NLP 2 - Project 2

April 18, 2016

In this project you will implement a phrase-based translation system. You will experiment with monotone translation and lattice translation. Importantly, you will learn about formal tools such as finite-state transducers and the underlying complexity of the translation problem.

In summary, you will experiment with

- monotone translation
- decision rules
- preordering and lattice translation

You will not implement finite-state transducers (and their operations) yourselves, instead you are expected to use `OpenFST` (Allauzen et al., 2007).[1] We advise you to use the command line tools provided with the library for they are **much simpler to use** than the `C++` API. The illustrations in this document were all produced by `OpenFST` and that is what you should use in your report. There is a tutorial on design and operations[2] and another on applications[3]. For a very comprehensive survey on weighted automata algorithms refer to (Mohri, 2009).

We are providing data for a development set of English-Japanese sentence pairs.[4] The data amounts to 1416 sentence pairs, and even though you may experiment with as many sentences as you like, you are expected to submit results only for the first 100 sentence pairs. The data folder contains README files with additional information, but in sum you will find:

- Source, reference and permutations
- Phrase tables

[1] http://www.openfst.org
[2] http://www.openfst.org/twiki/bin/view/FST/FstHltTutorial
[3] http://www.openfst.org/twiki/bin/view/FST/FstSltTutorial
[4] https://wilkeraziz.github.io/uva-nlp2/resources/project2/data/pbsmt.tgz

- Examples of packed permutations
- Examples of translations
- Model parameters

1 Phrase-based MT

In phrase-based MT (Koehn et al., 2003), we do not have a proper generative model from where we can infer latent derivations and thus generate translations of an input sentence. Instead, we create a large repository of rules (phrase pairs) from some large word-aligned training data. Rules consist of phrase pairs extracted by enumeration of connected subsets of word alignments – check (Koehn et al., 2003) for a more precise definition. Then, for a given input, we instantiate the space of possible translations by concatenating known phrase pairs in target word order such that they cover some permutation of the input sentence.[5] To keep the solution tractable, we impose some form of limit to reordering (typically by means of a distortion limit). Once the space of translations is defined, we score each and every hypothesis using a linear model. Because the features of this linear model are rather local (e.g. at the level of phrases or short target n-grams), this scoring can be done quite efficiently in a packed representation of the complete set of translation derivations (that is, there is no need for enumeration as in a list). In this project, you will assume the parameters of the linear model are known, and you will explicitly instantiate the weighted set of translation derivations of a given input. You will also experiment with different decision rules, i.e., ways in which to select one translation as output.[6]

2 Tasks

You will implement phrase-based MT using finite-state transducers. Some of the constructs are similar to those presented by Knight and Al-Onaizan (1998) for word-based models based on $\text{IBM}_{\geq 3}$ and by Kumar and Byrne (2003) for the alignment template model. For example, check Figure 1 in (Kumar and Byrne, 2003) for a general architecture. Keep in mind that phrase-based MT does not offer explicit models for segmentation and permutation. Segmentation is basically guided by the known source phrases and permutation is arbitrary within a distortion limit. A few features are designed in an attempt to capture segmentation and permutation preferences, such as phrase-penalty and distortion cost.

[5] This is reminiscent of data-oriented parsing (Bod, 1992).

[6] In project 3, you will fit the parameters of the linear model to data using discriminative learning algorithms.

2.1 Task 1: source sentence

A source sentence can be seen as a trivial linear finite-state acceptor whose arcs are labelled with the words in original order. Consider the example `the black dog`, Figure 1 illustrates one way to encode it as a transducer.[7]

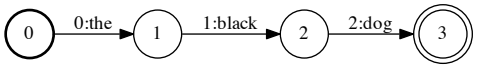

Figure 1: Input seen as a transducer: the label encodes an input position and its corresponding word.

Table 1 summarises the task.

TASK	encode source sentences as transducers
INPUT	first 100 English sentences in `dev.en`
OUTPUT	one transducers for each sentence
SUBMIT	nothing to submit here
REPORT	add a graphical illustration for one example from `dev.en`

Table 1: Task 1 summarised

2.2 Task 2: phrase table

A phrase-table can be seen as recursive finite-state transducer that maps contiguous sequences of source words onto contiguous sequences of target words. Consider the rules in Figure 2, the resulting transducer is shown in Figure 3. Note that the phrase table transducer defines an infinite set of binary relations over $\Sigma^* \times \Delta^*$ (where Σ is the source language vocabulary and Δ is the target language vocabulary).

There are different possible designs and there is not one that is per se a standard. The one I chose is similar to the one presented in (Kumar and Byrne, 2003, Figure 6) and (Knight and May, 2009, Figure 1). In my design, there is one initial state which is also final, every translation rule starts and ends in this state. If a translation rule maps a single source word to a single target word, then this rule can be compactly represented by one arc (see `black:noir` in Figure 3). Otherwise, when a rule maps m source words to n target words (where $m \neq n$), I decided to first recognise all source words, without producing any

[7] Obviously, there are different ways to do so. I chose to encode it as a transducer from 0-based input positions to source words because the positions will turn out to be useful when we are interested in retrieving the alignment between source and target after translation.

target words, and only then to generate target words one by one making sure the last arc of the rule returns to the initial/final state. Observe that the internal states associated with a certain rule (for example 1, 2 and 3 in Figure 3) are there to memorise how much of the rule we have already matched. Even though there are more compact ways to represent translation rules, this seems rather clean and easy to turn into an algorithm.

Source	**Target**
`the`	`le`
`the`	`un`
`dog`	`chien`
`black`	`noir`
`black`	`noirs`
`black dog`	`chien noir`

Figure 2: Example phrase table.

In this task you will have to produce a weighted finite-state transducer for each of the first 100 sentences in `dev.en`. Under `rules.monotone.dev` you will find one phrase table for each source sentence, they are stored in files named `grammar.`n where n is the sequential (0-based) position of the sentence in `dev.en`.

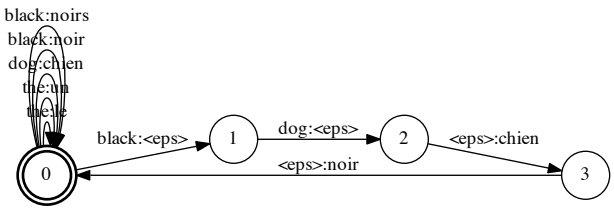

Figure 3: Phrase table as a transducer.

In a grammar file, each rule is represented by 5 fields separated by 3 vertical bars:

- the first field can be ignored
- the second field contains the source phrase
- the third field contains the target phrase
- the fourth field contains a feature map:

  - `IsSingletonF`: whether the source phrase occurred only once in the training data
  - `IsSingletonFE`: whether the phrase pair occurred only once in the training data
  - `SampleCountF`: frequency of the source phrase in the training data
  - `CountEF`: frequency of the phrase pair in the training data
  - `EgivenFCoherent`: frequency of the target phrase normalised w.r.t. frequency of source phrase
  - `MaxLexEgivenF`: lexical smoothing
- the last field (which you can ignore) represents the internal alignment of the phrase pair

In a phrase-based system, translation rules are parametrised by a linear model. Let (f, e) be a phrase pair and $\phi : \Sigma^+ \times \Delta^+ \rightarrow$ a mapping between phrase pairs and a d-dimensional feature vector. Then each phrase pair (f, e) is associated with a parameter $\theta_{(f,e)} = w^\top \phi(f, e)$ where w is a vector of model parameters.[8] You will note that rules are annotated with features, those are components in the linear model. You will need those, and the model parameters in `weights.monotone`, in order to score phrase pairs. Besides the features in the phrase table, you need to consider 3 other features which also decompose at the phrase level, but aim mostly at scoring segmentations, namely:

- `Glue`: the total number of phrases used in a derivation - each phrase pair contributes to this feature with a count of 1
- `WordPenalty`: the total number of target words in a derivation - each phrase pair contributes to this feature with $\frac{-1}{\ln(10)}$ times the number of tokens in the target phrase

Finally, to make sure phrase tables can cover all words in the input we typically augment them with "passthrough rules" (attention: the phrase tables we provide do not contain such rules, you have to take care of that yourself). A passthrough rule is a trivial rule that copies a symbol from the input stream onto the output stream. If you watch carefully, you will see that we provided a model parameter for the feature `PassThrough`. This feature simply counts the number of passthrough rule applications used in a derivation, in other words, it captures how many source words were unknown. Note that as defined, each "passthrough phrase pair" contributes to this feature with a count of 1. In most cases, this feature is meaningless as the number of unknown source words is constant for each source sentence. In practice, for reasons beyond the scope of this section, translation systems might estimate a parameter for this features nevertheless.

There are at least two ways do implement passthrough rules:

[8] In the transducer, you can weight translation rules by weighting one of its arcs, typically the first or the last.

TASK	encode phrase tables as transducers
INPUT	one phrase table per English sentence: `rules.en-ja.dev.tgz`
OUTPUT	one transducers per phrase table in `rules.en-ja.dev.tgz`
SUBMIT	nothing to submit here
REPORT	discuss the steps involved in creating a phrase table transducer and scoring its arcs, illustrate fragments of phrase tables containing at least one phrase of each of the following types: $1-1$, $m-n$ with $n > m$ and with $m > n$

Table 2: Task 2 summarised

1. You can add to your phrase table a single translation rule `OOV:OOV` whose weight is $1 \times w_{\texttt{PassThrough}}$, where `OOV` is a symbol reserved for out-of-vocabulary (OOV) source words. Then, you will need to pre-process the source sentence so that every OOV word is replaced by `OOV`. For example, if the source sentence is `the black cat`, then `cat` is OOV with respect to the phrase table in Figure 2, we would then pre-process the sentence so that its corresponding transducer recognises/produces `0:the 1:black 2:OOV`.

2. An alternative method is to pre-process the phrase table as to include rules for each OOV word type essentially making them in-vocabulary. In this case, we would add to our phrase table a rule `cat:cat` and its score would be $1 \times w_{\texttt{PassThrough}}$.

Table 2 summarises the task.

2.3 Taks 3: monotone translation

Once we have represented both the input and the translation rules as transducers, finding the set of translation derivations boils down to *composing* the two devices. Finite-state composition between two binary relations yields a third binary relation where the input of the second is constrained to the output of the first (Mohri, 2009). In other words, composing the input sentence and the phrase table (both seen as transducers) yields the set of paths whose input side corresponds exactly to the source sentence.

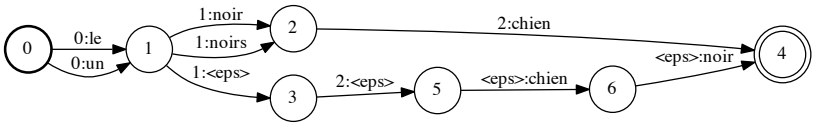

Figure 4: Translation derivations (or translation lattice).

A path in the composed transducer represents a translation of the input. Figure 4 illustrates all possible translations of our running example. Note how the finite-state transducer efficiently packs multiple translation paths. At this point it is clear why I chose to have input positions in the source transducer, the alignment between source and target is now obvious. For example, consider the path `0:le 1:eps 2:eps eps:chien eps:noir` through states (0, 1, 3, 5, 6, 4). The first source position (0) translated directly producing the first target word (`le`); the second and third source positions (1-2) were grouped into a phrase and produced the second and third target words (`chien noir`). A somewhat standard text representation of that is: `le |0-0| chien noir |1-2|`.

Let us start by completely ignoring reordering. For each input sentence in `dev.en`, you will instantiate the space of all monotone translations by composing it with the corresponding weighted phrase table. Do pay special attention to out-of-vocabulary words. If a sentence contains words which are unknown to the phrase table, and you do not care to augment your phrase table with *passthrough rules*, the resulting set of translation derivations will be empty (can you see why?)!

Table 3 summarises the task.

TASK	monotone translation by FST composition
INPUT	an input transducer per English sentence a weighted FST phrase table per English sentence
OUTPUT	a packed weighted set of translation derivations

SUBMIT	`monotone.100best.`n: 100-best derivations from each automaton
	in text format showing the alignment with the input
	Example: 2-best derivations
	`le |0-0| chien noir |1-2|`
	`le |0-0| noir |1-1| chien |2-2|`

REPORT	present FST composition and discuss computational complexity

Table 3: Task 3 summarised

2.4 Task 4: decision rules

Ultimately, we need to select one translation out of the space of all possible translation derivations. The most typical *decision rule* is to compute the model's best derivation, that is, the derivation that has maximum score (or minimum cost) under the model (this rule is also known as VITERBI or *best-derivation*).[9] Given that translation derivations are sensitive to unobservable variables such as segmentation and alignment, multiple derivations may yield the same translation. A perhaps more principled decision rule would sum over the

[9]The linear models in this project were trained with a "cost" semantics so that they are compatible with `OpenFST`.

TASK	produce translation decisions
INPUT	k-best lists from previous task
OUTPUT	a file containing the best translation derivations of each sentence in `dev.en`: `monotone.der` a file containing the best translation string (in the k-best list) of each sentence in `dev.en`: `monotone.trans`
SUBMIT	`monotone.der` and `monotone.trans` use text format and do not report alignments
REPORT	BLEU scores for each decision rule using references in `dev.ja`

Table 4: Task 4 summarised

File	Description
`dev.enpp.nbest`	Up to 100-best permutations of each sentence in `dev.en`
`rules.n-best.dev`	A collection of phrase pairs relevant to permutations of each source sentence
`weights.lattice`	Parameters of a linear model for lattice translation

Table 5: Data for lattice translation.

space of latent assignments. This disambiguation (sometimes referred to as MAP or *best-translation*) is known to be NP-complete (Sima'an, 1996). A tractable alternative is to approximate the complete space of translation derivations by a list containing the n-best scoring derivations and perform the summation in this set.

You should experiment with both decision rules, where in the case of *best-translation* you should use k-best lists produced in the previous task. Table 4 summarises the task.

2.5 Task 5: permutation lattice

Monotone translation is not at all realistic. However, instead of experimenting with phrase-based distortion-based permutation mechanism, you will experiment with preordered input. We are providing you with lists containing up to 100 permutations of the sentences in `dev.en` and with phrase tables that are suitable for those permutations. The data available for this task is summarised in Table 5. You are not directly provided with lattices of preordered input, instead you are provided with lists such as illustrated in Figure 5.

There is a trivial way to represent a list using automata/transducers which are non-deterministic. See Figure 6, where I have represented each permutation using an independent path to an independent final state. I have weighted the path by weighting its final arc and I am using negative log-probabilities.

The downside is that there is no packing and the representation is rather inefficient. Consider finite-state operations that may alleviate that problem, such as determinisation

0.7 `the dog black`
0.2 `dog black the`
0.1 `the black dog`

Figure 5: Example of list of permutations (and their probabilities).

and minimisation (Mohri, 2009).[10] Figure 7 illustrates a much more efficient representation: observe that even though the topology of the automaton changed, the paths are exactly the same and so are their negative log-probabilities.[11]

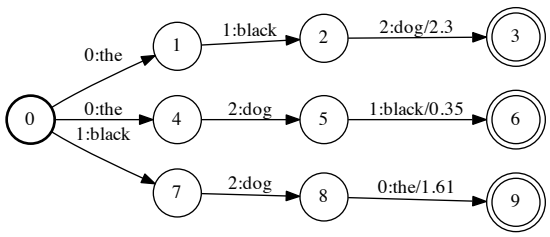

Figure 6: List of permutations as a non-deterministic transducer.

In order to discriminate alternative permutations, we will add a feature to our linear model. This feature is the negative log-probability of the permutation and the model parameter associated with it is called `LatticeCost` (in `weights.lattice`). Table 6 summarises the task.

TASK	pack permutations into weighted lattices
INPUT	up to 100 permutations per source sentence
OUTPUT	a permutation lattice per source sentence
SUBMIT	nothing to submit here
REPORT	discuss the strategies for packing: e.g. determinisation and minimisation

Table 6: Task 5 summarised

[10] Roughly, a deterministic automaton is such that, given an origin state and label, there is only one possible transition. An automaton is said minimal if it has no more states than the minimum necessary to recognise/accept a given language.

[11] You will note that the weights of the paths have been somehow distributed along the arcs towards the initial node. This is the result of a standard operation called *push* (Mohri, 2009). Pushing weights towards the initial node is usually a good idea because one can reason about the total weight of a path earlier on. This is particularly useful when decoding with A* or beam search.

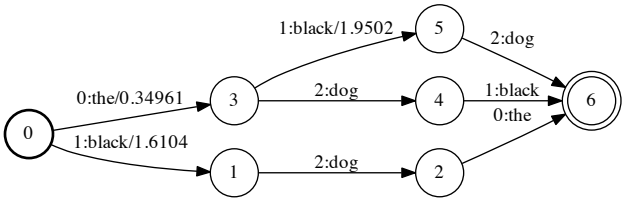

Figure 7: Permutations packed in a deterministic transducer.

2.6 Task 6: lattice translation

A simple (though inefficient) way to translate a source sentence in non-monotone order would be to enumerate all alternative permutations and translate them individually. Then, with some appropriate decision rule, we select the final translation. Besides inefficient, some decision rules can only be defined in the combined space of translations.

Now that the permutations have been efficiently represented using transducers, translating these permutations is no different from translating a trivial linear finite-state transducer. That is right, all it takes is a composition between the input lattice and the phrase-table transducer. Figure 8 illustrates the composition between the permutation lattice in Figure 6 and the phrase table in Figure 3.

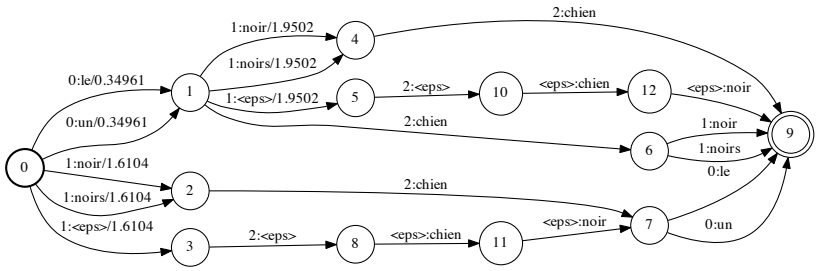

Figure 8: Translation derivations for permutations of the input.

Because we have changed the space of translations of our model (by including translations of permutations of the input), and because we have added a new feature to the linear combination (i.e. `LatticeCost`), it makes sense to re-estimate the parameters of the linear model. You will not need to do it yourself,[12] as I have already provided you with

[12] But wait for project 3 ;)

`weights.lattice`. You will need, however, to recompile your phrase tables so that they are weighted by the correct linear model. Table 7 summarises the task.

TASK	translate permutation lattices
INPUT	one permutation lattice per sentence and a recompiled weighted transducer per phrase table
OUTPUT	a weighted set of translation derivations per sentence 100-best paths from each transducer *best-derivation* and *best-translation* decision rules
SUBMIT	`lattice.100best.`*n*: 100-best derivations from each transducer in text format with alignments `lattice.der` and `lattice.trans`
REPORT	BLEU score for each decision rule using references in `dev.ja`

Table 7: Task 6 summarised

References

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). *Implementation and Application of Automata: 12th International Conference, CIAA 2007, Praque, Czech Republic, July 16-18, 2007, Revised Selected Papers*, chapter OpenFst: A General and Efficient Weighted Finite-State Transducer Library, pages 11–23. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bod, R. (1992). A computational model of language performance: Data oriented parsing. In *Proceedings of the 14th Conference on Computational Linguistics - Volume 3*, COLING '92, pages 855–859, Stroudsburg, PA, USA. Association for Computational Linguistics.

Knight, K. and Al-Onaizan, Y. (1998). Translation with finite-state devices. In *Proceedings of the Association for Machine Translation in the Americas*, AMTA, pages 421–437, Langhorne, PA, USA.

Knight, K. and May, J. (2009). *Handbook of Weighted Automata*, chapter Applications of Weighted Automata in Natural Language Processing, pages 571–596. Springer Berlin Heidelberg, Berlin, Heidelberg.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 48–54, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kumar, S. and Byrne, W. (2003). A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *Proceedings of*

the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1, NAACL '03, pages 63–70, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mohri, M. (2009). Weighted automata algorithms. In Droste, M., Kuich, W., and Vogler, H., editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. An EATCS Series, pages 213–254. Springer Berlin Heidelberg.

Sima'an, K. (1996). Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proceedings of the 16th conference on Computational linguistics - Volume 2*, COLING '96, pages 1175–1180, Stroudsburg, PA, USA. Association for Computational Linguistics.