SlowAlign Report

Vilém Zouhar
vilem.zouhar@gmail.com

March 2021

Word alignment is a well-established task, which found its use mostly in PBMT. This report presents SlowAlign, a system combining multiple hard alignment extracting strategies, which are determined by a small number of parameters. The main functionalities of SlowAlign are (1) heuristic parameter estimation in a supervised fashion using gridsearch, (2) combination of multiple soft alignments and (3) data-less alignment based on diagonal alignment, Levenstein distance and blurring.

This report is split into the following sections:

- System Description: introduction to different components of SlowAlign together with the heuristics
- Evaluation: dataset overview and evaluation, especially in comparison to `fast_align`
- Summary: concludes this report and lists future work
- Appendix: technical details, including building this project

Word Alignment

Traditionally, word alignment consists of two parts: (1) soft alignment, which produces alignment scores and (2) induction of hard alignment, which produces the discrete alignment itself from the scores. There are numerous strategies for the second step, *argmax* and *threshold* being the most intuitive and common ones. These extractors can be parametrized by a single number (e.g. cut-off threshold). Usually, it is up to the user to experiment with different values and choose the best one. This becomes increasingly difficult as multiple extractors can be combined together (e.g. intersection) to produce better performance. Then, instead of having just one number to manipulate, it is now a whole vector.

SlowAlign aims to automate this process by performing gridsearch over a subspace of possible values given a train/dev dataset with gold alignment annotations. To that goal, it defines several other extractors and combines them together.

Repository

The attached repository (also hosted at github.com/zouharvi/SlowAlign) is structured as follows:

- `src`: source code in Rust
- `meta`: source for this report, miscellaneous scripts for evaluation and plotting
- `data`: directory intended for data storage, contains two scripts to pre-process data cited in this report

System Description

The system is currently composed of two parts: SlowAlign-Dic and SlowAlign.

SlowAlign Dic

A viable way to align one sentence pair `(test_s0, test_t0)` given that we already have large parallel corpus `{(train_s0, train_t0), (train_s1, train_t1), ..)}` is to simply add the test sentence pair to the training data and run the unsupervised alignment algorithm again:

```
test_s0  ||| test_t0
train_s0 ||| train_t0
train_s1 ||| train_t1
...
```

This, however, does not scale well, especially if the user is interested in online usage of the alignment (data come in sequentially and not in batches). To alleviate this, it is possible to simply store the word translation probabilities (output of the Expectation step in IBM1) and load them at inference time. The results will not be similar as in the first case (the presence of the test sentence can influence the outcome of the training algorithm), but the changes may be considered negligible, and in general, this makes it possible to deploy and expect reasonable runtime. The tool `fast_align` also has this feature, though undocumented.

Another advantage is that one can re-use pretrained word translation probabilities of other systems. Namely, OPUS provides a great variety of such pre-trained word translation dictionaries (*dic*). The second column is the translation probability, the third is the source token, and the last column is the target token. The first column is the number of occurrences together. An example from Ubuntu v14.10 (de -> en), further columns omitted:

```
...
2       0.117647058823529       Abmeldeknopf    logout
5       0.192307692307692       Abmelden        Log Out
7       0.181818181818182       Abmelden        logout
3       0.113207547169811       Abmelden        Log out
5       0.0980392156862745      Abmelden        logging
2       0.0727272727272727      Abmelden        Logout
2       0.114285714285714       Abmeldeoption   logout
4       0.363636363636364       Abmessungen     Dimensions
5       0.27027027027027        Abmessungen     dimensions
...
```

Storing the whole translation matrix would lead to |V|x|V| number of entries, which is undesirable, especially for word pairs with translation probability close to 0. We, therefore, need to decide a threshold by which we determine if a given pair of words is to be stored or not.

SlowAlign-dic simply takes in two files of sentences to be aligned, the mentioned threshold and outputs the word translation dictionary (the first column contains a dummy value, as it is not used). For word translation probability estimation, the IBM1 model without NULL tokens is used. See Appendix for further usage information of SlowAlign-dic (binary `slow_align_dic`).

SlowAlign Main

The output of this component is always an alignment (0-indexed) of the input (either files or sentences passed through the CLI). Additional tasks may be performed, such as searching for optimal parameters or evaluating the performance when gold alignments are supplied. The stdout is always reserved for just the alignment. See Appendix for detailed usage information of SlowAlign (binary `slow_align`).

Extractors

SlowAlign is especially targeted to improve hard alignment extraction. The soft alignment can be the intermediate representation in IBM models, but it can also be other metrics: attention energies, the difference in word position in the sentence, Levenstein distance, etc.

$\mathbf{A_1}$: In IBM Model 1, the hard alignment induction is done by an argmax from the target side: align every target token with the source token of the mutual higher alignment score. This makes a strong assumption that every target token is aligned to exactly one source token. Because of this, the IBM Model 1 also uses NULL tokens so that a target word has the possibility to align itself to NULL.

$\mathbf{A_2^\alpha}$: The even simpler approach is to consider every alignment with the score above some threshold. This makes this extractor parametrized by one value α. In essence, the expressive power of this is higher than that of A_1 because it does not impose any restrictions on the number of alignments. This is at the cost of not considering the context of possible values for every target token.

$\mathbf{A_3^\alpha}$: The threshold can also be set dynamically. For every source token s, compute the threshold as $\alpha \times max_t\{\text{score}(s,t)\}$ and then take all alignments with score at least this value. The alpha values are bounded between 0 - take everything and 1 - take only the argmax (+every alignment with a score equal to the maximum). Note: this can be further generalized to accommodate also negative scores by dividing instead of multiplying.

$\mathbf{A_4^\alpha}$: The last extractor is equivalent to $\mathbf{A_3^\alpha}$ with the only difference of aggregating from the target side.

Combination: Further improvements can be made by the combination of the two via set operations. Usually, the union improves the recall, while the precision is improved by the intersection. For example, the semantics of $A_1 \cap A_2^\alpha$ would be: *align every target token to its maximum source counterpart, but with the rule of all scores begin above* α. Finally, an extra improvement is to consider alignment from both directions. Note that, for example, the output of IBM Model 1 is not just the transpose of the output on switched parallel data because the E-M algorithm does asymmetric operations.

The main formula used in SlowAlign is parametrized by 7 real values:

$$\mathbf{A_5^\alpha} = \Big[\big[A_4^{\alpha_1}(\text{IBM}_{\text{fwd}}) \cap A_3^{\alpha_2}(\text{IBM}_{\text{rev}}) \cap A_2^{\alpha_3}(\text{diag}) \cap A_2^{\alpha_4}(\text{blur}^{\alpha_5}(\text{IBM}_{\text{fwd}}))\big] \cup A_2^{\alpha_6}(\text{levenstein})\Big] \cap A_4^{\alpha_7}(\text{IBM}_{\text{fwd}})$$

The soft alignments are the following:

- IBM_{fwd}: IBM Model 1 without the NULL tokens.
- IBM_{rev}: IBM Model 1 without the NULL tokens performed on switched data (target-source) and then transposed.
- diag: Absolute value of relative positions in the sentence: $\left|\frac{i}{|S|} - \frac{j}{|T|}\right|$
- levenstein: Levenstein distance of two words. This is useful for non-text tokens, such as interpunction, but can also be used for, e.g. the alignment of post-edited dialect to the standard language.
- blur: Applies blurring filter $[0, \alpha, 0], [\alpha, 1 - 4 * \alpha, \alpha], [0, \alpha, 0]$ on inside nodes of the soft alignment. This is motivated by the fact that if adjacent words have high scores to be aligned to the same target word, then the source word in the middle is also probably aligned to the same target word.

The parameters are specified in this format (square brackets mandatory): $[\alpha_1], [\alpha_2], [\alpha_3], [\alpha_4, \alpha_5], [\alpha_6], [\alpha_7]$. Default is `[0.0],[1.0],[0.8],[0.0,0.1],[0.95],[0.8]`.

The gridsearch method searches the parameters in the following space. The behaviour of `linspace` is similar to the one of NumPy (endpoint included). This space is defined in `src/optimizer.rs`.

```
linspace(0.95, 1.0, 4),
linspace(0.90, 1.0, 6),
linspace(0.1, 1.0, 10),
cartesian_product([linspace(0.1, 0.3, 8), linspace(0.0, 0.005, 4)]),
linspace(0.7, 1.0, 4),
linspace(0.0, 0.005, 4),
```

Methods

The following table lists methods available under the argument `--method` together with a minimum description. The methods were chosen to fill a specific requirement niche. Their top-level behaviour is defined in `src/main.rs`. Parameters have defaults that can be changed using the `--params` argument. Currently, there is no mechanism nor typing system to enforce the shape of the parameters. The structure is, however, invariant for every method, and it can be observed from the defaults. Extra parameters will be ignored; not enough parameters will cause a panic.

Name	Comment	Extraction	Purpose
`ibm1`	Standard IBM1 model (without NULL tokens). Default number of steps is 5.	argmax ($\mathbf{A_1}$)	Baseline comparison to other methods.
`levenstein`	Alignment score of two words is based on their Levenstein distance: $1.0 - \frac{\text{lev}(s,t)}{\|s\|+\|t\|}$	threshold ($\mathbf{A_2}$) default `[0.75]`	Lexical based approximation of alignment.
`static`	Combination of diagonal alignment and Levenstein. Soft alignment is the arithmetic average of Levenstein distance and $\left\|\frac{i}{\|S\|} - \frac{j}{\|T\|}\right\|$. Default method of `slow_align`.	threshold ($\mathbf{A_2}$) default `[0.4]`	Alignment between two dialects for which little data is available.[1] This does not require any training.
`search`	Performs gridsearch over a hardcoded subset of the possible values of α. The (first) set of parameters with the lowest AER is then outputted. Parameters `--dev-count` controls how many sentences (from the top) are used for parametric estimation. For the final evaluation, `--test-offset` determines from which sentence (until the end of supplied aligned sentences) to compute the final AER.	Searched space above. Defined in `src/optimizer.rs`.	Using a small number of supervised examples to achieve better performance.
`a5_fixed`	The same as `search`, only the `params` have to be provided.	$\mathbf{A_5}$ (above) default above	Transfer testing.
`dic`[2]	A combination of multiple soft alignments. Requires OPUS-like translation probability table passed by `--dic`. This table can be either downloaded from OPUS or trained using `slow_align_dic`.	$\mathbf{A_5}$ (above) default above	Fast inference given the parameters of `search`.

Speed

The following table lists runtimes measured once on Ryzen 7 3800x. It is debatable whether the parameter search in `search` is part of the training or whether it can be viewed as simply hyperparameters and `a5_fixed` should be measured. Despite its name, SlowAlign is currently slower by a tolerable margin, with bottlenecks (unparalleled Levenstein) clearly visible and improvable in future work.

Method	Czech	German	French
levenstein	1.2s	<0.1s	18.7s
static	1.2s	<0.1s	19.5s
ibm1	1.2s	<0.1s	8.9s
search	8 hours	6 min	6.5 min
dic	3.3s	0.2s	1 min
fast_align	1s	0.3s	8.6s

[1] Added per request by a classmate, but still serves a great purpose as a training-less baseline.
[2] A similar feature is offered by fast_align, though it is hidden in the code and is undocumented.

Evaluation

Datasets

There are three datasets used:

- English-Czech [1] with 5003 aligned sentences. The last 2503 is used for test evaluation. (shuffled, see `data/process_encs.sh`) The alignment is indexed by one, therefore use `--gold-index-one`.
- German-English [2] with 100 aligned sentences. The last 50 is used for test evaluation. (shuffled, see `data/process_deen.sh`)
- French-English [3] with 37 aligned sentences. Used solely for test evaluation.

Since English is one side of each of them, this report refers to the datasets as Czech, German and French. In all cases, the whole data is visible to the system, but only the first {2500,50,0} alignments.

Furthermore, we use translation dictionaries made available by OPUS [5], namely Europarl v8, TildeMODEL v2018, DGT v2019 and EUBookshop v2.

Performance

For measuring word alignment performance, Alignment Error Rate is used. Given hypothesis alignment A, sure gold alignments S and possible gold alignments P, then $AER = \frac{|A \cap S| + |A \cap P|}{|A| + |S|}$. The following table lists AER (percentage) of three Slow Align baselines (levenstein, static, ibm1), best Slow Align (search + OPUS dic) all run with `--lowercase` and fast_align [4] (default parameters `-dov`). Configurations can be run by manipulating `meta/evaluate{,dic}.sh`.

Method	Czech	German	French
levenstein	63.4	62.4	50.7
static	51.4	46.9	43.1
ibm1	48.9T	62.2	31.1
search	41.6 (41.5)	48.5 (48.0)	(16.8)
dic	**36.0**	**24.3**	(19.2)
fast_align	38.9	46.3	18.5

Values in brackets mark AER on training data. Parameters for `dic` were chosen manually by performance on the training set (this could be done automatically, though the intended usage for `dic` is only fast inference). Even though this comparison looks promisingly good to Slow Align, it is unfair since Slow Align can make use of gold alignments and also translation probabilities from much larger corpora, while fast_align is limited to only unsupervised parallel data in the given corpus.

These were the following parameters for `search` and `dic`:

```
search:
Czech:   [0.95], [0.90], [0.7], [0.21, 0.0000], [0.70], [0.0050]
German:  [1.00], [1.00], [0.5], [0.21, 0.0000], [0.70], [0.0033]
French:  [1.00], [0.96], [0.7], [0.17, 0.0050], [0.80], [0.0033]

dic:
Czech:   [0.7], [0.20], [0.7], [0.10, 0.0001], [0.80], [0.0]
German:  [0.5], [0.20], [0.7], [0.10, 0.0001], [0.98], [0.0]
French:  [0.1], [0.29], [0.7], [0.08, 0.0001], [0.76], [0.0]
```

The `dic` configuration is heavily dependent (up to `+10` AER) on the data used. The performance is affected by the dictionary size and mostly by the domain. This is demonstrated by Europarl dictionary vastly outperforming TildeMODEL for Czech (0.6M vs 3.1k sentences). For German, the smaller in-domain

dictionary of Europarl outperformed TildeMODEL (2M vs. 4.3M sentences). DGT and EUBookshop had lower performance in comparison to Europarl possibly due to either smaller size or unfitting domains.

Parameter Transfer

The previous results showed that different parameters are suited for each of the datasets. The following table simply runs the inference with parameters estimated from another language pair.

Transfer	AER
German→Czech	42.6
French→Czech	42.0
Czech→German	47.0
French→German	48.4
Czech→French	24.7
German→French	25.1

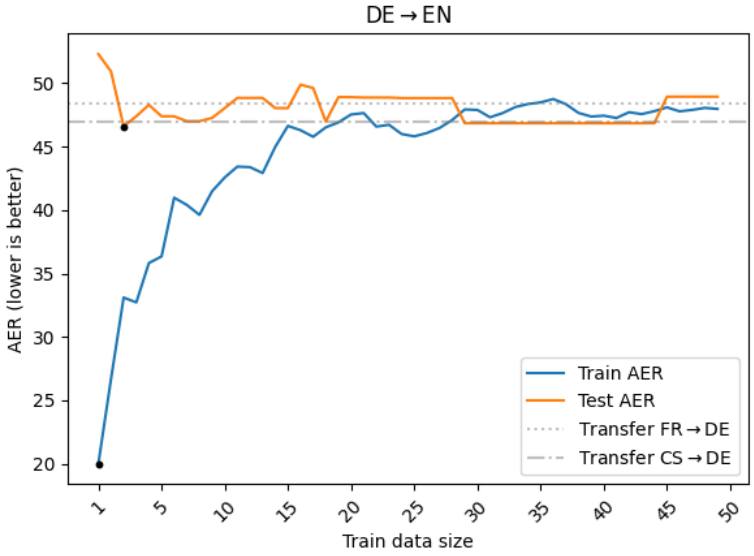
We can observe an expected drop in performance, though not unreasonably large (excluding German). The parameters, in this case, are not only language-specific but also dataset-specific. Annotation guidelines for word alignment are not unified and may lead to a different number of aligned tokens even if applied to the same parallel corpus.

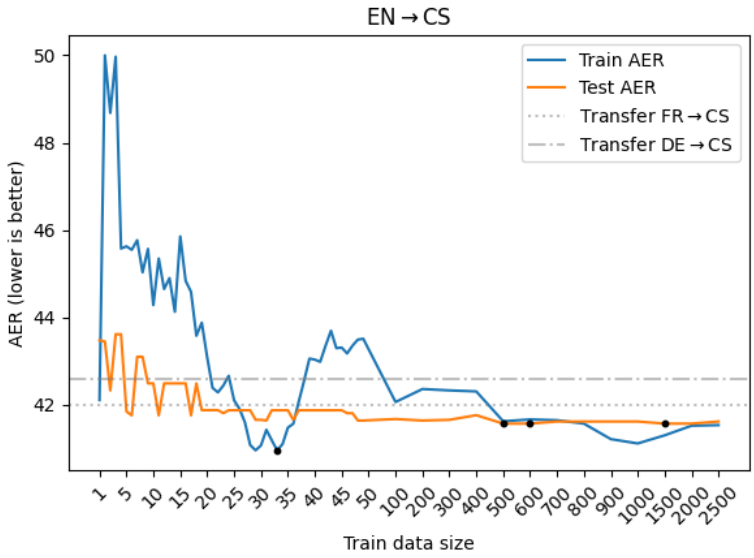
For German, the transfer had better results than training on the language data. One possible explanation is higher variance in the German corpus itself (together with its small size).

Train Data Size

The last two sections suggest that despite the very limited search space, the `search` method is dependent on training data properties. Due to available data limitations, we examine only German and Czech. The latter should have a higher weight since it contains 50x more data. In this specific setup, top-N training sentences are considered.

The following two figures show the results. Only specific values are tested for Czech (given the computational cost of exhaustive search). Horizontal lines depict the performance by transfer. Black dots mark the lowest points separately for test and train AER. Note the non-linear scaling of the x-axis for Czech.





An immediately observable result is the high variability in the training performance. This may be the result of the dataset being small, and for a full study, cross-validation should be used to determine the performance with better confidence (and also provide statistical significances). The expected behaviour would be (1) decreasing test AER (more training data provides a better estimate for the test distribution) and (2) increasing train AER (model "capacity" is being used up to fit more and more sentences). A trend for (1) can be observed in Czech (especially for higher values) and (2) in German. A possible explanation is that the Czech dataset is more homogenous and more consistent in the alignments than German. Because of this, "new'' sentences to the training dataset will have a similar optimal solution and, therefore, can even decrease the training data size.

A conclusion from this evaluation is, that especially given the computational cost of gridsearch with larger training data, it is viable to use only small subset of them to get reasonable results.

Summary

This report introduced a new tool for word alignment, Slow Align. Despite being (tolerably) slower and having worse vanilla performance, it is still a versatile tool with the advantage of (1) clearly manipulatable parameters that affect the output alignment and (2) being able to use existing translation tables, which yield vastly better performance. Compared to fast_align, it provides competitive results and allows for higher degree of explainability and extensibility.

Future Work

Built-in HTTP server: Word alignment can be used in multiple scenarios. It can be useful for research purposes as a binary installed on a computer, but it is also necessary in some deployment scenarios. Furthermore, sharing an alignment service in collaboration is easier than to manually send around parallel data to align or to learn to use a specific word alignment tool. The simplest solution would just accept the request and call the appropriate method from the main binary. This would be resource wasteful since, e.g. the word translation dictionary would have to be loaded from the filesystem (or cached in the memory by the OS) and parsed for every request (usually a pair of sentences). A solution to this would be to specify a list of dictionaries to load when starting the server (more secure but also more restrictive) or keep an explicit cache of one most recently used dictionary.

Gridsearch multithreading: Currently, only the IBM soft alignment implementation is multithreaded (fixed to 4 threads), even though it is problematic and bottlenecked between the Expectation and Maximization steps. Inducing hard alignment using the recipes is, however, a pure function which only needs read access to the soft alignment package. In the end, only the argmax needs to be extracted (parameters) together with the AER. Even though this does not influence all the use-cases and especially the inference scenario, it is possible to gain a multiplicative speedup determined solely by the number of cores (e.g. 8x). This is also true for Levenstein computation.

Custom alignment score input: It has been shown that attention scores can be used for word alignment, especially if one of the attention heads is trained for this explicitly. Further improvements can be made if these scores are joined with other soft alignments. The main advantage is also the more complex hard alignment induction scheme. An MT practitioner who wishes to, e.g. also present the word alignments next to the MT output, may choose to send a request to SlowAlign together with their attention scores (and possibly other parameters) to get a better hard alignment.

References

- [1] Mareček, D. (2011). Automatic Alignment of Tectogrammatical Trees from Czech-English Parallel Corpus.
- [2] Biçici, M. E. (2011). The regression model of machine translation (Doctoral dissertation, Koç University).
- [3] Original: Germann, U. (2001). Aligned hansards of the 36th parliament of canada.
  Processed: https://github.com/xutaima/jhu-mt-hw/tree/master/hw2/data
- [4] Chris Dyer, Victor Chahuneau, and Noah A. Smith. (2013). A Simple, Fast, and Effective Reparameterization of IBM Model 2. In Proc. of NAACL.
- [5] Jörg Tiedemann, 2012, Parallel Data, Tools and Interfaces in OPUS. In Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC'2012)
  Namely: Europarl v8, TildeMODEL v2018, DGT v2019 and EUBookshop v2

Appendix

SlowAlign is written in Rust (1.50 in this report). It is somewhat thinner (1378 LOC) compared to fast_align (1726 LOC).

For installation, please refer to the official resources. Assuming that `cargo` is in the path, the main binary can be run as: `cargo run --release --bin slow_align -- <arguments>`. The target binary is going to be stored in `target/release/slow_align` (and can be run as `./target/release/slow_align <arguments>`). To just build the binary and not run it, replace `run` with `build`. The rest of this appendix simply lists abbreviated help pages of `slow_align` and `slow_align_dic` binaries.

Output of `slow_align -h`:

```
USAGE:
    slow_align [FLAGS] [OPTIONS]

FLAGS:
        --gold-index-one    Treat gold alignments as if they are 1-indexed
                            (default is 0-indexed)
    -h, --help              Prints help information
        --lowercase         Treat everything case-insensitive (default is case-sensitive,
                            even though that provides slightly worse performance).
        --switch-dic        Switch the columns for dictionaries, default (src, tgt).
    -V, --version           Prints version information

OPTIONS:
        --dev-count <dev-count>          Number of sentences (from the top) to use for
                                         parameter estimation. [default: 0]
    -d, --dic <dic>                      OPUS-like dictionary of word translation
                                         probabilities
    -f, --file1 <file1>                  Path to the source file to align. (If both files
                                         and sentences are provided, only files are used).
    -f, --file2 <file2>                  Path to the target file to align. (If both files
                                         and sentences are provided, only files are used).
    -g, --gold <gold>                    Path to the file with alignments (single space
                                         separated, x-y for sure alignments, x?y for
                                         possible). `x` and `y` are (by default) 0-indexed
                                         token indicies.
        --ibm-steps <ibm-steps>          Number of steps to use for IBM1 computation.
                                         [default: 5]
    -m, --method <method>                Which alignment method pipeline to use (static,
                                         dic, levenstein, ibm1, search, a5_fixed)
                                         [default: static]
    -p, --params <params>                Comma-separated arrays of parameters to the
                                         estimator recipe
    -s, --sent1 <sent1>                  List of source sentences (separated by \n)
                                         to align.
    -s, --sent2 <sent2>                  List of target sentences (separated by \n)
                                         to align.
        --test-offset <test-offset>      Offset from which to evaluate data. If not
                                         supplied, use --dev-count value (so that dev and
                                         test do not overlap).
```

Output of `slow_align_dic -h`:

```
USAGE:
    slow_align_dic [FLAGS] [OPTIONS] <file1> <file2> <out>

ARGS:
    <file1>    Path to the source file to train word translation probabilities on.
    <file2>    Path to the target file to train word translation probabilities on.
    <out>      Path to the output word translation probabilities dictionary.

FLAGS:
    -h, --help         Prints help information
        --lowercase    Treat everything case-insensitive (default is case-sensitive,
                       even though that provides slightly worse performance).
    -V, --version      Prints version information

OPTIONS:
    -t, --threshold <threshold>    Threshold under which translation probabilities
                                   will be omitted. Lower values lead to better
                                   approximation, but also larger file size.
                                   [default: 0.2]
```