the length of hypothesis to calculate per word agreement scores. One needs to be careful when using position dependent features, since in tasks like MT words might be shifted due to insertion or deletion of other words which might be meaningful and gammatical. Hence we generate two more features based on position dependent word agreement, however for these features we relax the definition of position agreement and consider a windows of size 1 and 2 around the word position i.e. For a word w at position i, we consider it i to have a matching word order with other hypothesis even if w is at position $i-1$ or $i+1$ for the feature with window size 1.

Position Independent N-gram Agreements

These features capture the percentage of hypothesis for a source sentence that contain same n-grams as the candidate translation under consideration. The n-gram matching is position independent because phrases often are seen to appear in different orders in sentences with same meaning and correct grammar. The scores for each n-gram are summed and normalized by sentence length. We use n-grams of length $1 \cdots 5$ as five features.

Length Features

The length of translation of sentence to source sentence is a good indication of quality of translation, for a lengthy source sentence a short translation is most likely to be bad. This is the intuition behind the length features. We calculate two length features

1. Ratio of source length to length of target sentence.
2. Difference between length to the candidate translation and meano of other hypothesis translations for the same source sentence.

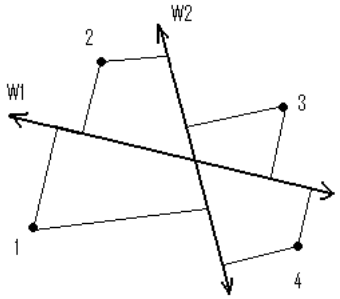
Language Model Features

Language Models are widely used across MT systems to ensure a fluent output translation. We use language models for target language to calculate perplexity of a given transaltion. The lower the perplexity better is the transaltion quality in most cases. We use two different language models and n-grams of length 4 and 5 for a total of 4 features. The first language model we use is version 4 of English gigaword corpus. We generate another language model from all the hypothesis transalation for a source sentence and calculated perplexity. The perplexity values are normalized by sentence length, so that these values are comparable for sentences of varying length.

Rank in original System N-best list

This is the only feature where we use information from original systems. We use the rank of a hypothesis in the N-best list of system which generated it as a feature.

Figure 1: Example of SVM rank (Joachims, 2002)



3.3 SVM for ranking

Our first approach to ranking hypothesis is based on SVM. SVMs are used for binary classication problems. Given some labeled training data learn a separating hyperplane with maximal margin between the two classes. This property of SVM's makes it possible to extend SVM's for ranking with an empirical risk minimization approach as shown in (Joachims, 2002). Kendall's rank correlation coeffecient τ (Kendall and Smith, 1939) is a commonly used measure for determining the simlarity between two ranked sets.[4] The SVM$light$ implementation for ranking maximizes τ. Given a set of rankings, the SVM rank trainer will learn a function that maximizes τ on the training examples.

We formalize the problem as follows:

For a set of hypothesis $h_1, h_2 \cdots h_n$ and a ranking function f
A pairwise ordering (h_i, h_j) is in f if and only if score of $h_i > h_j$

$$(h_i, h_j) \in f \Longleftrightarrow \vec{w}\phi(h_i) > \vec{w}\phi(h_j)$$

where $\vec{w}$ is a weight vector and $\phi(h_i)$ is the feature vector for each hypothesis(In our case the 15 features that we generate).

We found the best explanation of SVM for ranking in (Joachims, 2002) and we explain it based on figure 1 (Joachims, 2002).

Each candidate hypothesis is represented by a vector in n-dimensional space, the figure shows a two-dimensional example. Every hypothesis here is represented by a dot in 2-dimensional space. W_1 and W_2 are two separating hyperplanes. For any weight vector $\vec{W}$ the ranking algorithm orders the points by their projection onto $\vec{W}$. Hence for $\vec{W_1}$ the points are ordered (1,2,3,4) and for $\vec{W_2}$ as (2,3,1,4).

[4]Wikipedia: The Kendall τ distance is a metric that counts the number of pairwise disagreements between two lists.

Scaling in SVM

It has been shown that SVM performance can degrade if scaling is not performed on training data. The main motivation for scaling is to prevent terms of higher magnitude from domination lower magnitude terms. We scale the training data and test data to a range of [-1, +1].

Optimal Parameters and Kernel decision

When using SVM's we need to tune the ξ (slack) parameter. We used RBF kernel with SVM rank and hence had to tune γ parameter. The optimal value we obtained for γ was 0.001. Since the value is small, it indicates that our data is linear, we followed this intuition and used linear SVM ranker as well. The results as reported in the results section show that linear SVM's work better in our case.

3.4 MERT for ranking

MERT (Minimum Error Rate Training) (Och, 2003) is commonly used in training pipeline of many MT systems to find best translation amongst various hypothesis generated by the system. MERT is based on a log-linear model, where for a given source sentence s and a set of hypothesis $h_1, h_2, \cdots h_n$, we chose the hypothesis with highest probability conditioned on s (Zaidan, 2009).
$\grave{h} = argmax_h P(h|s)$
As explained in (Och, 2003) and (Zaidan, 2009), the posterior probability $P(h|s)$ can be modeled by a log-linear model. In this approach we represent each hypothesis h_i with a feature vector ϕ_i and calculate weight vector $\vec{w}$. The weight vector $\vec{w}$ obtained by training the model. The model is trained so as to maximize performance measured by evaluation metric, which should be the metric used to evaluate the performance of the MT system. Intuitively this makes perfect sense, since optimizing model parameters with other metrics (e.g. MAP, F-measure etc..) won't result in optimal weights. We train MERT to optimize for TER scores and used Z-MERT implementation.

MERT optimizes to find weights which would give the best translation given a set of hypothesis. Our intuition was that the weights which give 1-best translation can be used to score n-best hypothesis.

3.5 Implementation Details

Code and Scripts

As mentioned in previous sections we used four systems and 20-best hypothesis from each system. The systems participating in GALE generally output a single text file which consists of $100 \rightarrow 200$ translations for every source sentence. In addition to modifying 4 scripts from the JHU combination GALE pipeline, we wrote 2 scripts to separate data in a format that would be convenient to use throught rest of our pipeline. We also implemented a code to generate all the 15 features we use. Generation of these features was time consuming and hence we had to write a code which was easily parallelizeable, so that we could use the CLSP grid.[5] The feature generation code also includes the interface to the tercom calculation module made available by NIST, which we used to calculate TER scores. It also includes interface to SRILM toolkit (Stockle, 2002), which we used to calculate perplexities under various language models. We implemented an interface for training and ranking using SVM$light$ (SVMlight,). This interface merged translations for all the source sentences, calculated the features for each hypothesis, sorted hypothesis by TER scores and generated a training file in SVM$light$ format (SVMlight,). We used libSVM (Chang and Lin, 2001) implementation of SVM for scaling of our training data and wrote scripts which converted from SVM$light$ to libSVM (Chang and Lin, 2001) formats and back. The interface to SVM$light$ also had implementation for searching for best parameter settings given a set of values for each parameter. We also implemented interface for MERT training and ranking. Lastly, we had to write multiple scripts to include the output we generated by ranking in the JHU combination pipeline.

Packages used

We used the follwing external packages or modules:

1. SRILM toolkit - For generating Language Models and calculating perplexities (Stockle, 2002)

2. SVM$light$ - For ranking of hypothesis. (SVMlight,)

3. ZMERT - To learn weights on features, which were used for scoring and ranking. (Zaidan, 2009)

4. libSVM - For scaling of features. (Chang and Lin, 2001)

5. $tercom$.7.2 - To calculate TER scores for training and evaluations.

Generating Data Sets

Generating training data and evaluation of results was time consuming. Apart from the time taken to generate features and TER evaluations of every hypothesis we had to do multiple system combination using the JHU combination pipeline(which is very elaborate and complicated). As mentioned for training data we used TER for ranking, so we calculated TER for every hypothesis using it reference translation[6]. For test in addition to evaluating individual systems we did entire system combination with the 4-systems we were working with so that we could compare our results with JHU combination.[7]

Problems Faced

SVM ranker was very slow for RBF kernels and as shown in our results section we used both RBF kernel and linear SVM. 613 source sentences and 80 hypothesis translation per sentence meant we were with data size of around 50K. Evaluation with JHU combination pipeline were time consuming.

3.6 RESULTS

In this section we descibe the results obtained. Note that smaller TER and higher BLEU indicate better translations.

[5]The feature generation for training data took two hours when we used 50 cluster nodes

[6]Since we used GALE data reference translations were available to us

[7]This task usually takes a couple of days with 12 systems

System	TER	BLEU
Linear SVM-ranking	51.83	27.43
best individual system	51.88	27.37
JHU combination	50.32	30.13
SVM-ranking(rbf)	52.15	27.15
MERT	55.60	22.65

We used TER for ranking test data, hence the models were trained to minimize TER. The results table compares our approach using linear SVM's, SVM with rbf and MERT with single best system and JHU combination. As mentioned earlier we did a search over a range of slack (ξ) and γ values. The results show that we did better than the best individual system using linear SVM's. JHU combination is amongst the state-of-art methods for system combination and we never expected to get results better than JHU combination. One interesting point to note the difference in TER scores is much better than difference in BLEU scores for JHU combination and linear SVM's. We believe this is because we optimized ranking using TER. While experimenting with rbf kernel for SVM, the γ values that gave best results on dev data were in range of 10^{-3}, smaller γ values indicate that the data is linear this prompted us to try linear SVM's. The results with rbf kernel for SVM's are not good compared to linear SVM's because our data seems to be linear and by using a richer model we tend to overfit the data, also ranking based approaches for system combination are known to do well when the number of systems are more and with a greater n-best list (n = 100). MERT is used to determine 1-best translation from a set of translation, and our intuition that weights that work well for 1-best translation can be used for scoring doesn't look to be correct from the results. We also believe that our feature set was not developed to suit MERT, where features between source and target sentences are more helpful and was one of the reasons for poor results with MERT.

4 Learning Linear Ordering Problem for System Combination

In the confusion network based system described in (Karakos and Khudanpur, 2008)[8] confusion networks for every hypothesis translation are incrementally aligned to each and the resulting confusion network is used to generate the translation. Alignment of two confusion networks involves addition of new words, addition of null arcs for missing words etc. Hence the order in which the confusion networks get aligned has an impact on the final translation generated. Currently in JHU combination process confusion networks for all the hypothesis of individual systems are aligned first to generate one confusion network per system. Individual systems are then ordered based on the TER and BLEU scores calculated on a development set and confusion networks are aligned in the order of scores. This a static approach to ordering where systems are combined in a pre-defined order for all the sentences. In this section we treat the ordering of systems for combination as a linear ordering problem and train a modified perceptron algorithm to get optimal ordering of systems. We adapt the techniques in (Eisner and Tromble, 2006) and (Tromble and Eisner, 2009) to system combination setup.

4.1 Linear Ordering Problem

Given a set of pairwise preference score between a set of nodes (individual systems in our case) π, deriving an optimal orderπ^* is known as Linear Ordering Problem. In the system combination scenario determining the optimal ordering of systems for combination is equivalent to Linear Ordering problem, where given a set of systems $S = s_1, s_2 \cdots s_n$ we would like to develop a model which assigns high score to the optimal ordering π. In Combinatorial literature, it is well known that the Linear Ordering problem is NP-hard. Further there is no polynomial time approximation solution (Eisner and Tromble, 2006). So we need to use heurisitics to determine a close to optimal ordering. The approach we use for approximation of Linear Ordering is based on Local Search which is explained the next section. We also use techniques from language parsing explained in (Eisner and Tromble, 2006) for finding solution in polynomial time.

4.2 Local Search

Local search is an iterative procedure, in each iteration, it improves the solution my making local optimal change (Eisner and Tromble, 2006). In our

[8]We use the incremental alignment method described in (Karakos and Khudanpur, 2008)

Figure 2: Local Search Algorithm

```
1  LocalSearch(B, π, n)
2  for i= 0 to [n−1] do
3     β[i, i + 1] = 0
4     for k =i + 1 to n do
5        △[i, i, k] =△[i, k, k] = 0
6     end for
7  end for
8  for w = 2 to n do
9     for i = 0 to n − w do
10       k =i+w
11       β[i, k] = − ∞
12       for j = i + 1 to k − 1 do
13       △[i, j, k] = △[i, j, k − 1] +△[i + 1, j, k] −△[i + 1, j, k − 1] + B[πk, πi+1 ] − B[πi+1 , πk ]
14        Val = β[i] [j] + β[j] [k] + max(0, △[i] [j] [k])
15        if(β[i][k] < Val):
16              perm[i][k]=[i,j,k]
17        else:
18              perm[i][k]=[i,-1,k]
19        β[i, k] = max(β[i, k], β[i, j] + β[j, k] + max(0, △[i, j, k]))
20        end for
21     end for
22 end for
23 return β[0, n]
24  end localSearch
```

Figure 3: Grammar rules for large neighborhood calculations

$$S \rightarrow S_{0,n}$$
$$S_{i,k} \rightarrow S_{i,j}\ S_{j,k}$$
$$S_{i-1,} \rightarrow \pi_i$$

scenario we start with ordering of systems based on TER-BLEU scores and at each iteration find a locally optimal solution, which is used as ordering for next iteration. If we are given a sequence and pairwise cost of any two elements in it; we need to find the order these element in such a way that the total cost of ordering is minimum. We can use CKY parsing based algorithm based on dynamic programming, to find the best ordering in given imput sequence. To calculate pairwise preference cost, we define Benefit matrix B (Eisner and Tromble, 2006) whose entries are

$$B_w[l, r] = \vec{w}\phi(l, r) \text{ - [5]}$$

where $\vec{w}$ is a vector of weights, we use modified perceptron algorithm to calculate these weights. ϕ is a pairwise feature vector for systems l and r.
It is critical to have a good set of pairwise features for local search to be helpful, we explain our feature set in later section, for now let us assume we have a set of pairwise feature systems and associated weight vectors. We implemented the algorithm in figure 2 for local search based on CKY parsing.

A brief explanation of Local Search algorithm

Here is a brief description of the Local search algorithm in figure 2 that we implemented. For given input permutation π the algorithm computes the nearby optimal permutation with the best score. Set of all such nearby optimal permutations is called as neighborhood $N(\pi)$ of input permutation π. This neighborhood is exponentially large. We can calculate the optimum permuation of π in $N(\pi)$ using dynamic programming. Here, Instead of swapping two elements of sequence, we swap two subsequence of a permuations and try to go towards the optimal permuatation in the $N(\pi)$. Hence this neighborhood of π is also called as twisted-subsequence neighborhood (Tromble and Eisner, 2009). These twisted subsequences of input permutation are essentially Non-terminals in the CKY parsing grammar. The grammar for this algorithm is shown in figure-4. Here each grammar rule is binary rule and Each grammar rule is assigned a score. The score of $S_{i,k} \rightarrow S_{i,j}S_{j,k}$ is $max(0, \Delta[i, j, k])$ where $\Delta[i, j, k]$ is the benefit of swapping the sub-sequence $\pi_{i+1}, \pi_{i+2}, \cdots, \pi_j$ and $\pi_{j+1}, \pi_{j+2}, \cdots, \pi_k$. The grammar rule are considered as swap rule. We would swap the two subsequence if its score is positive which means there is benefit in swapping the two subsequence. The benefit to swapping $\pi_{i+1}, \pi_{i+2}, \cdots, \pi_j$ and $\pi_{j+1}, \pi_{j+2}, \cdots, \pi_k$ is calculated by dynamic programming recurrence.

$$\Delta[i, j, k] = \sum_{l=i+1}^{j} \sum_{r=j+1}^{k} B[\pi_r, \pi_l] - B[\pi_l, \pi_r]$$

The base case for the recurrence is $\Delta[i, j, k] = 0$ if $i = j$ or $j = k$. CKY parsing algorithm uses the weighted grammar rule to obtain highest score parse. From the highest parse we can get the optimal permutation in O(n) time.

This algorithm is modified to keep the back pointers to the choices of maximum as shown in fig-3. We can use this information to find the optimal sequence.

Figure 4: Algorithm for finding best permutation

```
order(i, k, list):
1.  if(list[0] != -1 and list[1] == -1 and list[2] != -1):
2.      for i in range(list[0], list[2]+1):
3.          optPerm.append(π_i)
4.      return optPerm
5.  else if(list[0] == -1 and list[1] == -1 and list[2] == -1):
6.      optPerm.append(π_index1)
7.      if(index1 != index2):
8.          optPerm.append(π_index2])
9.      return optPerm
10.  else:
11.      order(list[1]+1, list[2], perm[list[1]] [list[2]])
12.      order(list[0], list[1], perm[list[0]] [list[1]])
```

4.3 Adaptation of LOP and Local search to System Combination

The order in which systems are combined in a system combination setup has an impact on overall quality of the systems, hence it is important to learn an ordering of systems so as to maximize performance. Doing a brute force search for all N systems is intractable, this makes LOP an ideal choice for learning orders if we can define a set of pairwise features. The pipeline we follow here is, For every system we combine hypothesis for a particular source sentence to generate a single confusion network which is representative of that system. For every source sentence we learn an order in which all the systems should be combined, this is based on pairwise features that we generate for every pair of systems based on their confusion networks. We then combine the systems based on this learned order. Note that here the combination order is different for every source sentence. As usual the final confusion networks are combined with a language model and the resulting best path is considered as the final translation.

4.4 Training with Perceptron

We use Perceptron algorithm for training weights used to calculate benefit scores (equation [5]). The training and predcition phase of perceptron are shown in figure-4.

DataSet and generation of training data

We used GALE speech data for training and test. The broadcast news part of data was used to train the perceptron, the training data consists of 613 source sentences we used 20-best from 3-systems for a total of 60 hypothesis per source sentence. The test data consists of 267 source sentences form the broadcast channels portion of GALE speech data. For test data as well we used 20-best from 3 systems. For every source sentence we combined the confusion networks for the 3 systems in all 3! ways, resulting in 6 labels. Each training segment was then labeled with the combination of systems that gave highest TER scores.

Features

As mentioned earlier the effectiveness of local search depends heavily upon pairwise features. We calculate the following 4 pairwise features[9]

Alignment Costs

For every pair of systems (actually confusion networks representing each system), we calculate the costs of aligning them using ITG (Wu, 1997). Lower alignment costs indicate less number of insertions, deletions and block shifts when the two confusion networks are aligned and can be thought of as the measure of matching between confusion networks.

Number of null arcs in the resulting confusion network

When two confusion networks are combined with ITG alignments a certain number of null arcs may be introduced in the resulting confusion network. This feature counts the number of null arcs in the resulting confusion network.

Score of best path in the resulting confusion network

The best translation obtained from the resulting confusion network has a cost associated with it, which is cost over all the arcs through the confusion which are in the best path.

Likelihood of best path

We score the the best translation obtained from the resulting confusion network with a language model and use the perplexity as a feature.

[9] We also experimented with a larger set of 16 features but results were not encouraging and considering the length of our report we chose not to give details

We also experimented using binary features, where we calculated mean of values for each feature over all training instances and assigned binary values to features based on mean values. However the results with binary features were not encouraging.

Description of Perceptron algorithm

In this section we describe the perceptron algorithm and modifications we made to learn linear ordering problem. Perceptron is an online training algorithm, which observes only one example at a time and makes updates based on that example. This property of online algorithms make them most suitable for our task, since it is more intuitive to predict order of systems for one sentence at a time and update the model best on this prediction. The prediction phase thus involves using local search to predict an ordering of systems based on the benefit matrix calculated from features described above and the weight vector which is learned by the perceptron algorithm.
The training phase is more complicated. We intialize the weight vectors and train for 5 iterations. We observed that number of mistakes that the perceptron algorithm makes for our data generally converges after 4 iterations. The online learning rate is set to 1 and we use a 0-1 loss function. In every iterations we iterate over number of sentences in training data and for each sentence use the predictor to predict the optimal ordering to combine 3 systems. No updates are made if the order predicted matches the actual best order for the sentence. When the predicted order does not match the actual best order we find the number of pairwise orders that we predicted incorrectly. For eg:- if the predicted order was 1,2,3 and the best order is 2,3,1 the number of incorrect orders is 2 (2,1 and 3,1). On incorrect predictions we update the weight vector by penalizing for pairs for which the predicted order was incorrect rewarding for correct ordering. The intuition behind this is that we are trying to learn benefit of ordering one system before another and hence we only penalize for incorrect ordering and reward for orderings which were correct in the incorrectly predicted order. Since we had only three systems we tried a brute force approach to predict best ordering. As expected the results were better than local search. In the brute force approach for every possible (3!) combinations we calculated scores based on feature vectors and weigths (i.e. B) and returned the ordering with highest score.

Figure 5: Modified Perceptron Algorithm

```
1. Perceptron_predictor(B_matrix)
2.       identity_combination = [1,2,3]
3.       best_permutation = Iterative_local_search(B_matrix, identity_combination, n=3)
4.  return best_permuation
5. end Perceptron_predict

1. Perceptron_train(iterations, online_learning)
2.       for i <- 1 to iter do
3.                for j <- 1 to number_of_hypothesis
4.       permutation = hypothesis[j].label
5.       B_matrix = get_matrix(hypothesis[j], systems_list)
6.       predicted_permutation = predict(B_matrix)
7.                         if (permutation != predicted_permutation)
8.                                  wrong_orders = get_wrong_orders(permutation, predicted_permutation)
9.                                  correct_order = get_correct_order(permutation, predicted_permutation)
10.                                 for z <- 1 to len(weight_vector)
11.                                          for pairs in wrong_order:
12.                                          update -= eta *B_matrix[system1][system2]
13.                                          weight_vector += update
14.                                 return weight_vector
15. end Perceptron_train
```

4.5 Implementation Details

Code and Scripts

We adapted most of the JHU system combination pipeline for evaluation and implemented code for the following three tasks

1. Generating features on training data: Code to generate all the features mentioned in the features section. It had wrappers to call alignment code used by JHU combination and AT&T FSM tools.

2. Local Search: We chose to implement local search and not use local search code used by authors of (Eisner and Tromble, 2006), since our model was much simpler and it turned to be less time consuming than adapting the old code. The local search was implemented based on algorithms in fig-2 and fig-3.

3. Perceptron: We implemented the perceptron algorithm in fig-5

Packages used

We used the following packages and code:

1. AT&T FSM toolkit: Used to compose language models and confusion network.

2. ITG alignment code: This code is used in JHU combination pipeline and we used to score alignments.

Generating Data Sets

Generating training data was very very time comsuming and tedious, as we had to try all possible combinations ($3! = 6$)for 3 systems that we used. This was the major reason we decided to experiment with only three systems. We generated training labels using the JHU combination pipeline.

4.6 RESULTS

In this section we review the results for combination with Linear Ordering.[10]

System	TER	BLEU
JHU combination with LOP	50.61	29.67
JHU combination	50.77	29.87
1 best system	51.88	27.37

Again we used TER to determine the best order for hypothesis translations corresponding to a source sentence and hence the model was trained to optimize for TER. The results show that with linear ordering we do better than both the best individual system and JHU combination on TER scores. [11]. The gain compared to individual system is close to 2 points which is considered to be statistically significant. (Hildebrand and Vogel, 2008) However the gain over JHU combination is only marginal. There can be a couple of reasons for this behaviour, one we used only 3 systems and on a bigger set of systems one would expect a bigger gain. Our feature set was small and most of the features were designed to learn preference of keeping a set of systems together rather than ordering them. However a slight improvement compared to JHU combination means there is lot of potential in this approach.

5 Comparison to Project Proposal

We have completed almost everything we proposed to do, and the results we got look promising for future work in these areas. We list the proposed items and their completion status below:

1. Identify feature set: We identified 15 features for phase 1 and 4 features for phase 2. The results using these features are encouraging and we believe can be made better with more feature engineering.

2. Implement a ranking algorithm to rank hypothesis: We had proposed to implement an algorithm for ranking of hypothesis to determine the order in which they should be combined. We completed this phase with SVM rank and also used MERT for ranking.

3. Implement Local search based on CKY parser: We have implemented local search as described in (Tromble and Eisner, 2009)

4. Implement a model based on LOP for ordering of systems: We have results for a model based on LOP, which look good. However we could experiment only with 3 systems. Since the JHU combination pipeline is very elaborate and complicated trying more than 3 systems [12] was infeasible.

5. Implement clustering algorithm to cluster systems before using LOP techniques: Since we knew we would use only 3 systems, we did not do clustering as it would have not helped improve performance or reduce search space.

6 Conclusion and Future work

In this project we presented two different approaches to system combination. Both the approaches we proposed can be integrated seamlessly into the current system combination pipelines. We introduced a novel approach to system combination using Linear Ordering Problem as a first step in combination pipeline. Both the approaches show improvement over the best individual system and the results are encouraging. This leaves a scope for lot of future work on this topics and we plan to pursue these mehtods further. One promising approach would be to use SVM rank as pre processing step to the second approach based on linear ordering. We can use SVM to rank hypothesis of individual systems and use this to generate confusion networks for each system and then use linear ordering to order the systems.

[10] Again note that TER should be as small as possible and BLEU should be bigger

[11] these results are obtained with a brute force search

[12] even 4 would mean $4! = 24$ combinations to generate training data

References

Almut Silja Hildebrand and Stephan Vogel. 2008. *Combination of Machine Translation Systems via Hypothesis Selection from Combined N-Best Lists.*

Almut Silja Hildebrand and Stephan Vogel. 2009. *CMU System Combination for WMT'09*.

Antti-Veikko I. Rosti and Necip Fazil Ayan and Bing Xiang and Spyros Matsoukas and Richard Schwartz and Bonnie J. Dorr 2007. *Combining Outputs from Multiple Machine Translation Systems*.

Damianos Karakos and Sanjeev Khudanpur 2008. *SEQUENTIAL SYSTEM COMBINATION FOR MACHINE TRANSLATION OF SPEECH*.

D. Wu 1997. *Stochastic inversion transduction grammars and bilingual parsing of parallel corpora*.

Och, Franz Josef. 2003. *Minimum error rate training in statistical machine translation.*.

Omar Zaidan 2009. *Z-MERT: A Fully Configurable Open Source Tool for Minimum Error Rate Training of Machine Translation Systems*.

Andreas Stockle. 2002. *Srilm - an extensible language modeling toolkit*.

Thorsten Joachims 2009. *http://svmlight.joachims.org/*.

Chih-Chung Chang and Chih-Jen Lin 2001. *LIBSVM: a library for support vector machines*.

Thorsten Joachims 2002. *Optimizing Search Engines using Clickthrough Data*.

Kendall, M. G.; Babington Smith, B 1939. *The problem of m rankings*.

Jason Eisner and Roy Tromble 2006. *Local Search woth very large-scale neighborhoods fo optimal permutations in machine transalation*.

Roy Tromble and Jason Eisner 2009. *Learning Linear Ordering Problems for Better Translations* .