PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009 27–36

PostCAT – Posterior Constrained Alignment Toolkit

João Graça, Kuzman Ganchev, Ben Taskar

Abstract

In this paper we present a new open-source toolkit for statistical word alignments - Posterior Constrained Alignment Toolkit (PostCAT). The toolkit implements three well known word alignment algorithms (IBM M1, IBM M2, HMM) as well as six new models. In addition to the usual Viterbi decoding scheme, the toolkit provides posterior decoding with several flavors for tuning the threshold. The toolkit also provides an implementation of alignment symmetrization heuristics and a set of utilities for analyzing and pretty printing alignments. The new models have already been shown to improve intrinsic alignment metrics and also to lead to better translations when integrated into a state of the art machine translation system. The toolkit is developed in Java and available in source at its website [1]. We encourage other researchers to build on our work by modifying the toolkit and using it for their research.

1. Motivation

Word alignments are a valuable resource for several areas of natural language processing but especially for statistical machine translation (SMT) as they are a first step in most SMT pipelines. There has been a large quantity of research on improving word alignment models, the impact of different word alignments on SMT quality, and how to better use word alignments to extract the minimal units used in SMT systems such as phrases or rules. The vast majority of this work has unfortunately been done on in-house systems not released publicly with the result that researchers often have to re-implement previous work before they can improve upon it. The notable exception is the widely used GIZA++ toolkit (Och and Ney, 2003). Unfortunately there have been many improvements since the toolkit's publication, and the toolkit does not have many of the components for easy analysis of the alignment results.

We address this gap by introducing a new open-source toolkit - Posterior Constrained Alignment Toolkit (PostCAT) - that implements improved alignment models with results proven

[1] http://www.seas.upenn.edu/~strctlrn/CAT/CAT.html

Please cite this article as: João Graça, Kuzman Ganchev, Ben Taskar, PostCAT – Posterior Constrained Alignment Toolkit. The Prague Bulletin of Mathematical Linguistics No. 91, 2009, 27–36.

to boost SMT performance, as well as a set of utilities for the investigation of the effect of word alignment in overall SMT systems. The new models are trained with a new training procedure, called Constrained Expectation Maximization (Graça, Ganchev, and Taskar, 2008) that allows the user to specify prior information such as "alignments should be symmetric" or "each word should align to at most one word".

This toolkit has been used by us in three peer-reviewed publications. The procedure was introduced by Graça et al. (2008) and shown to improve intrinsic measures of alignment quality. Ganchev et al. (2008) show that the new word alignments and posterior decoding improve overall SMT quality on 6 different language pairs for different training sizes. The new models lead to an significant improvement (as measured by BLEU (Papineni et al., 2002)) in 16 out of 18 test cases. Finally the pretty printing and alignment statistics code was used in Graça et al. (2008) where golden sets of word alignments were produced for all combination of 4 different languages.

This paper is organized as follows. Section 2 gives a brief presentation of the models implemented in the framework. Section 3 describes the code organization, and some easy access points for improvement. Section 4 describes how to use the code to replicate the results in our previous papers. Section 5 describes related work and Section 6 concludes the paper.

2. Word Alignments Models

This section briefly describes the models implemented in the PostCAT. More detailed descriptions are available in (Brown et al., 1994), (Vogel, Ney, and Tillmann, 1996) and (Graça, Ganchev, and Taskar, 2008) along with derivations. Our goal here is to include enough details to make the next section easier to understand.

2.1. Baseline Models

The "baseline" models are the well know IBM Model 1, IBM Model 2 (Brown et al., 1994) and the HMM model proposed by (Vogel, Ney, and Tillmann, 1996). The three models can be expressed as:

$$p(\mathbf{t}, \mathbf{a} \mid \mathbf{s}) = \prod_j p_d(a_j|j, a_{j-1}) p_t(\mathbf{t}_j|\mathbf{s}_{a_j}), \tag{1}$$

with the three differing in their definition of the distortion probability $p_d(a_j|j, a_{j-1})$. Model 1 assumes that the positions of the words are not important and assigns uniform distortion probability. Model 2 allows a dependence on the positions $p_d(a_j|j, a_{j-1}) = p_d(a_j|j)$ and the HMM model assumes that the only the distance between the current and previous source word are important $p_d(a_j|j, a_{j-1}) = p_d(a_j|a_j - a_{j-1})$. All the models are augmented by adding a special "null" word to the source sentence. The likelihood of the corpus, marginalized over possible alignments is concave for Model 1, but not for the other models (Brown et al., 1994).

All these models we consider are normally trained using the Expectation Maximization (EM) algorithm (Dempster, Laird, and Rubin, 1977). The EM algorithm attempts to maximize the marginal likelihood of the observed data ($\mathbf{s}$, $\mathbf{t}$ pairs) by repeatedly finding a maximal lower

bound on the likelihood and finding the maximal point of the lower bound. The lower bound is constructed by using posterior probabilities of the hidden alignments ($\mathbf{a}$) and can be optimized in closed form from expected sufficient statistics computed from the posteriors. The posteriors are hardest to compute for the HMM alignment model but can be efficiently calculated by the forward-backward algorithm.

2.2. New Models

A well known problem when using the EM algorithm is the potential to be stuck in local maxima of the likelihood function. More importantly for word alignment, the models are very gross over-simplifications of the world and the optimal likelihood might not correspond to the optimal model parameters or optimal alignments. It has been shown that increases in likelihood can actually decrease alignment performance. The obvious solution to this problem is to bring the model closer to a faithful representation of the real world. This is the approach taken by IBM models 4+ (Brown et al., 1994) and by (Liang, Taskar, and Klein, 2006) who introduce an agreement component into the models with an intent similar to the one we address. Unfortunately, these changes can make the models intractable, requiring an approximation, often without any approximation guarantees. Graça et al. (2008) introduce an augmentation of the EM algorithm where the model remains unchanged, but the posteriors used during learning are constrained to be "meaningful." This has the advantage of allowing tractable inference while encoding prior knowledge that would be complicated to encode directly in the model.

The PostCAT provides training procedures that augment the baseline models by imposing constrains on the posteriors that roughly encode the following intuitions: "one word should not translate to many words" and "translation is approximately symmetric." We call the former "substochastic" and the latter "agreement." The class names in the PostCAT reflect this terminology. For example "SubstochasticM1" is IBM model one trained with the constraint that each source word should generate at most one target word in expectation. The original paper has a more detailed description of the constrained EM framework.

2.3. Decoding

For each sentence pair, the alignment models define a distribution over alignments. For use in an MT pipeline, we need to extract a single alignment from this distribution. The standard approach, called Viterbi decoding, is to choose the most probable alignment according to the model. Another possibility, that sometimes works better (Liang, Taskar, and Klein, 2006, Graça, Ganchev, and Taskar, 2008, Ganchev, Graça, and Taskar, 2008) is to include the alignment $i - j$ if the posterior probability that word i aligns to word j is above some threshold θ. This allows the accumulation of probability from several low-scoring alignments that agree on one point. This accumulated probability is easily extracted from the model posteriors. Note that this could potentially result in an alignment having zero probability under the model that generated it. PostCAT implements both decoding strategies with a variety of ways to tune θ, either by maximing/minimizing some intrinsic alignment metric such as alignment error rate

(AER (Och and Ney, 2003)) or (balanced) F-1 (Fraser and Marcu, 2007) with respect to a hand annotated corpus, or using a heuristic when aligned data is not available. One effective heuristic is to tune the threshold to have roughly the same number of points as Viterbi decoding. This trades less confident points for more confident ones.

2.4. Symmetrization

The word alignment models described above are asymmetric while most most applications including SMT require a single alignment for each sentence pair. Typically this gap is bridged by applying a symmetrization heuristic that takes as input two directional alignments and produces a single "consensus" alignment. The PostCAT implements the 4 most common alignment heuristics (Och and Ney, 2003): Intersection, Grow Diag, Grow Diag Final and Union. Should the user want another heuristic, implementing one in the PostCAT is very straightforward.

3. Code Organization

The code is organized around 4 main packages: The `corpus` that contains a representation of a sentence-aligned bilingual corpus, the `alignment` package containing a representation of word alignments, symmetrization heuristics, evaluation code as well as code to output alignments in machine-readable or human-readable form as well as collect statistics about alignments. The `models` package which contains the implementation the algorithms for training and extracting alignments from the models. Finally the `programs` package contains code to reproduce experiments from previous work, and example applications such as training and saving models, decoding with models and producing detailed reports of model performance. We describe each package in mode detail.

3.1. The `alignment` package

A word alignment, introduced by (Brown et al., 1994), consists of an object representing which words in a source language correspond to translations of other words in a target language, between two parallel sentences. Figure 1 shows an example of a word alignment represented as a matrix. An English sentence of length 8 (the rows of the matrix) is the translation of a Spanish sentence of length 9 (the columns of the matrix). Each entry of the matrix a_{ij} contains information about whether the i^{th} English word is the translation of the j^{th} Spanish word. The `Alignment` class stores this information and contains the identity of the respective sentences in the corpus and only contains the identities of the corresponding words. To save space, the `String` representations of the words are stored separately.

In addition to these identities, the `Alignment` class contains a matrix represent the word alignment. Each entry in the matrix takes one of several values indicating if the point from the gold standard, is obtained through decoding or has been added by a symmetrization heuristic. The `Alignment` class also stores posterior alignment probability of each word pair. This is

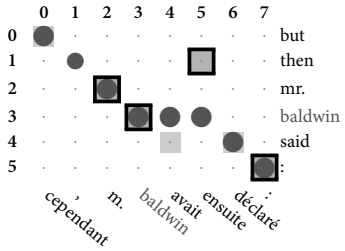

Figure 1. Example of a word alignment

useful for visualizations to understand how the alignment model works, as well as for more sophisticated phrase extraction techniques. In our pretty-printer posterior probabilities are represented by the size of the ball for each alignment entry. The class provides output methods for the most widely used word alignment formats (NACCL workshops, Giza++, Moses (Koehn et al., 2007)). Human readable output is implemented in `AlignerOutput` for plain text and `AlignerOutputLatex` for LaTeX. The latter was used to generate Figure 1.

The `AlignmentEvaluator` class implements the metrics for evaluating an alignment or group of alignments with respect to a gold standard. Currently implemented are precision, recall, AER , F-1, balanced F1 as well as methods to compute the number and type of unaligned points, useful in comparing different models. `AlignmentSymetrization` implements the symmetrization heuristics for word alignments: Intersection, Union, Grow Diagonal and Grow Diagonal Final.

3.2. The `corpus` package

The major class in the `corpus` package is `BilingualCorpus` which represents a parallel corpus, including any testing and development hand-aligned data. The relevant information about a corpus, such as file locations, existance of hand-aligned data and sentence length cutoffs is normally read from a plain-text configuration file. We found this to be much easier to use than passing this information piecemeal to our executables.

The class creates a dictionary for each language mapping string representations to integers to reduce the memory footprint of the program. It also collects statistics about individual words and creates a dictionary of word pairs that occur in the same sentence, also used for improved efficiency (both time and size) when training the models. There is also an option to load only part of the corpus for experiments with part of the training data.

3.3. The `model` package

The word alignment algorithms in the framework are implemented inside the `model` package. We implement IBM Model 1 and IBM Model 2 and the hidden Markov word alignment model, as well as constrained E-M version of these models. The constraints implemented

for the constrained E-M versions are "substochastic constraints" and "agreement constraints." Substochastic constraints capture the intuition that one word on one language should not align to many words in the other language (this mitigates the well known garbage collector effect (Brown et al., 1993)). Agreement constraints capture the intuition that alignment should be a roughly symmetric process so aligning with one language in source position should give the same results as aligning with that language in target position. Each of these 9 models is in a separate class. There are also some classes that efficiently represent model parameters that are shared between models. The models can be initialized with default parameters or with parameters from other models in order to boostrap complicated models from simpler ones. For example, it is standard practice to initialize the hidden Markov model with the translation table from IBM Model 1. The models can use smoothing add-n, specified at creation time.

3.4. The `programs` package

The `programs` package contains classes with `main` methods and can be used to reproduce the results reported in (Graça, Ganchev, and Taskar, 2008) and (Ganchev, Graça, and Taskar, 2008) without programming. Additionally there are illustrative examples of how to use the framework as a library. Some useful classes in the `programs` package:

- `SaveModels` - Trains a given model on a given corpus and save the trained model.
- `ComputeAlignmentError` - Loads a trained model and evaluates it against hand annotated data for a user-specified decoding scheme.
- `AlignmentsForMoses` - Loads a trained model and saves the alignments for different symmetrization heuristics in a format usable by the Moses script. To use Moses with our alignments, call the moses training script with `--firstStep 4`.
- `PrettyPrintAlignments` - Pretty prints the test set alignments for a given model and decoding scheme. We found this very useful for getting an understating of model behavior.

4. How to use PostCAT

This section describes how to use the PostCAT as a program to reproduce our results as well as how to extend it.

4.1. Getting and Installing the toolkit

The toolkit can be downloaded from its website as a gzipped tar file. It is implemented in Java and uses the GNU Trove library [2] which is distributed with the toolkit source code. The toolkit includes an Apache Ant[3] buildfile, so compiling it should just require running `ant`.

[2] http://trove4j.sourceforge.net/

[3] http://ant.apache.org/

4.2. Included Example

Included with the code is some sample data in the `small_data/` sub-directory. We have written some `bash` scripts as illustrations of how to use the code. To run these, you will need some auxiliary programs (e.g. `tee`, `find`, `tail`) that are standard on *NIX systems. If you don't have them, it should be easy to change the scripts or just copy the commandline. The small_data directory contains a corpus parameters file called `small_hansards.params`. In addition to the comments, the file has the following entries:

- Source and target language suffixes. For English, French we have 'en', 'fr'.
- A training, word-alignment development and test file bases. The files have '.en' appended for the English side and '.fr' for the French language.
- A name for the corpus. This is used when creating output directories.

We include scripts that we found useful when running our experiments. One script trains the hidden Markov alignment models (both baseline and agreement) and saves them. Another script computes alignment error metrics for each model. A third included script outputs alignments in a format accepted by the Moses SMT system. The package `README` file has more details on how to run the scripts and how to integrate the output with Moses.

To reproduce the results reported in (Graça, Ganchev, and Taskar, 2008) and (Ganchev, Graça, and Taskar, 2008) one needs to obtain the corpora separately.

4.3. Extending the toolkit

It is fairly straightforward to extend the toolkit with your own alignment model, or to produce statistics you might be interested in. In this subsection we will explain how to write the HMM with substochastic constraints, as an example for how to extend the toolkit with new models. We wrote a class `SubstochasticHMM extends HMM` to implement the model. In the interest of space we do not describe the implementation of the straightforward methods: the constructors and the methods to load and save the model, since their implementation is trivial. The main difference between the substochastic HMM and the regular HMM is that on the E-Step of the training method, we need to project the posteriors to a space where the sum of the posteriors for each source word is smaller than or equal to one. The mathematical derivation of the projection step is not central to the explanation of how to extend the toolkit so we include it as Appendix A. Suffice it to say that we can implement the projection step as a few iterations of gradient descent.

We created a copy of the `public EStepStats eStep()` method. After the computation of the posteriors (`makePosterior(forward, backward, likelihood)`) we project them onto the constraint set. The projection is implemented as the method

```
processPosteriors(posteriors, s, f, probCache)
```

where `s`, `f` and `probCache` are the source sentence, target sentence, and a cache of the translation probabilities respectively. The `MStep` method and inference methods are the same as in the original `HMM` so we do not need to implement them and we are done.

Suppose that having implemented our new model, we decide that we would like to per-

form a different kind of decoding. In particular, since we could perform the projection by computing some sentence-specific translation probabilities, we might want to try using these translation probabilities at decode time. If we want to do this, we would need to override the `posteriorDecodingAlignment` method of the `HMM` class. The new method will be almost identical to the old, with the exception that we would use the

```
processPosteriors(posteriors, s, f, probCache)
```

method that we implemented for `eStep()` in order to compute the projected posteriors right after the call to `makePosteriors`. To make a similar update to Viterbi decoding we need to override the `viterbiAlignment` method. At the start of the method, we would add code similar to the start of `posteriorDecodingAlignment` to compute posteriors and project them to the constraint set. We would then replace the call to `_tb.getProbability`, which currently uses the translation table to look up the translation probability and we would instead look up the translation probability in our `probCache` array, which has been updated by the call to `processPosteriors`.

5. Related Work

In a standard SMT pipeline PostCAT is a plug-in replacement for the GIZA++ toolkit and serves the same purpose. In fact if the goal is to produce word alignments using the baseline models we cannot recommend PostCAT over GIZA++ since the GIZA++ implementations are faster, and GIZA++ is integrated into most MT systems scripts (e.g. Moses [4], Syntax Augmented Machine Translation (SAMT) (Zollmann and Venugopal, 2006) [5]). Having said that, if the goal is to analyze the alignment results to understand how they impact translation performance, PostCAT provides useful visualization and analysis tools. Additionally, it provides a set of alignment algorithms, not implemented elsewhere and proven to work well for SMT. Furthermore, the code is easy to read and modify so if a researcher wants to extend the models, try different decoding or symmetrization schemes, or fine-tune some aspect of alignment this should be easy to do in PostCAT. For instance, two recent trends are the use of n-best alignments, and the use of alignment posterior probabilities when extracting phrases. Both of these are easily done in PostCAT since the posteriors are available within the `Alignment` object. Another open source word aligner, the Berkeley Aligner, available online[6] contains the implementation from (Liang, Taskar, and Klein, 2006). Their contribution is a model that has a similar intuition as our agreement constraints, but with very different realization. They define an intractable joint model and use an approximation to optimize model parameters.

[4] `http://www.statmt.org/moses/`

[5] `http://www.cs.cmu.edu/~zollmann/samt/readme.html`

[6] `http://www.cs.berkeley.edu/~pliang/software/`

6. Conclusions and Future Work

We have presented the Posterior Constrained Alignment Toolkit (PostCAT), an open-source toolkit for word alignment. Ongoing work on the toolkit is in three directions. Firstly the toolkit is being extended to work over the map reduce paradigm for parallelization. Secondly, new alignment models are being developed in the toolkit's framework and will be available in future versions. Thirdly, we are integrating new methods for phrase extraction, and rule extraction from a word aligned corpora, using the full information available in the word alignments.

Acknowledgments

J. V. Graça was supported by a fellowship from Fundação para a Ciência e Tecnologia (SFRH/ BD/ 27528/ 2006). K. Ganchev was supported by ARO MURI SUBTLE W911NF-07-1-0216.

Bibliography

Brown, P. F., S. A. Della Pietra, V. J. Della Pietra, M. J. Goldsmith, J. Hajic, R. L. Mercer, and S. Mohanty. 1993. But dictionaries are data too. In *Proc. HLT*.

Brown, Peter F., Stephen Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1994. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the em algorithm. *Royal Statistical Society, Ser. B*, 39(1):1–38.

Fraser, Alexander and Daniel Marcu. 2007. Measuring word alignment quality for statistical machine translation. *Comput. Linguist.*, 33(3):293–303.

Ganchev, Kuzman, João V. Graça, and Ben Taskar. 2008. Better alignments = better translations? In *Proc. of ACL-08: HLT*, Columbus, Ohio. Association for Computational Linguistics.

Graca, Joao, Joana Paulo Pardal, Luisa Coheur, and Diamantino Caseiro. 2008. Building a golden collection of parallel multi-language word alignment. In *LREC'08*.

Graça, Joao, Kuzman Ganchev, and Ben Taskar. 2008. Expectation maximization and posterior constraints. In *Proc. NIPS*.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *ACL*. The Association for Computer Linguistics.

Liang, Percy, Ben Taskar, and Dan Klein. 2006. Alignment by agreement. In *Proc. HLT-NAACL*.

Och, Franz Josef and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Comput. Linguist.*, 29(1):19–51.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proc. ACL*.

Vogel, Stephan, Hermann Ney, and Christoph Tillmann. 1996. Hmm-based word alignment in statistical translation. In *Proc. COLING*.

Zollmann, Andreas and Ashish Venugopal. 2006. Syntax augmented machine translation via chart parsing. In *WSMT*. Association for Computational Linguistics, June.

A. Derivation of Substochastic Projection

For notational convenience we encode the desired constraints in terms of some feature functions $f_i(\mathbf{x}, \mathbf{a})$ for each source word i, where $\mathbf{x}$ are the two sentences and $\mathbf{a}$ is an alignment. Each feature is associated with a source word i. The feature is a function from an alignment $\mathbf{a}$ to a real number, counting how many foreign words are aligned to source word i in that particular alignment. We group all features f_i into a vector $\mathbf{f}$. Note that there are only linearly many such feature functions, even though we have exponentially many possible alignments $\mathbf{a}$. The projection step is the solution to the optimization problem:

$$\arg\min_{q} \mathrm{KL}(q(\mathbf{a}) \,||\, p_\theta(\mathbf{a}|\mathbf{x})) \;\; \text{s.t.} \;\; \mathbf{E}_q[\mathbf{f}(\mathbf{x}, \mathbf{a})] - 1 \leq 0. \tag{2}$$

where KL denotes Kullback-Leibler divergence, p_θ is the current model (with parameters θ) and $\mathbf{E}_q$ denotes expectation with respect to the probability distribution q. For more details of why we might want to do this we refer the reader to Graça et al. (2008). The optimization problem in Equation 2 can be efficiently solved in its dual formulation:

$$\arg\max_{\lambda \leq 0} \lambda^\top \mathbf{1} - \log \sum_{\mathbf{a}} p_\theta(\mathbf{a} \mid \mathbf{x}) \exp\{\lambda^\top \mathbf{f}(\mathbf{x}, \mathbf{a})\} \tag{3}$$

where we have solved for the primal variables q as:

$$q_\lambda(\mathbf{a}) = p_\theta(\mathbf{a} \mid \mathbf{x}) \exp\{\lambda^\top \mathbf{f}(\mathbf{x}, \mathbf{a})\}/Z, \tag{4}$$

with Z a normalization constant that ensures q sums to one. We have only one dual variable λ_i per constraint, and we optimize them by taking a few projected gradient steps. The partial derivative of the objective in Equation 3 with respect to parameter λ_i is simply $1 - \mathbf{E}_{q_\lambda}[\mathbf{f}_i(\mathbf{x}, \mathbf{a})]$. So we have reduced the problem to computing expectations of our features under the model q. For our particular features this reduces to computing expectations under the normal HMM model. To see this, we have by the definition of q_λ and p_θ,

$$\begin{aligned} q_\lambda(\mathbf{a}) = \;& \overrightarrow{p}(\mathbf{a} \mid \mathbf{x}) \exp\{\lambda^\top \mathbf{f}(\mathbf{x}, \mathbf{a})\}/Z \\ = \;& \prod_j \overrightarrow{p}_d(a_j | a_j - a_{j-1}) \overrightarrow{p}_t(\mathbf{t}_j | \mathbf{s}_{a_j}) \exp\{\lambda^\top \mathbf{f}(\mathbf{x}, \mathbf{a})\}/Z \\ = \;& \prod_{j, i = a_j} \overrightarrow{p}_d(i | i - a_{j-1}) \overrightarrow{p}'_t(\mathbf{t}_j | \mathbf{s}_i) \end{aligned}$$

Where we have let $\overrightarrow{p}'_t(\mathbf{t}_j|\mathbf{s}_i) = \overrightarrow{p}_t(\mathbf{t}_j|\mathbf{s}_i)e^{\lambda_i}$, and retained the same form for the model. So the projection step just creates some sentence-specific translation probabilites for each word pair. The inference procedures are other unchanged but use these updated translation probabilites. We enforce the constraint that $\lambda \leq 0$ by projecting to the negative orthant after each gradient step.