Learning to translate with products of novices: a suite of open-ended challenge problems for teaching MT

Adam Lopez[1], **Matt Post**[1], **Chris Callison-Burch**[1,2], **Jonathan Weese, Juri Ganitkevitch, Narges Ahmidi, Olivia Buzek, Leah Hanson, Beenish Jamil, Matthias Lee, Ya-Ting Lin, Henry Pao, Fatima Rivera, Leili Shahriyari, Debu Sinha, Adam Teichert, Stephen Wampler, Michael Weinberger, Daguang Xu, Lin Yang,** and **Shang Zhao***
Department of Computer Science, Johns Hopkins University
[1]Human Language Technology Center of Excellence, Johns Hopkins University
[2]Computer and Information Science Department, University of Pennsylvania

Abstract

Machine translation (MT) draws from several different disciplines, making it a complex subject to teach. There are excellent pedagogical texts, but problems in MT and current algorithms for solving them are best learned by doing. As a centerpiece of our MT course, we devised a series of open-ended challenges for students in which the goal was to improve performance on carefully constrained instances of four key MT tasks: alignment, decoding, evaluation, and reranking. Students brought a diverse set of techniques to the problems, including some novel solutions which performed remarkably well. A surprising and exciting outcome was that student solutions or their combinations fared competitively on some tasks, demonstrating that even newcomers to the field can help improve the state-of-the-art on hard NLP problems while simultaneously learning a great deal. The problems, baseline code, and results are freely available.

1 Introduction

A decade ago, students interested in natural language processing arrived at universities having been exposed to the idea of machine translation (MT) primarily through science fiction. Today, incoming students have been exposed to services like Google Translate since they were in secondary school or earlier. For them, MT is science fact. So it makes sense to teach statistical MT, either on its own or as a unit in a class on natural language processing (NLP), machine learning (ML), or artificial intelligence (AI). A course that promises to show students how Google Translate works and teach them how to build something like it is especially appealing, and several universities and summer schools now offer such classes. There are excellent introductory texts—depending on the level of detail required, instructors can choose from a comprehensive MT textbook (Koehn, 2010), a chapter of a popular NLP textbook (Jurafsky and Martin, 2009), a tutorial survey (Lopez, 2008), or an intuitive tutorial on the IBM Models (Knight, 1999b), among many others.

But MT is not just an object of academic study. It's a real application that isn't fully perfected, and the best way to learn about it is to build an MT system. This can be done with open-source toolkits such as Moses (Koehn et al., 2007), cdec (Dyer et al., 2010), or Joshua (Ganitkevitch et al., 2012), but these systems are not designed for pedagogy. They are mature codebases featuring tens of thousands of source code lines, making it difficult to focus on their core algorithms. Most tutorials present them as black boxes. But our goal is for students to learn the key techniques in MT, and ideally to **learn by doing**. Black boxes are incompatible with this goal.

We solve this dilemma by presenting students with concise, fully-functioning, self-contained components of a statistical MT system: word alignment, decoding, evaluation, and reranking. Each implementation consists of a naïve baseline algorithm in less than 150 lines of Python code. We assign them to students as open-ended challenges in which the goal is to improve performance on objective evaluation metrics as much as possible. This setting mirrors evaluations conducted by the NLP research

* The first five authors were instructors and the remaining authors were students in the worked described here. This research was conducted while Chris Callison-Burch was at Johns Hopkins University.

Transactions of the Association for Computational Linguistics, 1 (2013) 165–178. Action Editor: David Chiang.
Submitted 11/2012; Revised 3/2013; Published 5/2013.

community and by the engineering teams behind high-profile NLP projects such as Google Translate and IBM's Watson. While we designate specific algorithms as benchmarks for each task, we encourage creativity by awarding more points for the best systems. As additional incentive, we provide a web-based leaderboard to display standings in real time.

In our graduate class on MT, students took a variety of different approaches to the tasks, in some cases devising novel algorithms. A more exciting result is that some student systems or combinations of systems rivaled the state of the art on some datasets.

2 Designing MT Challenge Problems

Our goal was for students to freely experiment with different ways of solving MT problems on real data, and our approach consisted of two separable components. First, we provided a framework that strips key MT problems down to their essence so students could focus on understanding classic algorithms or invent new ones. Second, we designed incentives that motivated them to improve their solutions as much as possible, encouraging experimentation with approaches beyond what we taught in class.

2.1 Decoding, Reranking, Evaluation, and Alignment for MT (DREAMT)

We designed four assignments, each corresponding to a real subproblem in MT: alignment, decoding, evaluation, and reranking.[1] From the more general perspective of AI, they emphasize the key problems of unsupervised learning, search, evaluation design, and supervised learning, respectively. In real MT systems, these problems are highly interdependent, a point we emphasized in class and at the end of each assignment—for example, that alignment is an exercise in parameter estimation for translation models, that model choice is a tradeoff between expressivity and efficient inference, and that optimal search does not guarantee optimal accuracy. However, presenting each problem independently and holding all else constant enables more focused exploration.

For each problem we provided data, a naïve solution, and an evaluation program. Following Bird et al. (2008) and Madnani and Dorr (2008), we implemented the challenges in Python, a high-level programming language that can be used to write very concise programs resembling pseudocode.[2,3] By default, each baseline system reads the test data and generates output in the evaluation format, so setup required zero configuration, and students could begin experimenting immediately. For example, on receipt of the alignment code, aligning data and evaluating results required only typing:

```
> align | grade
```

Students could then run experiments within minutes of beginning the assignment.

Three of the four challenges also included unlabeled test data (except the decoding assignment, as explained in §4). We evaluated test results against a hidden key when assignments were submitted.

2.2 Incentive Design

We wanted to balance several pedagogical goals: understanding of classic algorithms, free exploration of alternatives, experience with typical experimental design, and unhindered collaboration.

Machine translation is far from solved, so we expected more than reimplementation of prescribed algorithms; we wanted students to really explore the problems. To motivate exploration, we made the assignments competitive. Competition is a powerful force, but must be applied with care in an educational setting.[4] We did not want the consequences of ambitious but failed experiments to be too dire, and we did not want to discourage collaboration.

For each assignment, we guaranteed a passing grade for matching the performance of a specific target algorithm. Typically, the target was important but not state-of-the-art: we left substantial room for improvement, and thus competition. We told students the exact algorithm that produced the target accuracy (though we expected them to derive it themselves based on lectures, notes, or literature). We did not specifically require them to implement it, but the guarantee of a passing grade provided a powerful incentive for this to be the first step of each assignment. Submissions that beat this target received additional credit. The top five submissions received full credit, while the top three received extra credit.

[1] http://alopez.github.io/dreamt

[2] http://python.org

[3] Some well-known MT systems have been implemented in Python (Chiang, 2007; Huang and Chiang, 2007).

[4] Thanks to an anonymous reviewer for this turn of phrase.

This scheme provided strong incentive to continue experimentation beyond the target algorithm.[5]

For each assignment, students could form teams of any size, under three rules: each team had to publicize its formation to the class, all team members agreed to receive the same grade, and teams could not drop members. Our hope was that these requirements would balance the perceived competitive advantage of collaboration against a reluctance to take (and thus support) teammates who did not contribute to the competitive effort.[6] This strategy worked: out of sixteen students, ten opted to work collaboratively on at least one assignment, always in pairs.

We provided a web-based leaderboard that displayed standings on the test data in real time, identifying each submission by a pseudonymous handle known only to the team and instructors. Teams could upload solutions as often as they liked before the assignment deadline. The leaderboard displayed scores of the default and target algorithms. This incentivized an early start, since teams could verify for themselves when they met the threshold for a passing grade. Though effective, it also detracted from realism in one important way: it enabled hill-climbing on the evaluation metric. In early assignments, we observed a few cases of this behavior, so for the remaining assignments, we modified the leaderboard so that changes in score would only be reflected once every twelve hours. This strategy trades some amount of scientific realism for some measure of incentive, a strategy that has proven effective in other pedagogical tools with real-time feedback (Spacco et al., 2006).

To obtain a grade, teams were required to submit their results, share their code privately with the instructors, and publicly describe their experimental process to the class so that everyone could learn from their collective effort. Teams were free (but not required) to share their code publicly at any time. Some did so after the assignment deadline.

[5]Grades depend on institutional norms. In our case, high grades in the rest of class combined with matching all assignment target algorithms would earn a B+; beating two target algorithms would earn an A-; top five placement on any assignment would earn an A; and top three placement compensated for weaker grades in other course criteria. Everyone who completed all four assignments placed in the top five at least once.

[6]The equilibrium point is a single team, though this team would still need to decide on a division of labor. One student contemplated organizing this team, but decided against it.

3 The Alignment Challenge

The first challenge was word alignment: given a parallel text, students were challenged to produce word-to-word alignments with low alignment error rate (AER; Och and Ney, 2000). This is a variant of a classic assignment not just in MT, but in NLP generally. Klein (2005) describes a version of it, and we know several other instructors who use it.[7] In most of these, the object is to implement IBM Model 1 or 2, or a hidden Markov model. Our version makes it open-ended by asking students to match or beat an IBM Model 1 baseline.

3.1 Data

We provided 100,000 sentences of parallel data from the Canadian Hansards, totaling around two million words.[8] This dataset is small enough to align in a few minutes with our implementation—enabling rapid experimentation—yet large enough to obtain reasonable results. In fact, Liang et al. (2006) report alignment accuracy on data of this size that is within a fraction of a point of their accuracy on the complete Hansards data. To evaluate, we used manual alignments of a small fraction of sentences, developed by Och and Ney (2000), which we obtained from the shared task resources organized by Mihalcea and Pedersen (2003). The first 37 sentences of the corpus were development data, with manual alignments provided in a separate file. Test data consisted of an additional 447 sentences, for which we did not provide alignments.[9]

3.2 Implementation

We distributed three Python programs with the data. The first, `align`, computes Dice's coefficient (1945) for every pair of French and English words, then aligns every pair for which its value is above an adjustable threshold. Our implementation (most of

[7]Among them, Jordan Boyd-Graber, John DeNero, Philipp Koehn, and Slav Petrov (personal communication).

[8]http://www.isi.edu/natural-language/download/hansard/

[9]This invited the possibility of cheating, since alignments of the test data are publicly available on the web. We did not advertise this, but as an added safeguard we obfuscated the data by distributing the test sentences randomly throughout the file.

Listing 1 The default aligner in DREAMT: thresholding Dice's coefficient.

```
for (f, e) in bitext:
  for f_i in set(f):
    f_count[f_i] += 1
    for e_j in set(e):
      fe_count[(f_i,e_j)] += 1
  for e_j in set(e):
    e_count[e_j] += 1

for (f_i, e_j) in fe_count.keys():
  dice[(f_i,e_j)] = \
    2.0 * fe_count[(f_i, e_j)] / \
    (f_count[f_i] + e_count[e_j])

for (f, e) in bitext:
  for (i, f_i) in enumerate(f):
    for (j, e_j) in enumerate(e):
      if dice[(f_i,e_j)] >= cutoff:
        print "%i-%i " % (i,j)
```

which is shown in Listing 1) is quite close to pseudocode, making it easy to focus on the algorithm, one of our pedagogical goals. The `grade` program computes AER and optionally prints an alignment grid for sentences in the development data, showing both human and automatic alignments. Finally the `check` program verifies that the results represent a valid solution, reporting an error if not—enabling students to diagnose bugs in their submissions.

The default implementation enabled immediate experimentation. On receipt of the code, students were instructed to align the first 1,000 sentences and compute AER using a simple command.

`> align -n 1000 | grade` By varying the number of input sentences and the threshold for an alignment, students could immediately see the effect of various parameters on alignment quality.

We privately implemented IBM Model 1 (Brown et al., 1993) as the target algorithm for a passing grade. We ran it for five iterations with English as the target language and French as the source. Our implementation did not use null alignment or symmetrization—leaving out these common improvements offered students the possibility of discovering them independently, and thereby rewarded.

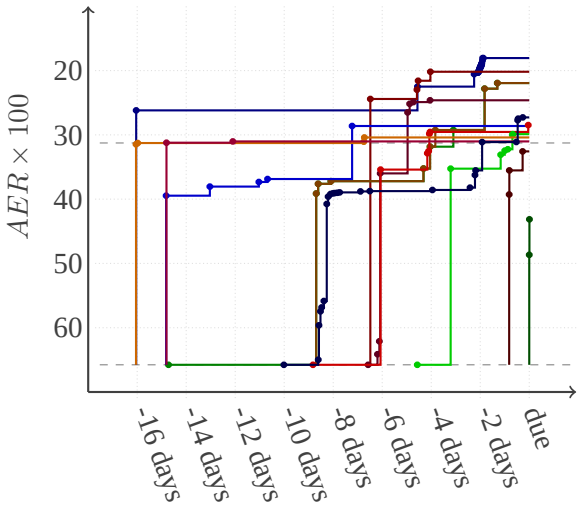

Figure 1: Submission history for the alignment challenge. Dashed lines represent the default and baseline system performance. Each colored line represents a student, and each dot represents a submission. For clarity, we show only submissions that improved the student's AER.

3.3 Challenge Results

We received 209 submissions from 11 teams over a period of two weeks (Figure 1). Everyone eventually matched or exceeded IBM Model 1 AER of 31.26. Most students implemented IBM Model 1, but we saw many other solutions, indicating that many truly experimented with the problem:

- Implementing heuristic constraints to require alignment of proper names and punctuation.
- Running the algorithm on stems rather than surface words.
- Initializing the first iteration of Model 1 with parameters estimated on the observed alignments in the development data.
- Running Model 1 for many iterations. Most researchers typically run Model 1 for five iterations or fewer, and there are few experiments in the literature on its behavior over many iterations, as there are for hidden Markov model taggers (Johnson, 2007). Our students carried out these experiments, reporting runs of 5, 20, 100, and even 2000 iterations. No improvement was observed after 20 iterations.

- Implementing various alternative approaches from the literature, including IBM Model 2 (Brown et al., 1993), competitive linking (Melamed, 2000), and smoothing (Moore, 2004).

One of the best solutions was competitive linking with Dice's coefficient, modified to incorporate the observation that alignments tend to be monotonic by restricting possible alignment points to a window of eight words around the diagonal. Although simple, it acheived an AER of 18.41, an error reduction over Model 1 of more than 40%.

The best score compares unfavorably against a state-of-the-art AER of 3.6 (Liu et al., 2010). But under a different view, it still represents a significant amount of progress for an effort taking just over two weeks: on the original challenge from which we obtained the data (Mihalcea and Pedersen, 2003) the best student system would have placed fifth out of fifteen systems. Consider also the *combined* effort of all the students: when we trained a perceptron classifier on the development data, taking each student's prediction as a feature, we obtained an AER of 15.4, which would have placed fourth on the original challenge. This is notable since *none* of the systems incorporated first-order dependencies on the alignments of adjacent words, long noted as an important feature of the best alignment models (Och and Ney, 2003). Yet a simple system combination of student assignments is as effective as a hidden Markov Model trained on a comparable amount of data (Och and Ney, 2003).

It is important to note that AER does not necessarily correlate with downstream performance, particularly on the Hansards dataset (Fraser and Marcu, 2007). We used the conclusion of the assignment as an opportunity to emphasize this point.

4 The Decoding Challenge

The second challenge was decoding: given a fixed translation model and a set of input sentences, students were challenged to produce translations with the highest model score. This challenge introduced the difficulties of combinatorial optimization under a deceptively simple setup: the model we provided was a simple phrase-based translation model (Koehn et al., 2003) consisting only of a phrase table and trigram language model. Under this simple model, for a French sentence f of length I, English sentence e of length J, and alignment a where each element consists of a span in both e and f such that every word in both e and f is aligned exactly once, the conditional probability of e and a given f is as follows.[10]

$$p(e, a|f) = \prod_{\langle i,i',j,j' \rangle \in a} p(f_i^{i'}|e_j^{j'}) \prod_{j=1}^{J+1} p(e_j|e_{j-1}, e_{j-2}) \quad (1)$$

To evaluate output, we compute the conditional probability of e as follows.

$$p(e|f) = \sum_a p(e, a|f) \quad (2)$$

Note that this formulation is different from the typical Viterbi objective of standard beam search decoders, which do not sum over all alignments, but approximate $p(e|f)$ by $\max_a p(e, a|f)$. Though the computation in Equation 2 is intractable (DeNero and Klein, 2008), it can be computed in a few minutes via dynamic programming on reasonably short sentences. We ensured that our data met this criterion. The corpus-level probability is then the product of all sentence-level probabilities in the data.

The model includes no distortion limit or distortion model, for two reasons. First, leaving out the distortion model slightly simplifies the implementation, since it is not necessary to keep track of the last word translated in a beam decoder; we felt that this detail was secondary to understanding the difficulty of search over phrase permutations. Second, it actually makes the problem more difficult, since a simple distance-based distortion model prefers translations with fewer permutations; without it, the model may easily prefer any permutation of the target phrases, making even the Viterbi search problem exhibit its true NP-hardness (Knight, 1999a; Zaslavskiy et al., 2009).

Since the goal was to find the translation with the highest probability, we did not provide a held-out test set; with access to both the input sentences and

[10] For simplicity, this formula assumes that e is padded with two sentence-initial symbols and one sentence-final symbol, and ignores the probability of sentence segmentation, which we take to be uniform.

the model, students had enough information to compute the evaluation score on any dataset themselves. The difficulty of the challenge lies simply in finding the translation that maximizes the evaluation. Indeed, since the problem is intractable, even the instructors did not know the true solution.[11]

4.1 Data

We chose 48 French sentences totaling 716 words from the Canadian Hansards to serve as test data. To create a simple translation model, we used the Berkeley aligner to align the parallel text from the first assignment, and extracted a phrase table using the method of Lopez (2007), as implemented in cdec (Dyer et al., 2010). To create a simple language model, we used SRILM (Stolcke, 2002).

4.2 Implementation

We distributed two Python programs. The first, `decode`, decodes the test data monotonically—using both the language model and translation model, but without permuting phrases. The implementation is completely self-contained with no external dependencies: it implements both models and a simple stack decoding algorithm for monotonic translation. It contains only 122 lines of Python—orders of magnitude fewer than most full-featured decoders. To see its similarity to pseudocode, compare the decoding algorithm (Listing 2) with the pseudocode in Koehn's (2010) popular textbook (reproduced here as Algorithm 1). The second program, `grade`, computes the log-probability of a set of translations, as outline above.

We privately implemented a simple stack decoder that searched over permutations of phrases, similar to Koehn (2004). Our implementation increased the codebase by 44 lines of code and included parameters for beam size, distortion limit, and the maximum number of translations considered for each input phrase. We posted a baseline to the leaderboard using values of 50, 3, and 20 for these, respectively. We also posted an oracle containing the most probable output for each sentence, selected from among all submissions received so far. The intent of this oracle was to provide a lower bound on the best possible output, giving students additional incentive to continue improving their systems.

[11] We implemented a version of the Lagrangian relaxation algorithm of Chang and Collins (2011), but found it difficult to obtain tight (optimal) solutions without iteratively reintroducing all of the original constraints. We suspect this is due to the lack of a distortion penalty, which enforces a strong preference towards translations with little reordering. However, the solution found by this algorithm is only approximates the objective implied by Equation 2, which sums over alignments.

4.3 Challenge Results

We received 71 submissions from 10 teams (Figure 2), again exhibiting variety of solutions.

- Implementation of greedy decoder which at each step chooses the most probable translation from among those reachable by a single swap or retranslation (Germann et al., 2001; Langlais et al., 2007).
- Inclusion of heuristic estimates of future cost.
- Implementation of a private oracle. Some students observed that the ideal beam setting was not uniform across the corpus. They ran their decoder under different settings, and then selected the most probable translation of each sentence.

Many teams who implemented the standard stack decoding algorithm experimented heavily with its pruning parameters. The best submission used extremely wide beam settings in conjunction with a reimplementation of the future cost estimate used in Moses (Koehn et al., 2007). Five of the submissions beat Moses using its standard beam settings after it had been configured to decode with our model.

We used this assignment to emphasize the importance of good models: the model score of the submissions was generally inversely correlated with BLEU, possibly because our simple model had no distortion limits. We used this to illustrate the difference between model error and search error, including *fortuitous search error* (Germann et al., 2001) made by decoders with less accurate search.

5 The Evaluation Challenge

The third challenge was evaluation: given a test corpus with reference translations and the output of several MT systems, students were challenged to produce a ranking of the systems that closely correlated with a human ranking.

Listing 2 The default decoder in DREAMT: a stack decoder for monotonic translation.

```
stacks = [{} for _ in f] + [{}]
stacks[0][lm.begin()] = initial_hypothesis
for i, stack in enumerate(stacks[:-1]):
  for h in sorted(stack.itervalues(),key=lambda h: -h.logprob)[:alpha]:
    for j in xrange(i+1,len(f)+1):
      if f[i:j] in tm:
        for phrase in tm[f[i:j]]:
          logprob = h.logprob + phrase.logprob
          lm_state = h.lm_state
          for word in phrase.english.split():
            (lm_state, word_logprob) = lm.score(lm_state, word)
            logprob += word_logprob
          logprob += lm.end(lm_state) if j == len(f) else 0.0
          new_hypothesis = hypothesis(logprob, lm_state, h, phrase)
          if lm_state not in stacks[j] or \
            stacks[j][lm_state].logprob < logprob:
              stacks[j][lm_state] = new_hypothesis
winner = max(stacks[-1].itervalues(), key=lambda h: h.logprob)
def extract_english(h):
  return "" if h.predecessor is None else "%s%s " %
    (extract_english(h.predecessor), h.phrase.english)
print extract_english(winner)
```

Algorithm 1 Basic stack decoding algorithm, adapted from Koehn (2010), p. 165.

```
place empty hypothesis into stack 0
for all stacks 0...n − 1 do
  for all hypotheses in stack do
    for all translation options do
      if applicable then
        create new hypothesis
        place in stack
        recombine with existing hypothesis
        prune stack if too big
```

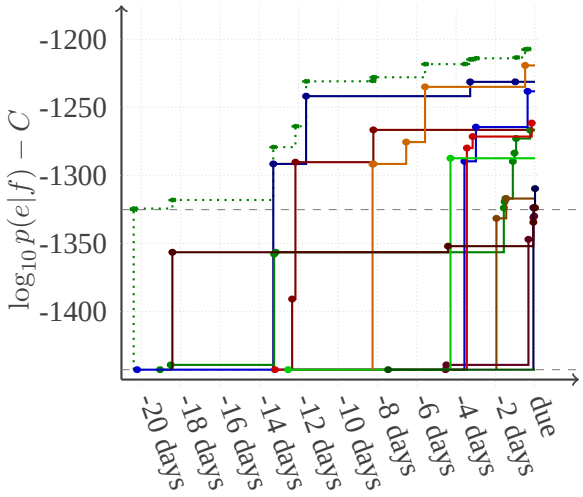

Figure 2: Submission history for the decoding challenge. The dotted green line represents the oracle over submissions.

5.1 Data

We chose the English-to-German translation systems from the 2009 and 2011 shared task at the annual Workshop for Machine Translation (Callison-Burch et al., 2009; Callison-Burch et al., 2011), providing the first as development data and the second as test data. We chose these sets because BLEU (Papineni et al., 2002), our baseline metric, performed particularly poorly on them; this left room for improvement in addition to highlighting some deficiencies of BLEU. For each dataset we provided the source and reference sentences along with anonymized system outputs. For the development data we also provided the human ranking of the sys-

tems, computed from pairwise human judgements according to a formula recommended by Bojar et al. (2011).[12]

5.2 Implementation

We provided three simple Python programs: `evaluate` implements a simple ranking of the systems based on position-independent word error rate (PER; Tillmann et al., 1997), which computes a bag-of-words overlap between the system translations and the reference. The `grade` program computes Spearman's ρ between the human ranking and an output ranking. The `check` program simply ensures that a submission contains a valid ranking.

We were concerned about hill-climbing on the test data, so we modified the leaderboard to report new results only twice a day. This encouraged students to experiment on the development data before posting new submissions, while still providing intermittent feedback.

We privately implemented a version of BLEU, which obtained a correlation of 38.6 with the human rankings, a modest improvement over the baseline of 34.0. Our implementation underperforms the one reported in Callison-Burch et al. (2011) since it performs no tokenization or normalization of the data. This also left room for improvement.

5.3 Evaluation Challenge Results

We received 212 submissions from 12 teams (Figure 3), again demonstrating a wide range of techniques.

- Experimentation with the maximum n-gram length and weights in BLEU.
- Implementation of smoothed versions of BLEU (Lin and Och, 2004).
- Implementation of weighted F-measure to balance both precision and recall.
- Careful normalization of the reference and machine translations, including lowercasing and punctuation-stripping.

[12]This ranking has been disputed over a series of papers (Lopez, 2012; Callison-Burch et al., 2012; Koehn, 2012). The paper which initiated the dispute, written by the first author, was directly inspired by the experience of designing this assignment.

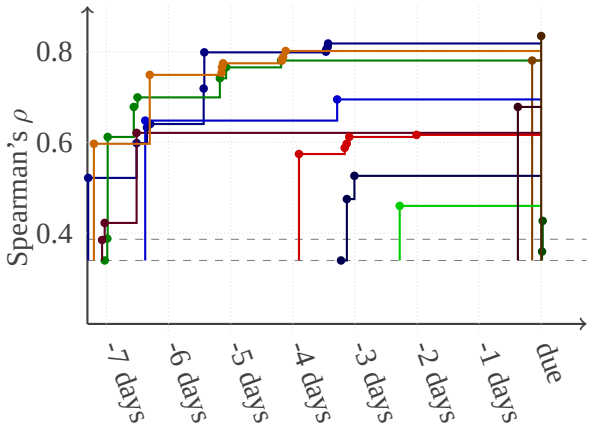

Figure 3: Submission history for the evaluation challenge.

- Implementation of several techniques used in AMBER (Chen and Kuhn, 2005).

The best submission, obtaining a correlation of 83.5, relied on the idea that the reference and machine translation should be good paraphrases of each other (Owczarzak et al., 2006; Kauchak and Barzilay, 2006). It employed a simple paraphrase system trained on the alignment challenge data, using the pivot technique of Bannard and Callison-Burch (2005), and computing the optimal alignment between machine translation and reference under a simple model in which words could align if they were paraphrases. When compared with the 20 systems submitted to the original task from which the data was obtained (Callison-Burch et al., 2011), this system would have ranked fifth, quite near the top-scoring competitors, whose correlations ranged from 88 to 94.

6 The Reranking Challenge

The fourth challenge was reranking: given a test corpus and a large N-best list of candidate translations for each sentence, students were challenged to select a candidate translation for each sentence to produce a high corpus-level BLEU score. Due to an error our data preparation, this assignment had a simple solution that was very difficult to improve on. Nevertheless, it featured several elements that may be useful for future courses.

6.1 Data

We obtained 300-best lists from a Spanish-English translation system built with the Joshua toolkit (Ganitkevitch et al., 2012) using data and resources from the 2011 Workshop on Machine Translation (Callison-Burch et al., 2011). We provided 1989 training sentences, consisting of source and reference sentences along with the candidate translations. We also included a test set of 250 sentences, for which we provided only the source and candidate translations. Each candidate translation included six features from the underlying translation system, out of an original 21; our hope was that students might rediscover some features through experimentation.

6.2 Implementation

We conceived of the assignment as one in which students could apply machine learning or feature engineering to the task of reranking the systems, so we provided several tools. The first of these, `learn`, was a simple program that produced a vector of feature weights using pairwise ranking optimization (PRO; Hopkins and May, 2011), with a perceptron as the underlying learning algorithm. A second, `rerank`, takes a weight vector as input and reranks the sentences; both programs were designed to work with arbitrary numbers of features. The `grade` program computed the BLEU score on development data, while `check` ensured that a test submission is valid. Finally, we provided an `oracle` program, which computed a lower bound on the achievable BLEU score on the development data using a greedy approximation (Och et al., 2004). The leaderboard likewise displayed an oracle on test data. We did not assign a target algorithm, but left the assignment fully open-ended.

6.3 Reranking Challenge Outcome

For each assignment, we made an effort to create room for competition above the target algorithm. However, we did not accomplish this in the reranking challenge: we had removed most of the features from the candidate translations, in hopes that students might reinvent some of them, but we left one highly predictive *implicit* feature in the data: the rank order of the underlying translation system. Students discovered that simply returning the first candidate earned a very high score, and most of them quickly converged to this solution. Unfortunately, the high accuracy of this baseline left little room for additional competition. Nevertheless, we were encouraged that most students discovered this by accident while attempting other strategies to rerank the translations.

- Experimentation with parameters of the PRO algorithm.
- Substitution of alternative learning algorithms.
- Implementation of a simplified minimum Bayes risk reranker (Kumar and Byrne, 2004).

Over a baseline of 24.02, the latter approach obtained a BLEU of 27.08, nearly matching the score of 27.39 from the underlying system despite an impoverished feature set.

7 Pedagogical Outcomes

Could our students have obtained similar results by running standard toolkits? Undoubtedly. However, our goal was for students to learn by doing: they obtained these results by *implementing* key MT algorithms, observing their behavior on real data, and improving them. This left them with much more insight into how MT systems actually work, and in this sense, DREAMT was a success. At the end of class, we requested written feedback on the design of the assignments. Many commented positively on the motivation provided by the challenge problems:

- *The immediate feedback of the automatic grading was really nice.*
- *Fast feedback on my submissions and my relative position on the leaderboard kept me both motivated to start the assignments early and to constantly improve them. Also knowing how well others were doing was a good way to gauge whether I was completely off track or not when I got bad results.*
- *The homework assignments were very engaging thanks to the clear yet open-ended setup and their competitive aspects.*

Students also commented that they learned a lot about MT and even research in general:

Question	1	2	3	4	5	N/A
Feedback on my work for this course is useful	-	-	-	4	9	3
This course enhanced my ability to work effectively in a team	1	-	5	8	2	-
Compared to other courses at this level, the workload for this course is high	-	1	7	6	1	1

Table 1: Response to student survey questions on a Likert scale from 1 (strongly disagree) to 5 (strongly agree).

- *I learned the most from the assignments.*
- *The assignments always pushed me one step more towards thinking out loud how the particular task can be completed.*
- *I appreciated the setup of the homework problems. I think it has helped me learn how to set up and attack research questions in an organized way. I have a much better sense for what goes into an MT system and what problems aren't solved.*

We also received feedback through an anonymous survey conducted at the end of the course before posting final grades. Each student rated aspects of the course on a five point Likert scale, from 1 (strongly disagree) to 5 (strongly agree). Several questions pertained to assignments (Table 1), and allay two possible concerns about competition: most students felt that the assignments enhanced their collaborative skills, and that their open-endedness did not result in an overload of work. For all survey questions, student satisfaction was higher than average for courses in our department.

8 Discussion

DREAMT is inspired by several different approaches to teaching NLP, AI, and computer science. Eisner and Smith (2008) teach NLP using a competitive game in which students aim to write fragments of English grammar. Charniak et al. (2000) improve the state-of-the-art in a reading comprehension task as part of a group project. Christopher et al. (1993) use NACHOS, a classic tool for teaching operating systems by providing a rudimentary system that students then augment. DeNero and Klein (2010) devise a series of assignments based on Pac-Man, for which students implement several classic AI techniques. A crucial element in such approaches is a highly functional but simple scaffolding. The DREAMT codebase, including grading and validation scripts, consists of only 656 lines of code (LOC) over four assignments: 141 LOC for alignment, 237 LOC for decoding, 86 LOC for evaluation, and 192 LOC for reranking. To simplify implementation further, the optional leaderboard could be delegated to Kaggle.com, a company that organizes machine learning competitions using a model similar to the Netflix Challenge (Bennet and Lanning, 2007), and offers pro bono use of its services for educational challenge problems. A recent machine learning class at Oxford hosted its assignments on Kaggle (Phil Blunsom, personal communication).

We imagine other uses of DREAMT. It could be used in an inverted classroom, where students view lecture material outside of class and work on practical problems in class. It might also be useful in massive open online courses (MOOCs). In this format, course material (primarily lectures and quizzes) is distributed over the internet to an arbitrarily large number of interested students through sites such as coursera.org, udacity.com, and khanacademy.org. In many cases, material and problem sets focus on specific techniques. Although this is important, there is also a place for open-ended problems on which students apply a full range of problem-solving skills. Automatic grading enables them to scale easily to large numbers of students.

On the scientific side, the scale of MOOCs might make it possible to empirically measure the effectiveness of hands-on or competitive assignments, by comparing course performance of students who work on them against that of those who do not. Though there is some empirical work on competitive assignments in the computer science education literature (Lawrence, 2004; Garlick and Akl, 2006; Regueras et al., 2008; Ribeiro et al., 2009), they generally measure student satisfaction and retention rather than the more difficult question of whether such assignments actually improve student learning. However, it might be feasible to answer such ques-

tions in large, data-rich virtual classrooms offered by MOOCs. This is an interesting potential avenue for future work.

Because our class came within reach of state-of-the-art on each problem within a matter of weeks, we wonder what might happen with a very large body of competitors. Could real innovation occur? Could we solve large-scale problems? It may be interesting to adopt a different incentive structure, such as one posed by Abernethy and Frongillo (2011) for crowdsourcing machine learning problems: rather than competing, everyone works together to solve a shared task, with credit awarded proportional to the contribution that each individual makes. In this setting, everyone stands to gain: students learn to solve problems as they are found in the real world, instructors learn new insights into the problems they pose, and, in the long run, users of AI technology benefit from overall improvements. Hence it is possible that posing open-ended, real-world problems to students might be a small piece of the puzzle of providing high-quality NLP technologies.

Acknowledgments

We are grateful to Colin Cherry and Chris Dyer for testing the assignments in different settings and providing valuable feedback, and to Jessie Young for implementing a dual decomposition solution to the decoding assignment. We thank Jason Eisner, Frank Ferraro, Yoav Goldberg, Matt Gormley, Ann Irvine, Rebecca Knowles, Ben Mitchell, Courtney Napoles, Michael Rushanan, Joanne Selinski, Svitlana Volkova, and the anonymous reviewers for lively discussion and helpful comments on previous drafts of this paper. Any errors are our own.

References

J. Abernethy and R. M. Frongillo. 2011. A collaborative mechanism for crowdsourcing prediction problems. In *Proc. of NIPS*.

C. Bannard and C. Callison-Burch. 2005. Paraphrasing with bilingual parallel corpora. In *Proc. of ACL*.

J. Bennet and S. Lanning. 2007. The netflix prize. In *Proc. of the KDD Cup and Workshop*.

S. Bird, E. Klein, E. Loper, and J. Baldridge. 2008. Multidisciplinary instruction with the natural language toolkit. In *Proc. of Workshop on Issues in Teaching Computational Linguistics*.

O. Bojar, M. Ercegovčević, M. Popel, and O. Zaidan. 2011. A grain of salt for the WMT manual evaluation. In *Proc. of WMT*.

P. E. Brown, S. A. D. Pietra, V. J. D. Pietra, and R. L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2).

C. Callison-Burch, P. Koehn, C. Monz, and J. Schroeder. 2009. Findings of the 2009 workshop on statistical machine translation. In *Proc. of WMT*.

C. Callison-Burch, P. Koehn, C. Monz, and O. Zaidan. 2011. Findings of the 2011 workshop on statistical machine translation. In *Proc. of WMT*.

C. Callison-Burch, P. Koehn, C. Monz, M. Post, R. Soricut, and L. Specia. 2012. Findings of the 2012 workshop on statistical machine translation. In *Proc. of WMT*.

Y.-W. Chang and M. Collins. 2011. Exact decoding of phrase-based translation models through Lagrangian relaxation. In *Proc. of EMNLP*.

E. Charniak, Y. Altun, R. de Salvo Braz, B. Garrett, M. Kosmala, T. Moscovich, L. Pang, C. Pyo, Y. Sun, W. Wy, Z. Yang, S. Zeiler, and L. Zorn. 2000. Reading comprehension programs in a statistical-language-processing class. In *Proc. of Workshop on Reading Comprehension Tests as Evaluation for Computer-Based Language Understanding Systems*.

B. Chen and R. Kuhn. 2005. AMBER: A modified BLEU, enhanced ranking metric. In *Proc. of WMT*.

D. Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2).

W. A. Christopher, S. J. Procter, and T. E. Anderson. 1993. The nachos instructional operating system. In *Proc. of USENIX*.

J. DeNero and D. Klein. 2008. The complexity of phrase alignment problems. In *Proc. of ACL*.

J. DeNero and D. Klein. 2010. Teaching introductory articial intelligence with Pac-Man. In *Proc. of Symposium on Educational Advances in Artificial Intelligence*.

L. R. Dice. 1945. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302.

C. Dyer, A. Lopez, J. Ganitkevitch, J. Weese, F. Ture, P. Blunsom, H. Setiawan, V. Eidelman, and P. Resnik. 2010. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proc. of ACL*.

J. Eisner and N. A. Smith. 2008. Competitive grammar writing. In *Proc. of Workshop on Issues in Teaching Computational Linguistics*.

A. Fraser and D. Marcu. 2007. Measuring word alignment quality for statistical machine translation. *Computational Linguistics*, 33(3).

J. Ganitkevitch, Y. Cao, J. Weese, M. Post, and C. Callison-Burch. 2012. Joshua 4.0: Packing, PRO, and paraphrases. In *Proc. of WMT*.

R. Garlick and R. Akl. 2006. Intra-class competitive assignments in CS2: A one-year study. In *Proc. of International Conference on Engineering Education*.

U. Germann, M. Jahr, K. Knight, D. Marcu, and K. Yamada. 2001. Fast decoding and optimal decoding for machine translation. In *Proc. of ACL*.

L. Huang and D. Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proc. of ACL*.

M. Johnson. 2007. Why doesn't EM find good HMM POS-taggers? In *Proc. of EMNLP*.

D. Jurafsky and J. H. Martin. 2009. *Speech and Language Processing*. Prentice Hall, 2nd edition.

D. Kauchak and R. Barzilay. 2006. Paraphrasing for automatic evaluation. In *Proc. of HLT-NAACL*.

D. Klein. 2005. A core-tools statistical NLP course. In *Proc. of Workshop on Effective Tools and Methodologies for Teaching NLP and CL*.

K. Knight. 1999a. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4).

K. Knight. 1999b. A statistical MT tutorial workbook.

P. Koehn, F. J. Och, and D. Marcu. 2003. Statistical phrase-based translation. In *Proc. of NAACL*.

P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proc. of ACL*.

P. Koehn. 2004. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Proc. of AMTA*.

P. Koehn. 2010. *Statistical Machine Translation*. Cambridge University Press.

P. Koehn. 2012. Simulating human judgment in machine translation evaluation campaigns. In *Proc. of IWSLT*.

S. Kumar and W. Byrne. 2004. Minimum bayes-risk decoding for statistical machine translation. In *Proc. of HLT-NAACL*.

P. Langlais, A. Patry, and F. Gotti. 2007. A greedy decoder for phrase-based statistical machine translation. In *Proc. of TMI*.

R. Lawrence. 2004. Teaching data structures using competitive games. *IEEE Transactions on Education*, 47(4).

P. Liang, B. Taskar, and D. Klein. 2006. Alignment by agreement. In *Proc. of NAACL*.

C.-Y. Lin and F. J. Och. 2004. ORANGE: a method for evaluating automatic evaluation metrics for machine translation. In *Proc. of COLING*.

Y. Liu, Q. Liu, and S. Lin. 2010. Discriminative word alignment by linear modeling. *Computational Linguistics*, 36(3).

A. Lopez. 2007. Hierarchical phrase-based translation with suffix arrays. In *Proc. of EMNLP*.

A. Lopez. 2008. Statistical machine translation. *ACM Computing Surveys*, 40(3).

A. Lopez. 2012. Putting human assessments of machine translation systems in order. In *Proc. of WMT*.

N. Madnani and B. Dorr. 2008. Combining open-source with research to re-engineer a hands-on introductory NLP course. In *Proc. of Workshop on Issues in Teaching Computational Linguistics*.

I. D. Melamed. 2000. Models of translational equivalence among words. *Computational Linguistics*, 26(2).

R. Mihalcea and T. Pedersen. 2003. An evaluation exercise for word alignment. In *Proc. on Workshop on Building and Using Parallel Texts*.

R. C. Moore. 2004. Improving IBM word alignment model 1. In *Proc. of ACL*.

F. J. Och and H. Ney. 2000. Improved statistical alignment models. In *Proc. of ACL*.

F. J. Och and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29.

F. J. Och, D. Gildea, S. Khudanpur, A. Sarkar, K. Yamada, A. Fraser, S. Kumar, L. Shen, D. Smith, K. Eng, V. Jain, Z. Jin, and D. Radev. 2004. A smorgasbord of features for statistical machine translation. In *Proc. of NAACL*.

K. Owczarzak, D. Groves, J. V. Genabith, and A. Way. 2006. Contextual bitext-derived paraphrases in automatic MT evaluation. In *Proc. of WMT*.

K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proc. of ACL*.

L. Regueras, E. Verdú, M. Verdú, M. Pérez, J. de Castro, and M. Muñoz. 2008. Motivating students through on-line competition: An analysis of satisfaction and learning styles.

P. Ribeiro, M. Ferreira, and H. Simões. 2009. Teaching artificial intelligence and logic programming in a competitive environment. *Informatics in Education*, (Vol 8_1):85.

J. Spacco, D. Hovemeyer, W. Pugh, J. Hollingsworth, N. Padua-Perez, and F. Emad. 2006. Experiences with marmoset: Designing and using an advanced submission and testing system for programming courses. In *Proc. of Innovation and technology in computer science education*.

A. Stolcke. 2002. SRILM - an extensible language modeling toolkit. In *Proc. of ICSLP*.

C. Tillmann, S. Vogel, H. Ney, A. Zubiaga, and H. Sawaf. 1997. Accelerated DP based search for statistical translation. In *Proc. of European Conf. on Speech Communication and Technology*.

M. Zaslavskiy, M. Dymetman, and N. Cancedda. 2009. Phrase-based statistical machine translation as a traveling salesman problem. In *Proc. of ACL*.