Latent-Variable Modeling of String Transductions with Finite-State Methods[*]

Markus Dreyer and **Jason R. Smith** and **Jason Eisner**
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218, USA
{markus,jsmith,jason}@cs.jhu.edu

Abstract

String-to-string transduction is a central problem in computational linguistics and natural language processing. It occurs in tasks as diverse as name transliteration, spelling correction, pronunciation modeling and inflectional morphology. We present a conditional loglinear model for string-to-string transduction, which employs overlapping features over latent alignment sequences, and which learns latent classes and latent string pair regions from incomplete training data. We evaluate our approach on morphological tasks and demonstrate that latent variables can dramatically improve results, even when trained on small data sets. On the task of generating morphological forms, we outperform a baseline method reducing the error rate by up to 48%. On a lemmatization task, we reduce the error rates in Wicentowski (2002) by 38–92%.

1 Introduction

A recurring problem in computational linguistics and language processing is transduction of character strings, e.g., words. That is, one wishes to model some systematic mapping from an input string x to an output string y. Applications include:

- *phonology*: underlying representation ↔ surface representation
- *orthography*: pronunciation ↔ spelling
- *morphology*: inflected form ↔ lemma, or differently inflected form
- *fuzzy name matching* (duplicate detection) and *spelling correction*: spelling ↔ variant spelling
- *lexical translation* (cognates, loanwords, transliterated names): English word ↔ foreign word

We present a configurable and robust framework for solving such word transduction problems. Our results in morphology generation show that the presented approach improves upon the state of the art.

2 Model Structure

A weighted edit distance model (Ristad and Yianilos, 1998) would consider each character in isolation. To consider more context, we pursue a very natural generalization. Given an input x, we evaluate a candidate output y by moving a sliding window over the aligned (x, y) pair. More precisely, since many alignments are possible, we sum over all these possibilities, evaluating each alignment *separately*.[1]

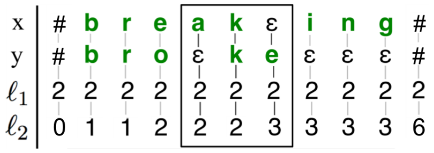
At each window position, we accumulate log-probability based on the material that appears within the current window. The window is a few characters wide, and successive window positions *overlap*. This stands in contrast to a competing approach (Sherif and Kondrak, 2007; Zhao et al., 2007) that is inspired by phrase-based machine translation (Koehn et al., 2007), which *segments* the input string into substrings that are transduced *independently*, ignoring context.[2]

[*] This work was supported by the Human Language Technology Center of Excellence and by National Science Foundation grant No. 0347822 to the final author. We would also like to thank Richard Wicentowski for providing us with datasets for lemmatization, and the anonymous reviewers for their valuable feedback.

[1] At the other extreme, Freitag and Khadivi (2007) use no alignment; each feature takes its own view of how (x, y) relate.

[2] We feel that this independence is inappropriate. By analogy, it would be a poor idea for a language model to score a string highly if it could be *segmented* into independently frequent n-grams. Rather, language models use overlapping n-grams (indeed, it is the language model that rescues phrase-based MT from producing disjointed translations). We believe phrase-based MT avoids overlapping phrases in the *channel* model only because these would complicate the modeling of reordering (though see, e.g., Schwenk et al. (2007) and Casacuberta (2000)). But in the problems of section 1, letter reordering is rare and we may assume it is local to a window.

Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing, pages 1080–1089,
Honolulu, October 2008. ©2008 Association for Computational Linguistics



x	#	**b**	**r**	**e**	**a**	**k**	ε	**i**	**n**	**g**	#
y	#	**b**	**r**	**o**	ε	**k**	**e**	ε	ε	ε	#
ℓ_1	2	2	2	2	2	2	2	2	2	2	2
ℓ_2	0	1	1	2	2	2	3	3	3	3	6

Figure 1: One of many possible alignment strings A for the observed pair *breaking*/*broke*, enriched with latent strings ℓ_1 and ℓ_2. Observed letters are shown in bold. The box marks a trigram to be scored. See Fig. 2 for features that fire on this trigram.

Joint n-gram models over the input and output dimensions have been used before, but not for morphology, where we will apply them.[3] Most notable is the local log-linear grapheme-to-phoneme model of Chen (2003), as well as generative models for that task (Deligne et al. (1995), Galescu and Allen (2001), Bisani and Ney (2002)).

We advance that approach by adding new *latent* dimensions to the (input, output) tuples (see Fig. 1).[4] This enables us to use certain linguistically inspired features and discover unannotated information. Our features consider *less* or *more* than a literal n-gram. On the one hand, we generalize with features that abstract away from the n-gram window contents; on the other, we specialize the n-gram with features that make use of the added latent linguistic structure.

In section 5, we briefly sketch our framework for concisely expressing and efficiently implementing models of this form. Our framework uses familiar log-linear techniques for stochastic modeling, and weighted finite-state methods both for implementation and for specifying features. It appears general enough to cover most prior work on word transduction. We imagine that it will be useful for future work as well: one might easily add new, linguistically interesting classes of features, each class defined by a regular expression.

2.1 Basic notation

We use an **input alphabet** Σ_{x} and **output alphabet** Σ_{y}. We conventionally use $x \in \Sigma_{\text{x}}^*$ to denote the input string and $y \in \Sigma_{\text{y}}^*$ to denote the output string.

There are many possible alignments between x and y. We represent each as an **alignment string** $A \in \Sigma_{\text{xy}}^*$, over an **alignment alphabet** of ordered pairs, $\Sigma_{\text{xy}} \overset{\text{def}}{=} ((\Sigma_{\text{x}} \cup \{\epsilon\}) \times (\Sigma_{\text{y}} \cup \{\epsilon\})) - \{(\epsilon, \epsilon)\}$.

For example, one alignment of $x = \texttt{breaking}$ with $y = \texttt{broke}$ is the 9-character string $A = (\texttt{b},\texttt{b})(\texttt{r},\texttt{r})(\texttt{e},\texttt{o})(\texttt{a},\epsilon)(\texttt{k},\texttt{k})(\epsilon,\texttt{e})(\texttt{i},\epsilon)(\texttt{n},\epsilon)(\texttt{g},\epsilon)$. It is pictured in the first two lines of Fig. 1.

The remainder of Fig. 1 shows how we introduce latent variables, by enriching the alignment characters to be tuples rather than pairs. Let $\Sigma \overset{\text{def}}{=} (\Sigma_{\text{xy}} \times \Sigma_{\ell_1} \times \Sigma_{\ell_2} \times \cdots \times \Sigma_{\ell_K})$, where Σ_{ℓ_i} are alphabets used for the latent variables ℓ_i.

FSA and **FST** stand for "finite-state acceptor" and "finite-state transducer," while **WFSA** and **WFST** are their weighted variants. The $\circ$ symbol denotes composition.

Let T be a relation and w a string. We write $T[w]$ to denote the image of w under T (i.e., range$(w \circ T)$), a set of 0 or more strings. Similarly, if W is a weighted language (typically encoded by a WFSA), we write $W[w]$ to denote the weight of w in W.

Let $\pi_{\text{x}} \subseteq \Sigma^* \times \Sigma_{\text{x}}^*$ denote the deterministic regular relation that projects an alignment string to its corresponding input string, so that $\pi_{\text{x}}[A] = x$. Similarly, define $\pi_{\text{y}} \subseteq \Sigma^* \times \Sigma_{\text{y}}^*$ so that $\pi_{\text{y}}[A] = y$. Let $\boldsymbol{A_{xy}}$ be the set of alignment strings A compatible with x and y; formally, $\boldsymbol{A_{xy}} \overset{\text{def}}{=} \{A \in \Sigma^* : \pi_{\text{x}}[A] = x \wedge \pi_{\text{y}}[A] = y\}$. This set will range over all possible alignments between x and y, and *also* all possible configurations of the latent variables.

2.2 Log-linear modeling

We use a standard log-linear model whose features are defined on alignment strings $A \in \boldsymbol{A_{xy}}$, allowing them to be sensitive to the alignment of x and y. Given a collection of features $f_i : \Sigma^* \to \mathbb{R}$ with associated weights $\theta_i \in \mathbb{R}$, the conditional likelihood of the training data is

$$p_{\boldsymbol{\theta}}(y \mid x) = \frac{\sum_{A \in \boldsymbol{A_{xy}}} \exp \sum_i \theta_i f_i(A)}{\sum_{y'} \sum_{A \in \boldsymbol{A_{xy'}}} \exp \sum_i \theta_i f_i(A)} \quad (1)$$

Given a parameter vector $\boldsymbol{\theta}$, we compute equation (1) using a finite-state machine. We define a WFSA, $U_{\boldsymbol{\theta}}$, such that $U_{\boldsymbol{\theta}}[A]$ yields the unnormalized probability $u_{\boldsymbol{\theta}}(A) \overset{\text{def}}{=} \exp \sum_i \theta_i f_i(A)$ for any $A \in \Sigma^*$. (See section 5 for the construction.) To obtain

[3] Clark (2001) does use pair HMMs for morphology.

[4] Demberg et al. (2007) similarly added extra dimensions. However, their added dimensions were supervised, not latent, and their model was a standard generative n-gram model whose generalization was limited to standard n-gram smoothing.

the numerator of equation (1), with its $\sum_{A\in\boldsymbol{A}_{\boldsymbol{xy}}}$, we sum over all paths in $U_{\boldsymbol{\theta}}$ that are compatible with x and y. That is, we build $x \circ \pi_{\mathrm{x}}^{-1} \circ U_{\boldsymbol{\theta}} \circ \pi_{\mathrm{y}} \circ y$ and sum over all paths. For the denominator we build the larger machine $x \circ \pi_{\mathrm{x}}^{-1} \circ U_{\boldsymbol{\theta}}$ and again compute the pathsum. We use standard algorithms (Eisner, 2002) to compute the pathsums as well as their gradients with respect to $\boldsymbol{\theta}$ for optimization (section 4.1).

Below, we will restrict our notion of *valid* alignment strings in Σ^*. $U_{\boldsymbol{\theta}}$ is constructed not to accept *invalid* ones, thus assigning them probability 0.

Note that the possible output strings y' in the denominator in equation (1) may have arbitrary length, leading to an infinite summation over alignment strings. Thus, for some values of $\boldsymbol{\theta}$, the sum in the denominator diverges and the probability distribution is undefined. There exist principled ways to avoid such $\boldsymbol{\theta}$ during training. However, in our current work, we simply restrict to finitely many alignment strings (given x), by prohibiting as invalid those with $> k$ *consecutive* insertions (i.e., characters like $(\epsilon, \mathtt{a})$).[5] Finkel et al. (2008) and others have similarly bounded unary rule cycles in PCFGs.

2.3 Latent variables

The alignment between x and y is a latent explanatory variable that helps model the distribution $p(y \mid x)$ but is not observed in training. Other latent variables can also be useful. Morphophonological changes are often sensitive to *phonemes* (whereas x and y may consist of graphemes); *syllable boundaries*; a *conjugation class*; *morpheme boundaries*; and the *position* of the change within the form.

Thus, as mentioned in section 2.1, we enrich the alignment string A so that it specifies additional latent variables to which features may wish to refer. In Fig. 1, two latent strings are added, enabling the features in Fig. 2(a)–(h). The first character is not just an input/output pair, but the 4-tuple $(\mathtt{b}, \mathtt{b}, \mathtt{2}, \mathtt{1})$.

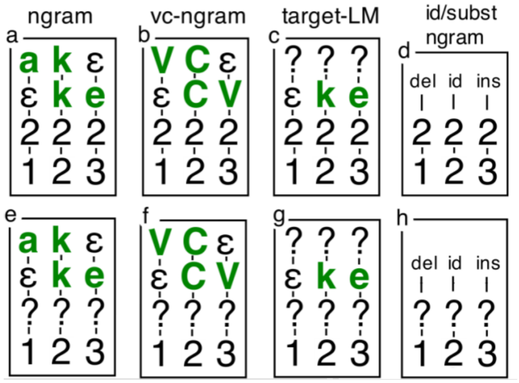


Figure 2: The boxes (a)-(h) represent some of the features that fire on the trigram shown in Fig. 1. These features are explained in detail in section 3.

Here, ℓ_1 indicates that this form pair (*breaking / broke*) as a whole is in a particular cluster, or *word class*, labeled with the arbitrary number $\mathtt{2}$. Notice in Fig. 1 that the class $\mathtt{2}$ is visible in all local windows throughout the string. It allows us to model how certain phenomena, e.g. the vowel change from $\mathtt{ea}$ to $\mathtt{o}$, are more likely in one class than in another. Form pairs in the same class as the *breaking / broke* example might include the following Germanic verbs: *speak*, *break*, *steal*, *tear*, and *bear*.

Of course, word classes are latent (not labeled in our training data). Given x and y, $\boldsymbol{A}_{\boldsymbol{xy}}$ will include alignment strings that specify class $\mathtt{1}$, and others that are identical except that they specify class $\mathtt{2}$; equation (1) sums over both possibilities.[6] In a valid alignment string A, ℓ_1 must be a constant string such as $\mathtt{111}\ldots$ or $\mathtt{222}\ldots$, as in Fig. 1, so that it specifies a single class for the entire form pair. See sections 4.2 and 4.3 for examples of what classes were learned in our experiments.

The latent string ℓ_2 splits the string pair into numbered *regions*. In a valid alignment string, the region numbers must increase throughout ℓ_2, although numbers may be skipped to permit omitted regions. To guide the model to make a useful division into regions, we also require that identity characters such as $(\mathtt{b}, \mathtt{b})$ fall in even regions while change characters such as $(\mathtt{e}, \mathtt{o})$ (substitutions, deletions, or inser-

[5] We set k to a value between 1 and 3, depending on the tasks, always ensuring that no input/output pairs observed in training are excluded. The insertion restriction does slightly enlarge the FSA U_θ: a state must keep track of the number of consecutive ϵ symbols in the immediately preceding x input, and for a few states, this cannot be determined just from the immediately preceding $(n-1)$-gram. Despite this, we found empirically that our approximation is at least as fast as the exact method of Eisner (2002), who sums around cyclic subnetworks to numerical convergence. Furthermore, our approximation does not require us to detect divergence during training.

[6] The latent class is comparable to the latent variable on the tree root symbol *S* in Matsuzaki et al. (2005).

tions) fall in odd regions.[7] Region numbers must not increase within a sequence of consecutive changes or consecutive identities.[8] In Fig. 1, the start of region 1 is triggered by `e`:`o`, the start of region 2 by the identity `k`:`k`, region 3 by ϵ:`e`.

Allowing region numbers to be skipped makes it possible to consistently assign similar labels to similar regions across different training examples. Table 2, for example, shows pairs that contain a vowel change in the middle, some of which contain an additional insertion of `ge` in the begining (*verbinden / verbunden*, *reibt / gerieben*). We expect the model to learn to label the `ge` insertion with a 1 and vowel change with a 3, skipping region 1 in the examples where the `ge` insertion is not present (see section 4.2, Analysis).

In the next section we describe features over these enriched alignment strings.

3 Features

One of the simplest ways of scoring a string is an n-gram model. In our log-linear model (1), we include ngram features $f_i(A)$, each of which counts the occurrences in A of a particular n-gram of alignment characters. The log-linear framework lets us include ngram features of different lengths, a form of backoff smoothing (Wu and Khudanpur, 2000).

We use additional backoff features on alignment strings to capture phonological, morphological, and orthographic generalizations. Examples are found in features (b)-(h) in Fig. 2. Feature (b) matches *vowel and consonant* character classes in the input and output dimensions. In the id/subst ngram feature, we have a similar abstraction, where the character classes ins, del, id, and subst are defined over input/output pairs, to match insertions, deletions, identities (matches), and substitutions.

In string transduction tasks, it is helpful to include a language model of the target. While this can be done by mixing the transduction model with a separate language model, it is desirable to include a target language model within the transduction model. We accomplish this by creating target language model features, such as (c) and (g) from Fig. 2, which ignore the input dimension. We also have features which mirror features (a)-(d) but ignore the latent classes and/or regions (e.g. features (e)-(h)).

Notice that our choice of Σ only permits monotonic, 1-to-1 alignments, following Chen (2003). We may nonetheless favor the 2-to-1 alignment (`ea`,`o`) with bigram features such as (`e`,`o`)(`a`,ϵ). A "collapsed" version of a feature will back off from the specific alignment of the characters within a window: thus, (`ea`,`o`) is itself a feature. Currently, we only include collapsed target language model features. These ignore epsilons introduced by deletions in the alignment, so that collapsed `ok` fires in a window that contains `o`ϵ`k`.

4 Experiments

We evaluate our model on two tasks of morphology generation. Predicting morphological forms has been shown to be useful for machine translation and other tasks.[9] Here we describe two sets of experiments: an inflectional morphology task in which models are trained to transduce verbs from one form into another (section 4.2), and a lemmatization task (section 4.3), in which *any* inflected verb is to be reduced to its root form.

4.1 Training and decoding

We train $\boldsymbol{\theta}$ to maximize the regularized[10] conditional log-likelihood[11]

$$\sum_{(x,y^*)\in\mathcal{C}} \log p_{\boldsymbol{\theta}}(y^* \mid x) + ||\boldsymbol{\theta}||^2/2\sigma^2, \qquad (2)$$

where $\mathcal{C}$ is a supervised training corpus. To maximize (2) during training, we apply the gradient-based optimization method L-BFGS (Liu and Nocedal, 1989).[12]

[7] This strict requirement means, perhaps unfortunately, that a single region cannot accommodate the change `ayc`:`xyz` unless the two `y`'s are not aligned to each other. It could be relaxed, however, to a prior or an initialization or learning bias.

[8] The two boundary characters `#`, numbered 0 and max (max=6 in our experiments), are neither changes nor identities.

[9] E.g., Toutanova et al. (2008) improve MT performance by *selecting* correct morphological forms from a knowledge source. We instead focus on generalizing from observed forms and *generating* new forms (but see *with rootlist* in Table 3).

[10] The variance σ^2 of the L_2 prior is chosen by optimizing on development data. We are also interested in trying an L_1 prior.

[11] Alternatives would include faster error-driven methods (perceptron, MIRA) and slower max-margin Markov networks.

[12] This worked a bit better than stochastic gradient descent.

To decode a test example x, we wish to find $\hat{y} = \mathrm{argmax}_{y \in \Sigma_{\mathrm{y}}^*} p_{\boldsymbol{\theta}}(y \mid x)$. Constructively, $\hat{y}$ is the highest-probability string in the WFSA $T[x]$, where $T = \pi_{\mathrm{x}}^{-1} \circ U_{\boldsymbol{\theta}} \circ \pi_{\mathrm{y}}$ is the trained transducer that maps x nondeterministically to y. Alas, it is NP-hard to find the highest-probability string in a WFSA, even an acyclic one (Casacuberta and Higuera, 2000). The problem is that the probability of each string y is a sum over many paths in $T[x]$ that reflect different alignments of y to x. Although it is straightforward to use a determinization construction (Mohri, 1997)[13] to collapse these down to a single path per y (so that $\hat{y}$ is easily read off the single best path), determinization can increase the WFSA's size exponentially. We approximate by pruning $T[x]$ back to its 1000-best paths before we determinize.[14]

Since the alignments, classes and regions are not observed in $\mathcal{C}$, we do not enjoy the convex objective function of fully-supervised log-linear models. Training equation (2) therefore converges only to some *local* maximum that depends on the starting point in parameter space. To find a good starting point we employ *staged training*, a technique in which several models of ascending complexity are trained consecutively. The parameters of each more complex model are initialized with the trained parameters of the previous simpler model.

Our training is done in four stages. All weights are initialized to zero. ① We first train only features that fire on unigrams of alignment characters, ignoring features that examine the latent strings or backed-off versions of the alignment characters (such as vowel/consonant or target language model features). The resulting model is equivalent to weighted edit distance (Ristad and Yianilos, 1998). ② Next,[15] we train all n-grams of alignment characters, including higher-order n-grams, but no backed-off features or features that refer to latent strings. ③ Next, we add backed-off features as well as all collapsed features. ④ Finally, we train all features. In our experiments, we permitted latent classes `1`–`2` and, where regions are used, regions `0`–`6`. For speed, stages ②–④ used a pruned Σ that included only "plausible" alignment characters: a may not align to b unless it did so in the trained stage-(1) model's optimal alignment of *at least one* training pair (x, y^*).

13SIA.	lieb**te**,	pick**te**,	red**ete**,	r**ieb**,	tr**ieb**,	zu**zog**
13SKE.	lieb**e**,	pick**e**,	red**e**,	r**eibe**,	tr**eibe**,	zu**ziehe**
2PIE.	lieb**t**,	pick**t**,	red**et**,	reib**t**,	treib**t**,	zuzieh**t**
13PKE.	lieb**en**,	pick**en**,	red**en**,	reib**en**,	treib**en**,	zuzieh**en**
2PKE.	abbreche**t**,	entgegentrete**t**,	zuziehe**t**			
z.	ab**zu**breche**n**,	entgegen**zu**trete**n**,	zu**zu**ziehe**n**			
rP.	redet,	**reibt**,	tr**eibt**,	verb**i**nde**t**,	überfischt	
pA.	**ge**redet,	**gerieben**,	**ge**tr**ieben**,	verb**u**nde**n**,	überfischt	

Table 2: CELEX forms used in our experiments. Changes from one form to the other are in bold (information not given in training). The changes from rP to pA are very complex. Note also the differing positions of `zu` in z.

4.2 Inflectional morphology

We conducted several experiments on the CELEX morphological database. We arbitrarily considered mapping the following German verb forms:[16] 13SIA → 13SKE, 2PIE → 13PKE, 2PKE → z, and rP → pA.[17] We refer to these tasks as 13SIA, 2PIE, 2PKE and rP. Table 2 shows some examples of regular and irregular forms. Common phenomena include stem changes (`ei:ie`), prefixes inserted after other morphemes (*ab**zu**brechen*) and circumfixes (***ge**rieb**en***).

We compile lists of form pairs from CELEX. For each task, we sample 2500 data pairs without replacement, of which 500 are used for training, 1000 as development and the remaining 1000 as test data. We train and evaluate models on this data and repeat

[13] Weighted determinization is not always possible, but it is in our case because our limit to k consecutive insertions guarantees that $T[x]$ is acyclic.

[14] This value is high enough; we see no degradations in performance if we use only 100 or even 10 best paths. Below that, performance starts to drop slightly. In both of our tasks, our conditional distributions are usually peaked: the 5 best output candidates amass $> 99\%$ of the probability mass on average. Entropy is reduced by latent classes and/or regions.

[15] When unclamping a feature at the start of stages ②–④, we initialize it to a random value from $[-0.01, 0.01]$.

[16] From the available languages in CELEX (German, Dutch, and English), we selected German as the language with the most interesting morphological phenomena, leaving the multilingual comparison for the lemmatization task (section 4.3), where there were previous results to compare with. The 4 German datasets were picked arbitrarily.

[17] A key to these names: 13SIA=*1st/3rd sg. ind. past*; 13SKE=*1st/3rd sg. subjunct. pres.*; 2PIE=*2nd pl. ind. pres.*; 13PKE=*1st/3rd pl. subjunct. pres.*; 2PKE=*2nd pl. subjunct. pres.*; z=*infinitive*; rP=*imperative pl.*; pA=*past part.*

	Features							**Task**			
	ng	vc	tlm	tlm-coll	id	lat.cl.	lat.reg.	13SIA	2PIE	2PKE	rP
ngrams	x							82.3 (.23)	88.6 (.11)	74.1 (.52)	70.1 (.66)
ngrams+x	x				x			82.8 (.21)	88.9 (.11)	74.3 (.52)	70.0 (.68)
	x		x					82.0 (.23)	88.7 (.11)	74.8 (.50)	69.8 (.67)
	x		x		x			82.5 (.22)	88.6 (.11)	74.9 (.50)	70.0 (.67)
	x		x	x				81.2 (.24)	88.7 (.11)	74.5 (.50)	68.6 (.69)
	x		x	x	x			82.5 (.22)	88.8 (.11)	74.5 (.50)	69.2 (.69)
	x	x						82.4 (.22)	88.9 (.11)	74.8 (.51)	69.9 (.68)
	x	x			x			83.0 (.21)	88.9 (.11)	74.9 (.50)	70.3 (.67)
	x	x	x					82.2 (.22)	88.8 (.11)	74.8 (.50)	70.0 (.67)
	x	x	x		x			82.9 (.21)	88.6 (.11)	75.2 (.50)	69.7 (.68)
	x	x	x	x				81.9 (.23)	88.6 (.11)	74.4 (.51)	69.1 (.68)
	x	x	x	x	x			82.8 (.21)	88.7 (.11)	74.7 (.50)	69.9 (.67)
ngrams+x +latent	x	x	x	x	x	x		84.8 (.19)	**93.6 (.06)**	75.7 (.48)	81.8 (.43)
	x	x	x	x	x		x	**87.4 (.16)**	**93.8 (.06)**	**88.0 (.28)**	83.7 (.42)
	x	x	x	x	x	x	x	**87.5 (.16)**	93.4 (.07)	**87.4 (.28)**	**84.9 (.39)**
Moses3								73.9 (.40)	92.0 (.09)	67.1 (.70)	67.6 (.77)
Moses9								85.0 (.21)	**94.0 (.06)**	82.3 (.31)	70.8 (.67)
Moses15								85.3 (.21)	**94.0 (.06)**	82.8 (.30)	70.8 (.67)

Table 1: Exact-match accuracy and average edit distance (the latter in parentheses) versus the correct answer on the German inflection task, using different combinations of feature classes. The label *ngrams* corresponds to the second stage of training, *ngrams+x* to the third where backoff features may fire (vc = vowel/consonant, tlm = target LM, tlm-coll = collapsed tlm, id = identity/substitution/deletion features), and *ngrams+x+latent* to the fourth where features sensitive to latent classes and latent regions are allowed to fire. The highest n-gram order used is 3, except for *Moses9* and *Moses15* which examine windows of up to 9 and 15 characters, respectively. We mark in **bold** the best result for each dataset, along with all results that are statistically indistinguishable (paired permutation test, $p < 0.05$).

the process 5 times. All results are averaged over these 5 runs.

Table 1 and Fig. 3 report separate results after stages ②, ③, and ④ of training, which include successively larger feature sets. These are respectively labeled *ngrams*, *ngrams+x*, and *ngrams+x+latent*. In Table 1, the last row in each section shows the full feature set at that stage (cf. Fig. 3), while earlier rows test feature subsets.[18]

Our baseline is the SMT toolkit Moses (Koehn et al., 2007) run over letter strings rather than word strings. It is trained (on the same data splits) to find substring-to-substring phrase pairs and translate from one form into another (with phrase reordering turned off). Results reported as *moses3* are obtained from Moses runs that are constrained to the same context windows that our models use, so the maximum phrase length and the order of the target language model were set to 3. We also report results using much larger windows, *moses9* and *moses15*.

Results. The results in Table 1 show that including *latent classes and/or regions* improves the results dramatically. Compare the last line in *ngrams+x* to the last line in *ngrams+x+latent*. The accuracy numbers improve from 82.8 to 87.5 (13SIA), from 88.7 to 93.4 (2PIE), from 74.7 to 87.4 (2PKE), and from 69.9 to 84.9 (rP).[19] This shows that error reductions between 27% and 50% were reached. On 3 of 4 tasks, even our simplest *ngrams* method beats the *moses3* method that looks at the same amount of context.[20] With our full model, in particular using latent features, we always outperform *moses3*—and even outperform *moses15* on 3 of the 4 datasets, reducing the error rate by up to 48.3% (rP). On the fourth task (2PIE), our method and *moses15* are statistically tied. *Moses15* has access to context windows of five times the size than we allowed our methods in our experiments.

[18] The number k of consecutive insertions was set to 3.

[19] All claims in the text are statistically significant under a paired permutation test ($p < .05$).

[20] This bears out our contention in footnote 2 that a "segmenting" channel model is damaging. Moses cannot fully recover by using overlapping windows in the *language* model.

While the gains from *backoff features* in Table 1 were modest (significant gains only on 13SIA), the learning curve in Fig. 3 suggests that they were helpful for smaller training sets on 2PKE (see *ngrams* vs *ngrams+x* on 50 and 100) and helped consistently over different amounts of training data for 13SIA.

Analysis. The types of errors that our system (and the moses baseline) make differ from task to task. Due to lack of space, we mainly focus on the complex rP task. Here, most errors come from wrongly copying the input to the output, without making a change (40-50% of the errors in all models, except for our model with latent classes and no regions, where it accounts for only 30% of the errors). This is so common because about half of the training examples contain identical inputs and outputs (as in the imperative *berechnet* and the participle *(ihr habt) berechnet*). Another common error is to wrongly assume a regular conjugation (just insert the prefix *ge-* at the beginning). Interestingly, this error by simplification is more common in the Moses models (44% of moses3 errors, down to 40% for moses15) than in our models, where it accounts for 37% of the errors of our *ngrams* model and only 19% if latent classes *or* latent regions are used; however, it goes up to 27% if both latent classes and regions are used.[21] All models for rP contain errors where wrong analogies to observed words are made (*verschweisst/verschwissen* in analogy to the observed *durchweicht/durchwichen*, or *bebt/geboben* in analogy to *hebt/gehoben*). In the 2PKE task, most errors result from inserting the *zu* morpheme at a wrong place or inserting two of them, which is always wrong. This error type was greatly reduced by latent regions, which can discover different parameters for different positions, making it easier to identify where to insert the *zu*.

Analysis of the 2 latent classes (when used) shows that a *split into regular and irregular conjugations* has been learned. For the rP task we compute, for each data pair in development data, the posterior probabilities of membership in one or the other class. 98% of the regular forms, in which the past participle is built with *ge-* … *-t*, fall into one class, which in turn consists nearly exclusively (96%) of these forms. Different irregular forms are lumped into the other class.

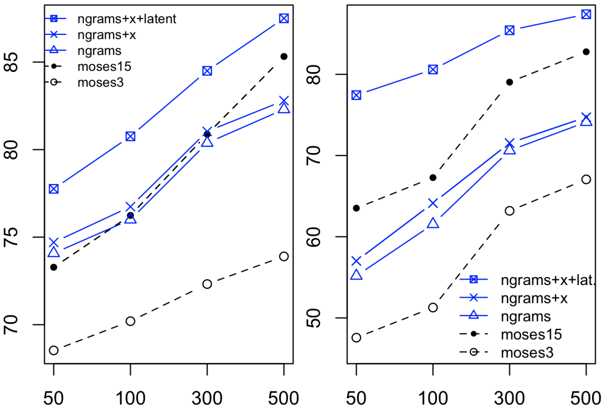


Figure 3: Learning curves for German inflection tasks, 13SIA (left) and 2PKE (right), as a function of the number of training pairs. *ngrams+x* means all backoff features were used, *ngrams+x+latent* means all latent features were used in addition. *Moses15* examines windows of up to 15 characters.

The learned regions are consistent across different pairs. On development data for the rP task, 94.3% of all regions that are labeled 1 are the insertion sequence (ϵ,ge), region 3 consists of vowel changes 93.7% of the time; region 5 represents the typical suffixes (t,en), (et,en), (t,n) (92.7%). In the *2PKE* task, region 0 contains different prefixes (e.g. *entgegen* in *entgegenzutreten*), regions 1 and 2 are empty, region 3 contains the *zu* affix, region 4 the stem, and region 5 contains the suffix.

The pruned alignment alphabet excluded a few gold standard outputs so that the model contains paths for 98.9%–99.9% of the test examples. We verified that the insertion limit did not hurt oracle accuracy.

4.3 Lemmatization

We apply our models to the task of lemmatization, where the goal is to generate the lemma given an inflected word form. We compare our model to Wicentowski (2002, chapter 3), an alternative supervised approach. Wicentowski's *Base* model simply learns how to replace an arbitrarily long suffix string of an input word, choosing some previously observed suffix → suffix replacement based on the input word's

[21] We suspect that training of the models that use classes and regions together was hurt by the increased non-convexity; annealing or better initialization might help.

	Without rootlist (generation)						With rootlist (selection)					
	Wicentowski (2002)			This paper			Wicentowski (2002)			This paper		
Lang.	Base	Af.	WFA.	n	n+x	n+x+l	Base	Af.	WFA.	n	n+x	n+x+l
Basque	85.3	81.2	80.1	91.0 (.20)	91.1 (.20)	**93.6 (.14)**	94.5	94.0	**95.0**	90.9 (.29)	90.8 (.31)	90.9 (.30)
English	91.0	94.7	93.1	92.4 (.09)	93.4 (.08)	**96.9 (.05)**	98.3	98.6	98.6	**98.7 (.04)**	**98.7(.04)**	**98.7(.04)**
Irish	43.3	-	70.8	96.8 (.07)	97.0 (.06)	**97.8 (.04)**	43.9	-	89.1	**99.6 (.02)**	**99.6 (.02)**	**99.5 (.03)**
Tagalog	0.3	80.3	81.7	80.5 (.32)	83.0 (.29)	**88.6 (.19)**	0.8	91.8	96.0	97.0 (.07)	97.2 (.07)	**97.7 (.05)**

Table 3: Exact-match accuracy and average edit distance (the latter in parentheses) on the 8 lemmatization tasks (2 tasks × 4 languages). The numbers from Wicentowski (2002) are for his Base, Affix and WFAffix models. The numbers for our models are for the feature sets *ngrams*, *ngrams+x*, *ngrams+x+latent*. The best result per task is in **bold** (as are statistically indistinguishable results when we can do the comparison, i.e., for our own models). Corpus sizes: Basque 5,842, English 4,915, Irish 1,376, Tagalog 9,479.

final n characters (interpolating across different values of n). His *Affix* model essentially applies the Base model after stripping canonical prefixes and suffixes (given by a user-supplied list) from the input and output. Finally, his *WFAffix* uses similar methods to also learn substring replacements for a stem vowel cluster and other linguistically significant regions in the form (identified by a *deterministic* alignment and segmentation of training pairs). This approach is a bit like our change regions combined with Moses's region-independent phrase pairs.

We compare against all three models. Note that *Affix* and *WFAffix* have an advantage that our models do not, namely, user-supplied lists of canonical affixes for each language. It is interesting to see how our models with their more non-committal trigram structure compare to this. Table 3 reports results on the data sets used in Wicentowski (2002), for Basque, English, Irish, and Tagalog. Following Wicentowski, 10-fold cross-validation was used. The columns *n+x* and *n+x+l* mean *ngram+x* and *ngram+x+latent*, respectively. As latent variables, we include 2 word classes but no change regions.[22]

For completeness, Table 3 also compares with Wicentowski (2002) on a *selection* (rather than generation) task. Here, at test time, the lemma is selected from a candidate list of known lemmas, namely, all the output forms that appeared in training data.[23] These additional results are labeled *with rootlist* in the right half of Table 3.

On the supervised generation task *without rootlist*, our models outperform Wicentowski (2002) by a large margin. Comparing our results that use latent classes (*n+x+l*) with Wicentowski's best models we observe error reductions ranging from about 38% (Tagalog) to 92% (Irish). On the selection task *with rootlist*, we outperform Wicentowski (2002) in English, Irish, and Tagalog.

Analysis. We examined the classes learned on English lemmatization by our *ngrams+x+latent* model. For each of the input/output pairs in development data, we found the most probable latent class. For the most part, the 2 classes are separated based on whether or not the correct output ends in e. This use of latent classes helped address many errors like *wronging / wronge* or *owed / ow*). Such missing or surplus final e's account for 72.5% of the errors for *ngrams* and 70.6% of the errors for *ngrams+x*, but only 34.0% of the errors for *ngrams+x+latent*.

The test oracles are between 99.8% – 99.9%, due to the pruned alignment alphabet. As on the inflection task, the insertion limit does not exclude any gold standard paths.

5 Finite-State Feature Implementation

We used the OpenFST library (Allauzen et al., 2007) to implement all finite-state computations, using the expectation semiring (Eisner, 2002) for training.

Our model is defined by the WFSA $U_{\boldsymbol{\theta}}$, which is used to score alignment strings in Σ^* (section 2.2). We now sketch how to construct $U_{\boldsymbol{\theta}}$ from features.

n-gram construction The construction that we currently use is quite simple. All of our current features fire on windows of width $\leq$ 3. We build a WFSA with the structure of a 3-gram language

[22] The insertion limit k was set to 2 for Basque and 1 for the other languages.

[23] Though test data contained no (input, output) pairs from training data, it reused many of the output forms, since many inflected inputs are to be mapped to the same output lemma.

model over Σ^*. Each of the $|\Sigma|^2$ states remembers two previous alignment characters ab of history; for each $c \in \Sigma$, it has an outgoing arc that accepts c (and leads to state bc). The weight of this arc is the total weight (from θ) of the small set of features that fire when the trigram window includes abc. By convention, these also include features on bc and c (which may be regarded as backoff features $?bc$ and $??c$). Since each character in Σ is actually a 4-tuple, this trigram machine is fairly large. We build it lazily ("on the fly"), constructing arcs only as needed to deal with training or test data.

Feature templates Our experiments use over 50,000 features. How do we *specify* these features to the above construction? Rather than writing ordinary code to extract features from a window, we find it convenient to harness FSTs as a "little language" (Bentley, 1986) for specifying entire sets of features.

A **feature template** T is an nondeterministic FST that maps the contents of the sliding window, such as abc, to one or more **features**, which are also described as strings.[24] The n-gram machine described above can compute $T[((\texttt{a}^?\texttt{b})^?\texttt{c})^?]$ to find out what features fire on abc and its suffixes. One simple feature template performs "vowel/consonant backoff"; e.g., it maps abc to the feature named VCC. Fig. 2 showed the result of applying several actual feature templates to the window shown in Fig. 1. The extended regular expression calculus provides a flexible and concise notation for writing down these FSTs. As a trivial example, the trigram "vowel/consonant backoff" transducer can be described as $T = VVV$, where V is a transducer that performs backoff on a *single* alignment character. Feature templates should make it easy to experiment with adding various kinds of linguistic knowledge. We have additional algorithms for compiling $U_{\boldsymbol{\theta}}$ from a set of *arbitrary* feature templates,[25] including templates whose features consider windows of variable or even unbounded width. The details are beyond the scope of this paper, but it is worth pointing out that they exploit the fact that feature templates are FSTs and not arbitrary code.

6 Conclusions

The modeling framework we have presented here is, we believe, an attractive solution to most string transduction problems in NLP. Rather than learn the topology of an arbitrary WFST, one specifies the topology using a small set of feature templates, and simply trains the weights.

We evaluated on two morphology generation tasks. When inflecting German verbs we, even with the simplest features, outperform the *moses3* baseline on 3 out of 4 tasks, which uses the same amount of context as our models. Introducing more sophisticated features that have access to latent classes and regions improves our results dramatically, even on small training data sizes. Using these we outperform *moses9* and *moses15*, which use long context windows, reducing error rates by up to 48%. On the lemmatization task we were able to improve the results reported in Wicentowski (2002) on three out of four tested languages and reduce the error rates by 38% to 92%. The model's errors are often reasonable misgeneralizations (e.g., assume regular conjugation where irregular would have been correct), and it is able to use even a small number of latent variables (including the latent alignment) to capture useful linguistic properties.

In future work, we would like to identify a set of features, latent variables, and training methods that port well across languages and string-transduction tasks. We would like to use features that look at wide context on the *input* side, which is inexpensive (Jiampojamarn et al., 2007). Latent variables we wish to consider are an increased number of word classes; more flexible regions—see Petrov et al. (2007) on learning a state transition diagram for *acoustic* regions in phone recognition—and phonological features and syllable boundaries. Indeed, our local log-linear features over several aligned latent strings closely resemble the soft constraints used by phonologists (Eisner, 1997). Finally, rather than define a fixed set of feature templates as in Fig. 2, we would like to refine empirically useful features during training, resulting in language-specific backoff patterns and adaptively sized n-gram windows. Many of these enhancements will increase the computational burden, and we are interested in strategies to mitigate this, including approximation methods.

[24] Formally, if i is a string naming a feature, then $f_i(A)$ counts the number of positions in A that are immediately preceded by some string in $T^{-1}[i]$.

[25] Provided that the total number of features is finite.

References

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *Proc. of CIAA*, volume 4783.

Jon Bentley. 1986. Programming pearls [column]. *Communications of the ACM*, 29(8), August.

Maximilian Bisani and Hermann Ney. 2002. Investigations on jointmultigram models for grapheme-to-phoneme conversion.

Francisco Casacuberta and Colin De La Higuera. 2000. Computational complexity of problems on probabilistic grammars and transducers. In *Proc. of the 5th International Colloquium on Grammatical Inference: Algorithms and Applications*, volume 1891.

Francisco Casacuberta. 2000. Inference of finite-state transducers by using regular grammars and morphisms. In A.L. Oliveira, editor, *Grammatical Inference: Algorithms and Applications*, volume 1891.

Stanley F. Chen. 2003. Conditional and joint models for grapheme-to-phoneme conversion. In *Proc. of Interspeech*.

Alexander Clark. 2001. Learning morphology with Pair Hidden Markov Models. In *Proc. of the Student Workshop at the 39th Annual Meeting of the Association for Computational Linguistics*, Toulouse, France, July.

Sabine Deligne, Francois Yvon, and Frédéric Bimbot. 1995. Variable-length sequence matching for phonetic transcription using joint multigrams. In *Eurospeech*.

Vera Demberg, Helmut Schmid, and Gregor Möhler. 2007. Phonological constraints and morphological preprocessing for grapheme-to-phoneme conversion. In *Proc. of ACL*, Prague, Czech Republic, June.

Jason Eisner. 1997. Efficient generation in primitive Optimality Theory. In *Proc. of ACL-EACL*, Madrid, July.

Jason Eisner. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proc. of ACL*.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL-08: HLT*, pages 959–967, Columbus, Ohio, June. Association for Computational Linguistics.

Dayne Freitag and Shahram Khadivi. 2007. A sequence alignment model based on the averaged perceptron. In *Proc. of EMNLP-CoNLL*.

Lucian Galescu and James F. Allen. 2001. Bi-directional conversion between graphemes and phonemes using a joint N-gram model.

Sittichai Jiampojamarn, Grzegorz Kondrak, and Tarek Sherif. 2007. Applying many-to-many alignments and hidden markov models to letter-to-phoneme conversion. In *Proc. of NAACL-HLT*, Rochester, New York, April.

Philipp Koehn, 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 *Proc. of ACL, Companion Volume*, Prague, Czech Republic, June.

Dong C. Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Math. Programming*, 45(3, (Ser. B)):503–528.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proc. of ACL*, Ann Arbor, Michigan, June.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2).

Slav Petrov, Adam Pauls, and Dan Klein. 2007. Learning structured models for phone recognition. In *Proc. of EMNLP-CoNLL*.

Eric Sven Ristad and Peter N. Yianilos. 1998. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5).

Holger Schwenk, Marta R. Costa-jussa, and Jose A. R. Fonollosa. 2007. Smooth bilingual n-gram translation. In *Proc. of EMNLP-CoNLL*, pages 430–438.

Tarek Sherif and Grzegorz Kondrak. 2007. Substring-based transliteration. In *Proc. of ACL*, Prague, Czech Republic, June.

Kristina Toutanova, Hisami Suzuki, and Achim Ruopp. 2008. Applying morphology generation models to machine translation. In *Proceedings of ACL-08: HLT*, Columbus, Ohio, June.

Richard Wicentowski. 2002. *Modeling and Learning Multilingual Inflectional Morphology in a Minimally Supervised Framework*. Ph.D. thesis, Johns-Hopkins University.

Jun Wu and Sanjeev Khudanpur. 2000. Efficient training methods for maximum entropy language modeling. In *Proc. of ICSLP*, volume 3, Beijing, October.

Bing Zhao, Nguyen Bach, Ian Lane, and Stephan Vogel. 2007. A log-linear block transliteration model based on bi-stream HMMs. In *Proc. of NAACL-HLT*, Rochester, New York, April.