Problem 1. Working with GIZA++ [15 points]
This problem will focus on using the GIZA++ alignment tool, which implements a number of alignment models including Model 1, the HMM Model and Model 4. On patas, it is installed here:

```
/NLP_TOOLS/tool_sets/giza-pp/latest/GIZA++
```

Start with the README file, though you don’t need to compile. If you run GIZA++ with no parameters, it will write a list of options to stdout.

Data preparation

A common representation for parallel data is simple text files, with one segment or sentence per line, and the same number of lines between languages, like clean.en and clean.es from your last assignment. Before training, this data needs to be tokenized and case-normalized (lowercasing is often sufficient).

For GIZA++, however, we need to present the data in a different format, where each vocabulary item has been replaced by a word ID. Additionally, GIZA++ wants as input a list of all the words that co-occur in the same sentence pair – these will be the word pairs that may have a non-zero parameter from Model 1. The tools plain2snt.out and snt2cooc.out generate the files necessary for GIZA++ if you’d like to run them yourself. The first tool, plain2snt.out, takes in a pair of text files, and generates a vocabulary for each, as well as a file where the sentences have been replaced by word IDs. The second tool, snt2cooc.out, identifies the words that co-occur in the same sentence pair. Run them with no arguments for usage.

For the test corpus (same 10K sentences from last week), though, I’ve already done this and stored the files in /dropbox/10-11/575SMT/HW3/prob1

Make a local copy of these files. Then you should be able to run a command like the following to invoke GIZA and then dump the perplexity report.

```
/NLP_TOOLS/tool_sets/giza-pp/latest/GIZA++-v2/GIZA++ config
-S clean.en.vcb -T clean.es.vcb -C clean.en_clean.es.snt
-coocurrencefile enes.cooc -o enes_out && cat enes_out.perp
```

You can edit the configuration file config to change the number of iterations of each model, and swap “es” and “en” to run in the other direction.
For the problem set, use GIZA++ to train a set of word alignment models, and look at the likelihoods and perplexities of each. To indicate iteration counts of particular models, we use superscripts: $1^7 H^5 4^3$ indicates 7 iterations of Model 1, followed by 5 iterations of the HMM model, followed by 3 iterations of Model 4.

(a) [5pts] Report the perplexity (the true perplexity, not Viterbi perplexity) of each of the following nine training regimens. (Note: this should require only 3 GIZA++ runs: in the first run, specify 7 iterations of Model 1 and 0 iterations of the other models, then report only the intermediate perplexities

$$1^3, 1^5, 1^7; \quad 1^5 H^3, 1^5 H^5, 1^5 H^7; \quad 1^5 H^5 4^3, 1^5 H^5 4^5, 1^5 H^5, 4^7$$

I had hoped for students to provide both En-Es and Es-En perplexity. Since this wasn't clear, I didn't mark off if only one direction was provided.

There is also some ambiguity about how to report perplexity. When GIZA++ reports the perplexity at iteration 3 on screen, it is reporting the perplexity during iteration 3, but using the model from iteration 2. Since I wasn’t clear about whether I wanted the perplexity reported during iteration k (the “during” column below), or whether I wanted the perplexity using the model trained in iteration k (the “after” column below), I considered both answers correct. The perplexities in each case are listed below.

<table>
<thead>
<tr>
<th>Regimen</th>
<th>En→Es (during)</th>
<th>En→Es (after)</th>
<th>Es→En (during)</th>
<th>Es→En (after)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1^3$</td>
<td>74.0293</td>
<td>57.9810</td>
<td>71.3765</td>
<td>53.8166</td>
</tr>
<tr>
<td>$1^5$</td>
<td>53.0253</td>
<td>50.9828</td>
<td>48.2943</td>
<td>46.0352</td>
</tr>
<tr>
<td>$1^7$</td>
<td>49.9377</td>
<td>49.3237</td>
<td>44.9125</td>
<td>44.2814</td>
</tr>
<tr>
<td>$1^5 H^3$</td>
<td>24.3173</td>
<td>19.2971</td>
<td>19.1000</td>
<td>15.3325</td>
</tr>
<tr>
<td>$1^5 H^5$</td>
<td>17.5586</td>
<td>16.7428</td>
<td>14.0815</td>
<td>13.4899</td>
</tr>
<tr>
<td>$1^5 H^7$</td>
<td>16.2846</td>
<td>16.0002</td>
<td>13.1507</td>
<td>12.9385</td>
</tr>
<tr>
<td>$1^5 H^5 4^3$</td>
<td>18.5840</td>
<td>17.7749</td>
<td>13.6629</td>
<td>13.0416</td>
</tr>
<tr>
<td>$1^5 H^5 4^5$</td>
<td>17.3537</td>
<td>17.0936</td>
<td>12.7153</td>
<td>12.5089</td>
</tr>
<tr>
<td>$1^5 H^5 4^7$</td>
<td>16.9155</td>
<td>16.7812</td>
<td>12.3729</td>
<td>12.2777</td>
</tr>
</tbody>
</table>

(b) [5pts] As we progress through iterations with the same model, what happens to the perplexity? Why?

The perplexity decreases. **EM is trying to find parameters that optimize the likelihood of the data. It’s a hill climber. During each iteration the likelihood should increase toward a maximum likelihood solution (though it might be a local maximum rather than a global maximum). Perplexity and likelihood are inversely correlated, so the perplexity should decrease as the likelihood increases.**

Intuitively, we’re trying to fit parameters to our data, and the fit gets better as we optimize. Therefore the amount of confusion or perplexity in our data decreases.
(c) [5pts] What happens to perplexity when we shift from one model to the next (say Model 1 to HMM, or HMM to Model 4)? Why?

In moving from Model 1 to the HMM Model, perplexity continues to drop. This model fits the data better. Note that Model 1 can be considered a special case of HMM where the distortion distribution is fixed to be uniform. If instead this distribution is selected to maximize the likelihood of the data, we get a better fit. Therefore, the transition from Model 1 to HMM results in further decreases of perplexity.

Shifting from HMM to Model 4, the parameterization of the model is totally different. We may have been climbing toward a local maximum of the HMM model, but this may not be a locally good point for Model 4. Thus, the perplexity may jump upwards on conversion. Subsequent iterations of Model 4 lead to lower perplexity, though.

Some students noted that the perplexity on the first iteration of Model 4 continues to decrease, but then jumps on the second iteration. In the first iteration, the reported perplexity is from the HMM model rather than Model 4. GIZA++ is first gathering expectations of the possible alignments using the HMM model. Next, it uses these fractional counts to estimate the parameters of Model 4. Only the perplexity from the second iteration is computed according to Model 4, which accounts for the delayed jump.

Problem 2. Decoding complexity [15 points]
Say we start with the following German sentence:

Wiederaufname der Sitzungsperiode

for which we find the following possible phrasal translations:

<table>
<thead>
<tr>
<th>Wiederaufname</th>
<th>der</th>
<th>Sitzungsperiode</th>
</tr>
</thead>
<tbody>
<tr>
<td>resumption</td>
<td>of</td>
<td>session</td>
</tr>
<tr>
<td>recovery</td>
<td>the</td>
<td>of session</td>
</tr>
<tr>
<td>resumption of</td>
<td>of the</td>
<td></td>
</tr>
</tbody>
</table>

(a) [5pts] How many possible derivations are there, allowing all possible reorderings of the input? (Note that this question is asking about derivations, not translations. Multiple derivations may lead to the same string output.)

There are $3! = 6$ possible reorderings, and $3 \times 3 \times 2 = 18$ translation options, leading to $6 \times 18 = 108$ possible derivations.

(b) [5pts] In a monotone search with sufficiently wide beams to prevent pruning of any output and a bigram language model, how many hypotheses would be constructed during search without hypothesis recombination?

For this portion, represent hypotheses as
Then we have the following hypotheses on each stack:

**Stack 0: 1**

1. [000; |]

**Stack 1: 3**

1. [100; |resumption]
2. [100; |recovery]
3. [100; |resumption of]

**Stack 2: 9**

1. [110; |resumption |of]
2. [110; |recovery |of]
3. [110; |resumption of |of]
4. [110; |resumption |the]
5. [110; |recovery |the]
6. [110; |resumption of |the]
7. [110; |resumption of |of the]
8. [110; |recovery |of the]
9. [110; |resumption of |of the]

**Stack 3: 18**

1. [111; |resumption of |session]
2. [111; |recovery of |session]
3. [111; |resumption of |of |session]
4. [111; |resumption the |session]
5. [111; |recovery the |session]
6. [111; |resumption of the |session]
7. [111; |resumption of the |of |session]
8. [111; |recovery of the |session]
9. [111; |resumption of the |of |session]
10. [111; |resumption of |of |session]
11. [111; |recovery of |of |session]
12. [111; |resumption of |of |of |session]
13. [111; |resumption the |of |session]
14. [111; |recovery the |of |session]
15. [111; |resumption of the |of |session]
16. [111; |resumption of the |of |of |session]
17. [111; |recovery of the |of |session]
31 hypotheses are constructed in total. I also gave credit to students listing the number of hypotheses on the final stack (18), or to students who did not count the empty hypothesis.

(c) [5pts] How many hypotheses are constructed with hypothesis recombination?

In this portion, we need to track only the last word of each hypothesis. Since we’re considering a bigram language model, each word need only be scored in the context of the prior word. Therefore, only hypotheses that differ in the last word need be retained.

For this example, I'll number each of the hypotheses, so a hyp will be:

[#;bitmask;lastword]

When

Stack 0: 1 hyp / 1 hyp remaining after recombination

1. [1;000;]

Stack 1: 3 hyps constructed /3 hyps remaining after recombination

1. [2;100;resumption]
   a. [1;000;] + /resumption/
2. [3;100;recovery]
   a. [1;000;] + /recovery/
3. [4;100;of]
   a. [1;000;] + /resumption of/

Stack 2: 9 hyps constructed / 2 hyps remaining after combination

1. [5;110;of]
   a. [2;100;resumption] + /of/
   b. [3;100;recovery] + /of/
   c. [4;100;of] + /of/
2. [6;110;the]
   a. [2;100;resumption] + /the/
   b. [3;100;recovery] + / the /
   c. [4;100;of] + / the /
   d. [2;100;resumption] + /of the/
   e. [3;100;recovery] + / of the /
   f. [4;100;of] + / of the /

Stack 3: 4 hyps constructed / 1 hyp remaining after combination

1. [7;110;session]
1 + 3 + 9 + 4 = 17 hypotheses are constructed in total. There were 1 + 3 + 2 + 1 = 7 equivalence classes after combination. I would accept either answer for full credit. Students that only recombined string-identical hypotheses received partial credit.

Problem 3. Decoding with a phrase based system, Moses [20 points]
A trained moses system is available at: /dropbox/10-11/575SMT/HW3/prob3

You can run the moses decoder with the command

```
/NLP_TOOLS/mt_tools/moses/latest/moses.cmd/src/moses -f model/moses.ini < evaluation/test.es > evaluation/test.en.out
```

You can evaluate the quality of the model using BLEU scores with the script

```
/NLP_TOOLS/mt_tools/moses/latest/moses-scripts/scripts-20100308-1700/generic/multi-bleu.perl
```

(a) [5pts] Decode using different distortion limits: monotone, 1, 3, 5, and unlimited. Report BLEU score and timing. (The option “-dl X” indicates the distortion limit.)

As long as the BLEU scores were correct and the timings were a reasonable multiple of the ones I gathered below, I gave full credit.

<table>
<thead>
<tr>
<th>Limit</th>
<th>Time</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2m18</td>
<td>25.21</td>
</tr>
<tr>
<td>1</td>
<td>2m19</td>
<td>25.21</td>
</tr>
<tr>
<td>3</td>
<td>14m20</td>
<td>27.30</td>
</tr>
<tr>
<td>5</td>
<td>25m30</td>
<td>27.71</td>
</tr>
<tr>
<td>unlimited</td>
<td>156m19</td>
<td>24.67</td>
</tr>
</tbody>
</table>

(b) [5pts] Decode with different T table limits (-ttl). Try 5, 25, and 125. Report BLEU score and timing.

<table>
<thead>
<tr>
<th>Limit</th>
<th>Time</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10m52</td>
<td>27.59</td>
</tr>
<tr>
<td>25</td>
<td>37m28</td>
<td>27.72</td>
</tr>
<tr>
<td>125</td>
<td>56m50</td>
<td>27.72</td>
</tr>
</tbody>
</table>

(c) [5pts] Decode using different stack size limits (-s). Try 5, 25, and 125. Report BLEU score and timing.
(d) [5pts] How does the running time depend on each of these factors?

Since the homework only asked for a small set of numbers, it is difficult to determine the type of correlation with precision. Therefore, full points were assigned to correlations that could be reasonably justified, even if they didn’t exactly match the analysis below.

The effect of distortion limit is approximately linear, with a large coefficient: longer distortion limits lead to a major slowdown in translation speed. Unlimited distortion has greater asymptotic complexity – translation runs in time correlated with the square of the input length.

T-table limits apparently had a sub-linear relationship with runtime: moving from 5 to 25 caused a greater relative slowdown than moving from 25 to 125. Likely this holds because few phrases have more than 25 translations.

The stack size has a nearly linear relationship with runtime. In practice, there should be a log factor as well, because we need to maintain a sorted list of translations. Again, though, with such few data points, this was hard to determine, and grading was flexible.