Sarika Halarnakar - shalarn1
Sindhuula Selvaraju - sselvar4

Machine Translation Assignment 3: Decoder

3.09.16

Contents of the Repository:

1. data : This directory contains the data files given to us and a smaller file we used for testing
2. others: This directory contains other implementations(maybe partial) that we tried out but didn't work very well
3. results: This directory contains result files for the data given to us. This can be directly used with compute_model_score
4. compute_model_score : The original scoring program given to us
5. decode_original : The original decoder file given to us
6. decode_optimization : The file with phrase swapping optimization
7. decode_lagrange : The decoder file with initial Lagrange optimization
8. decode_final : The decoder file that combines decode_optimization and decode_lagrange
9. decode_greedy: The decoder file that tries to implement stack optimization and greedy hill climbing optimization. Though this gives a better log probability the sentences don't make much sense.
10. models.py : The original models file given to us

Part 1: Beam Search Decoder – Swapping Adjacent Phrases

Usage: (In Decoder)

```
python decode_optimization | python compute-model-score
```

Part 2: Beam Search Decoder with Lagrangian Relaxation (with swapping)

Usage: (In Decoder)

```
python decode_final | python compute-model-score
```

You can also try out the version without swapping(decode_lagrangian) but it gives the same final result as the optimization calculations may result in further re-ordering.

Part 3: Greedy Hill Climbing Optimization:

Usage: (In Decoder)

```
python decode_greedy | python compute-model-score
```

- This program takes much longer to run
- It fails to align one of the sentences

Mathematical Description:

We decided to implement the Lagrangian Relaxation. The beam search produces a certificate only if beam pruning is never used. The only way to ensure avoiding pruning is to bound out enough hypotheses. The effectiveness of the bounding inequality in removing hypotheses is directly dependent on the tightness of the bounds. The Lagrangian Relaxation method improves these bounds by relaxing the constraints and solve an unconstrained hypergraph problem with modified weights, unlike in the beam search which solves the constrained search problem. So while the constrained hypergraph problem looks like this:

$$\max_{x \in \mathcal{X}: Ax = b} \theta^\top x + \tau.$$

The Lagrangian dual of the optimization problem looks like this:

$$L(\lambda) = \max_{x \in \mathcal{X}} \theta'^\top x + \tau'$$

where $\lambda \in R^{|b|}$ is a vector of dual variables, $\theta' = \theta - A^\top \lambda$ and $\tau' = \tau + \lambda^\top b$.
This maximization is over X so for any value of lambda, the Lagrangian optimization can be calculated as the $\texttt{BestPathScore}(\theta', \tau')$.

For Phrase-Based Relaxation, we expand out the Lagrangian to

$$L(\lambda) = \max_{x \in \mathcal{X}} \theta^\top x + \tau - \lambda^\top (Ax - b) =$$

$$\max_{x \in \mathcal{X}} \sum_{e \in \mathcal{E}} \left(\theta(e) - \sum_{i=j(p(e))}^{k(p(e))} \lambda_i \right) x(e) + \tau + \sum_{i=1}^{|s|} \lambda_i$$

The weight of each edge ($\theta(e)$) is modified by the dual variables for each source word translated by the edge. A solution under these weights may use source words multiple times or not at all. However, if the solution uses each source word exactly once, then we have a certificate and the solution is optimal. To implement this we added these weights to our original beam-search by adding another loop of length K before analyzing the phrase as such:

procedure LRROUND(α_k, λ)
$x \leftarrow \arg\max_{x \in \mathcal{X}} \theta^\top x + \tau - \lambda^\top (Ax - b)$
$\lambda' \leftarrow \lambda - \alpha_k (Ax - b)$
$\text{opt} \leftarrow Ax = b$
$\text{ub} \leftarrow \theta^\top x + \tau$
return $\lambda', \text{ub}, \text{opt}$
procedure LAGRANGIANRELAXATION(α)
$\lambda^{(0)} \leftarrow 0$
for k in $1 \dots K$ **do**
$\lambda^{(k)}, \text{ub}, \text{opt} \leftarrow \text{LRROUND}(\alpha_k, \lambda^{(k-1)})$
if opt **then return** $\lambda^{(k)}, \text{ub}, \text{opt}$
return $\lambda^{(K)}, \text{ub}, \text{opt}$

We also tried the Greedy Optimization. We follow the algorithm with the source as input from Stack optimization method as follows:

Require: *source* a sentence to translate
current ← `seed`(*source*)
loop
s_current ← `score`(*current*)
s ← *s_current*
for all *h* ∈ `neighborhood`(*current*) **do**
c ← `score`(*h*)
if *c* > *s* **then**
s ← *c*
best ← *h*
if *s* = *s_current* **then**
return *current*
else
current ← *best*

Decoder Type	Corpus Log Probability
Original File	-1439.873990
Phrase Swapping	-1400.775004
Lagrange Alone	-1353.673782

Lagrange with Phrase Swapping	-1353.673782
Greedy - Stack decoder	-1285.780864

The results show that Lagrange Optimization with and without adjacent phrase swapping gives the same log probability. This is because Lagrange model re-arranges the phrases after the phrase swapping mode and gives the same result in both cases.

When using the greedy decoder some sentences(1 sentence in the given data) fails to be aligned for scoring. But, the sentence if seen separately is decoded correctly.