# Hard Non-Monotonic Attention for Character-Level Transduction

**Shijie Wu**[ə] **Pamela Shapiro**[ə] **Ryan Cotterell**[ə,fi]
[ə]Department of Computer Science, Johns Hopkins University, Baltimore, USA
[fi]The Computer Laboratory, University of Cambridge, Cambridge, UK
{shijie.wu, pshapiro, ryan.cotterell}@jhu.edu

## Abstract

Character-level string-to-string transduction is an important component of various NLP tasks. The goal is to map an input string to an output string, where the strings may be of different lengths and have characters taken from different alphabets. Recent approaches have used sequence-to-sequence models with an attention mechanism to learn which parts of the input string the model should focus on during the generation of the output string. Both soft attention and hard monotonic attention have been used, but hard non-monotonic attention has only been used in other sequence modeling tasks such as image captioning (Xu et al., 2015) and has required a stochastic approximation to compute the gradient. In this work, we introduce an exact, polynomial-time algorithm for marginalizing over the exponential number of non-monotonic alignments between two strings, showing that hard attention models can be viewed as neural reparameterizations of the classical IBM Model 1. We compare soft and hard non-monotonic attention experimentally and find that the exact algorithm significantly improves performance over the stochastic approximation and outperforms soft attention.

## 1 Introduction

Many natural language tasks are expressible as string-to-string transductions operating at the character level. Probability models with recurrent neural parameterizations currently hold the state of the art on many such tasks. On those string-to-string transduction tasks that involve a mapping between two strings of different lengths, it is often necessary to resolve which input symbols are related to which output symbols. As an example, consider the task of transliterating a Russian word into the Latin alphabet. In many cases, there exists a one-to-two mapping between Cyrillic and Latin letters: in Хрущёв (*Khrushchev*), the Russian Х can be considered to generate the Latin letters *Kh*. Supervision is rarely, if ever, provided at the level of character-to-character alignments—this is the problem that attention seeks to solve in neural models.

With the rise of recurrent neural networks, this problem has been handled with "soft" **attention** rather than traditional hard **alignment**. Attention (Bahdanau et al., 2015) is often described as "soft," as it does not clearly associate a single input symbol with each output symbol, but rather offers a fuzzy notion of what input symbols may be responsible for which symbols in the output. In contrast, an alignment directly associates a given input symbol with a given output symbol. To express uncertainty, practitioners often place a distribution over the exponential number of hard non-monotonic alignments, just as a probabilistic parser places a distribution over an exponential number of trees. The goal, then, is to learn the parameters of this distribution over all non-monotonic alignments through backpropagation. Incorporating hard alignment into probabilistic transduction models dates back much farther in the NLP literature; arguably, originating with the seminal paper by Brown et al. (1993). Some neural approaches have moved back towards this approach of a more rigid alignment, referring to it as "hard attention." We will refer to this as "hard attention" and to more classical approaches as "alignment."

This paper offers two insights into the usage of hard alignment. First, we derive a dynamic program for the exact computation of the likelihood in a neural model with latent hard alignment: Previous work has used a stochastic algorithm to approximately sum over the exponential number of alignments between strings. In so doing, we go on to relate neural hard alignment models to the classical IBM Model 1 for alignment in machine translation. Second, we provide an experimental comparison that indicates hard attention models outperform soft attention models on three character-level string-to-string transduction tasks: grapheme-to-phoneme conversion, named-entity transliteration and morphological inflection.


*Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4425–4438
Brussels, Belgium, October 31 - November 4, 2018. 

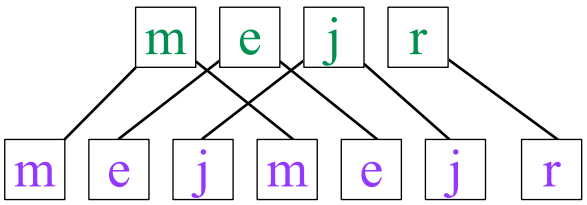


Figure 1: Example of a non-monotonic character-level transduction from the Micronesian language of Pingelapese. The infinitive *mejr* is mapped through a reduplicative process to its gerund *mejmejr* (Rehg and Sohl, 1981). Each input character is drawn in green and each output character is drawn in purple, connected with a line to the corresponding input character.

## 2 Non-Monotonic Transduction

This paper presents a novel, neural, probabilistic latent-variable model for **non-monotonic** transduction. As a concrete example of a non-monotonic transduction, consider the mapping of a Pingelapese infinitive to its gerund, as shown in Fig. 1. The mapping requires us to generate the output string left-to-right, bouncing around the input string out-of-order to determine the characters to transduce from. As the non-monotonic alignment is the latent variable, we will face a combinatorial problem: summing over all non-monotonic alignments. The algorithmic contribution of this paper is the derivation of a simple dynamic program for computing this sum in polynomial time that still allows for very rich recurrent neural featurization of the model. With respect to the literature, our paper represents the first instance of exact marginalization for a neural transducer with hard non-monotonic alignment; previous methods, such as Rastogi et al. (2016) and Aharoni and Goldberg (2017), are exclusively monotonic.

Non-monotonic methods dominate character-level transduction. Indeed, the state of art in classic character-level NLP tasks such as grapheme-to-phoneme conversion (Yao and Zweig, 2015), transliteration (Rosca and Breuel, 2016) and morphological inflection generation (Kann and Schütze, 2016) is held by the soft non-monotonic method of Bahdanau et al. (2015). Even though non-monotonicity is more common in word-level tasks, it also exists in character-level transduction tasks, as evidenced by our example in Fig. 1 and the superior performance of non-monotonic methods. Our error analysis in §8.4 sheds some light on why non-monotonic methods are the state of the art in a seemingly monotonic task.

**A Note on the Character-level Focus.** A natural question at this point is why we are not experimenting with word-level transduction tasks, such as machine translation. As we show in the §3.2 our method is often an order of magnitude slower, since it will involve a mixture of softmaxes. Thus, the exact marginalization scheme is practically unworkable for machine translation; we discuss future extensions for machine translation in §6. However, the slow-down is no problem for character-level tasks and we show empirical gains in §8.

## 3 Hard Non-Monotonic Alignment

### 3.1 The Latent-Variable Model

An alphabet is a finite, non-empty set. Given two alphabets $\Sigma_{\mathrm{x}} = \{x_1, \ldots, x_{|\Sigma_{\mathrm{x}}|}\}$ and $\Sigma_{\mathrm{y}} = \{y_1, \ldots, y_{|\Sigma_{\mathrm{y}}|}\}$, probabilistic approaches to the problem attempt to estimate a probability distribution $p(\mathbf{y} \mid \mathbf{x})$ where $\mathbf{y} \in \Sigma_{\mathrm{y}}^*$ and $\mathbf{x} \in \Sigma_{\mathrm{x}}^*$. Foreshadowing, we will define the parameters of $p$ to be, in part, the parameters of a recurrent neural network, in line with the state-of-the-art models. We define the set $A = \{1, \ldots, |\mathbf{x}|\}^{|\mathbf{y}|}$, which has an interpretation as the set of all (potentially non-monotonic) alignments from $\mathbf{x}$ to $\mathbf{y}$ with the restriction that each output symbol $y_i$ aligns to exactly one symbol in $\mathbf{x} \in \Sigma_{\mathrm{x}}^*$. In other words, $A$ is the set of all many-to-one alignments between $\mathbf{x}$ and $\mathbf{y}$ where many may be as few as zero. We remark that $|A| = |\mathbf{x}|^{|\mathbf{y}|}$, which is exponentially large in the length of the target string $\mathbf{y}$. For an $\mathbf{a} \in A$, $a_i = j$ refers to the event that $y_i$, the $i^{\text{th}}$ component of $\mathbf{y}$, is aligned to $x_j$, the $j^{\text{th}}$ component of $\mathbf{x}$.

We define a probability distribution over output strings $\mathbf{y}$ conditioned on an input string $\mathbf{x}$ where we marginalize out *unobserved* alignments $\mathbf{a}$:

$$p(\mathbf{y} \mid \mathbf{x}) = \sum_{\mathbf{a} \in A(\mathbf{x},\mathbf{y})} p(\mathbf{y}, \mathbf{a} \mid \mathbf{x}) \tag{1}$$

$$= \underbrace{\sum_{\mathbf{a} \in A} \prod_{i=1}^{|\mathbf{y}|} p(y_i \mid a_i, \mathbf{y}_{<i}, \mathbf{x})\, p(a_i \mid \mathbf{y}_{<i}, \mathbf{x})}_{\textit{exponential number of terms}} \tag{2}$$

$$= \underbrace{\prod_{i=1}^{|\mathbf{y}|} \sum_{a_i=1}^{|\mathbf{x}|} p(y_i \mid a_i, \mathbf{y}_{<i}, \mathbf{x})\, p(a_i \mid \mathbf{y}_{<i}, \mathbf{x})}_{\textit{polynomial number of terms}} \tag{3}$$

$$= \prod_{i=1}^{|\mathbf{y}|} \sum_{j=1}^{|\mathbf{x}|} \mathbb{1}_{a_{i,j}}\, \alpha_j(i)\, p(y_i \mid a_i, \mathbf{y}_{<i}, \mathbf{x}) \tag{4}$$

where we define $\alpha_j(i) = p(a_i \mid \mathbf{y}_{<i}, \mathbf{x})$ in order to better notationally compare our model to that

of Bahdanau et al. (2015) in §5. Each distribution $p(y_i \mid a_i, \mathbf{y}_{<i}, \mathbf{x})$ in the definition of the model has a clean interpretation as a distribution over the output vocabulary $\Sigma_{\mathrm{y}}$, given an input string $\mathbf{x} \in \Sigma_{\mathrm{x}}^*$, where $y_i$ is aligned to $x_j$. Thus, one way of thinking about this hard alignment model is as a product of mixture models, one mixture at each step, with mixing coefficients $\alpha_j(i)$.

**Why Does Dynamic Programming Work?** Our dynamic program to compute the likelihood, fully specified in eq. (3), is quite simple: The non-monotonic alignments are independent of each other, i.e., $\alpha_j(i)$ is independent of $\alpha_j(i-1)$, conditioned on the observed sequence $\mathbf{y}$. This means that we can cleverly rearrange the terms in eq. (2) using the distributive property. Were this not the case, we could not do better than having an exponential number of summands. This is immediately clear when we view our model as a graphical model, as in Fig. 2: There is no active trail from $a_i$ to $a_k$ where $k > i$, ignoring the dashed lines. Note that this is no different than the tricks used to achieve exact inference in $n^{\text{th}}$-order Markov models—one makes an independence assumption between the current bit of structure and the previous bits of structure to allow an efficient algorithm. For a proof of eq. (2)–eq. (3), one may look in Brown et al. (1993). Foreshadowing, we note that certain parameterizations make use of input feeding, which breaks this independence; see §5.1.

**Relation to IBM Model 1.** The derivation above is similar to that of the IBM alignment model 1. We remark, however, two key generalizations that will serve our recurrent neural parameterization well in §4. First, traditionally, derivations of IBM Model 1 omit a prior over alignments $p(a_i \mid \mathbf{x})$, taking it to be uniform. Due to this omission, an additional multiplicative constant $\varepsilon/|\mathbf{x}|^{|\mathbf{y}|}$ is introduced to ensure the distribution remains normalized (Koehn, 2009). Second, IBM Model 1 does not condition on previously generated words on the output side. In other words, in their original model, Brown et al. (1993) assume that $p(y_i \mid a_i, \mathbf{y}_{<i}, \mathbf{x}) = p(y_i \mid a_i, \mathbf{x})$, forsaking dependence on $\mathbf{y}_{<i}$. We note that there is no reason why we need to make this independence assumption—we will likely want a target-side language model in transduction. Indeed, subsequent statistical machine translation systems, e.g., MOSES (Koehn et al., 2007), integrate a language model into the decoder. It is of note that many models in NLP have made similar independence assumptions, e.g., the emission distribution hidden Markov models (HMMs) are typically taken to be independent of all previous emissions (Rabiner, 1989). These assumptions are generally not necessary.

### 3.2 Algorithmic Analysis: Time Complexity

Let us assume that the requisite probability distributions are computable in $\mathcal{O}(1)$ time and the softmax takes $\mathcal{O}(\Sigma_{\mathrm{y}})$. Then, by inspection, the computation of the distribution in eq. (4) is $\mathcal{O}\left(|\mathbf{x}| \cdot |\mathbf{y}| \cdot |\Sigma_{\mathrm{y}}|\right)$, as the sum in eq. (3) contains this many terms thanks to the dynamic program that allowed us to rearrange the sum and the product. While this "trick" is well known in the NLP literature—it dates from the seminal work in statistical machine translation by Brown et al. (1993)—it has been forgotten in recent formulations of hard alignment (Xu et al., 2015), which use stochastic approximation to handle the exponential summands. As we will see in §5, we can compute the soft-attention model of Bahdanau et al. (2015) in $\mathcal{O}\left(|\mathbf{x}| \cdot |\mathbf{y}| + |\mathbf{y}| \cdot |\Sigma_{\mathrm{y}}|\right)$ time. When $\Sigma_{\mathrm{y}}$ is large, for example in case of machine translation with tens of thousands of $\Sigma_{\mathrm{y}}$ at least, we can ignore $|\mathbf{x}| \cdot |\mathbf{y}|$ in soft-attention model, and the exact marginalization has an extra $|\mathbf{x}|$-factor compared to soft-attention model. In practice, Shi and Knight (2017) show the bottleneck of a NMT system is the softmax layer, making the extra $|\mathbf{x}|$-factor practically cumbersome.

## 4 Recurrent Neural Parameterization

How do we parameterize $p(y_i \mid a_i, \mathbf{y}_{<i}, \mathbf{x})$ and $\alpha_j(i)$ in our hard, non-monotonic transduction model? We will use a neural network identical to the one proposed in the attention-based sequence-to-sequence model of Luong et al. (2015) *without input feeding* (a variant of Bahdanau et al. (2015)).

### 4.1 Encoding the Input

All models discussed in this exposition will make use of the same mechanism for mapping a source string $\mathbf{x} \in \Sigma_{\mathrm{x}}^*$ into a fixed-length representation in $\mathbb{R}^{d_h}$. This mapping will take the form of a bidirectional recurrent neural network encoder, which works as follows: each element of $\Sigma_{\mathrm{x}}$ is mapped to an embedding vector of length $d_e$ through a mapping: $\mathbf{e} : \Sigma_{\mathrm{x}} \to \mathbb{R}^{d_e}$. Now, the RNN folds the

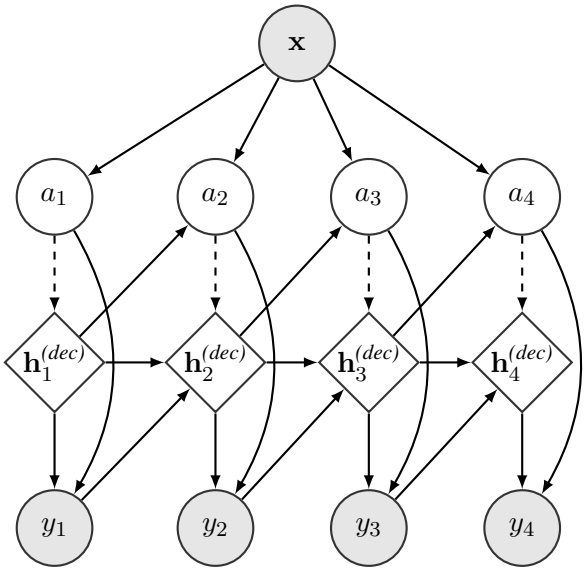


Figure 2: Our hard-attention model *without input feeding* viewed as a graphical model. Note that the circular nodes are random variables and the diamond nodes deterministic variables ($\mathbf{h}_i^{(dec)}$ is first discussed in §4.3). The independence assumption between the alignments $a_i$ when the $y_i$ are *observed* becomes clear. Note that we have omitted arcs from $\mathbf{x}$ to $y_1$, $y_2$, $y_3$ and $y_4$ for clarity (to avoid crossing arcs). We alert the reader that the dashed edges show the additional dependencies added in the *input feeding version*, as discussed in §5.1. Once we add these in, the $a_i$ are no longer independent and break exact marginalization. Note the hard-attention model does not enforce an exactly one-to-one constraint. Each source-side word is free to align to many of the target-side words, independent of context. In the latent variable model, the $x$ variable is a vector of source words, and the alignment may be over more than one element of $x$.

following recursion over the string $\mathbf{x}$ left-to-right:

$$\overrightarrow{\mathbf{h}}_j^{(enc)} = \tanh\Big(\overrightarrow{\mathbf{U}}^{(enc)}\,\mathbf{e}^{(enc)}(x_j) + \overrightarrow{\mathbf{V}}^{(enc)}\,\overrightarrow{\mathbf{h}}_{j-1}^{(enc)} + \overrightarrow{\mathbf{b}}^{(enc)}\Big) \tag{5}$$

where we fix the $0^{\text{th}}$ hidden state $\mathbf{h}_0^{(enc)}$ to the zero vector and the matrices $\overrightarrow{\mathbf{U}}^{(enc)} \in \mathbb{R}^{d_h \times d_e}$, $\overrightarrow{\mathbf{V}}^{(enc)} \in \mathbb{R}^{d_h \times d_h}$ and the bias term $\overrightarrow{\mathbf{b}}^{(enc)} \in \mathbb{R}^{d_h}$ are parameters to be learned. Performing the same procedure on the reversed string and using an RNN with different parameters, we arrive at hidden state vectors $\overleftarrow{\mathbf{h}}_j^{(enc)}$. The final hidden states from the encoder are the concatenation of the two, i.e., $\mathbf{h}_j^{(enc)} = \overrightarrow{\mathbf{h}}_j^{(enc)} \oplus \overleftarrow{\mathbf{h}}_j^{(enc)}$, where $\oplus$ is vector concatenation.

As has become standard, we will use an extension to this recursion: we apply the long short-term memory (LSTM; (Hochreiter and Schmidhuber, 1997)) recursions, rather than those of a vanilla RNN (Elman network; Elman (1990)).

### 4.2 Parameterization.

Now we define the alignment distribution $\alpha_j(i)$.

$$\alpha_j(i) = \frac{\exp(e_{ij})}{\sum_{j'=1}^{|\mathbf{x}|} \exp(e_{ij'})} \tag{6}$$

$$e_{ij} = \mathbf{h}_i^{(dec)\top}\,\mathbf{T}\,\mathbf{h}_j^{(enc)} \tag{7}$$

where $\mathbf{T} \in \mathbb{R}^{d_h \times 2d_h}$ and $\mathbf{h}_i^{(dec)}$, the decoder RNN's hidden state, is defined in §4.3. Importantly, the alignment distribution $\alpha_j(i)$ at time step $i$ will *only* depend on the prefix of the output string $\mathbf{y}_{<i}$ generated so far. This is clear since the output-side decoder is a unidirectional RNN.

We also define $p(y_i \mid a_i, \mathbf{y}_{<i}, \mathbf{x})$.

$$p(y_i \mid a_i, \mathbf{y}_{<i}, \mathbf{x}) = \text{softmax}\Big(\mathbf{W}\,\mathbf{f}(\mathbf{h}_i^{(dec)}, \mathbf{h}_{a_i}^{(enc)})\Big) \tag{8}$$

The function $\mathbf{f}$ is a non-linear and vector-valued; one popular choice of $\mathbf{f}$ is a multilayer perceptron with parameters to be learned. We define:

$$\mathbf{f}(\mathbf{h}_i^{(dec)}, \mathbf{h}_{a_i}^{(enc)}) = \tanh\Big(\mathbf{S}\,(\mathbf{h}_i^{(dec)} \oplus \mathbf{h}_{a_i}^{(enc)})\Big) \tag{9}$$

where $\mathbf{S} \in \mathbb{R}^{d_s \times 3d_h}$.

### 4.3 Updating the hidden state $\mathbf{h}_i^{(dec)}$

The hidden state $\mathbf{h}_i^{(dec)}$ is also updated through the LSTM recurrences (Hochreiter and Schmidhuber, 1997). The RNN version of the recurrence mirrors that of the encoder,

$$\mathbf{h}_i^{(dec)} = \tanh\Big(\mathbf{U}^{(dec)}\,\mathbf{e}^{(dec)}(y_{i-1}) + \mathbf{V}^{(dec)}\,\mathbf{h}_{i-1}^{(dec)} + \mathbf{b}^{(dec)}\Big) \tag{10}$$

where $\mathbf{e}^{(dec)} : \Sigma_{\text{Y}} \to \mathbb{R}^{d_e}$ produces an embedding of each of the symbols in the output alphabet. What

is crucial about this RNN, like the $\alpha_j(i)$, is that it only summarizes the characters decoded so far *independently of the previous attention weights*. In other words, the attention weights at time step $i$ will have no influence from the attention weights at previous time steps, shown in Fig. 2. This is what allows for dynamic programming.

## 5 Transduction with Soft Attention

In order to contrast it with the hard alignment mechanism we develop, we here introduce Luong attention (Luong et al., 2015) for recurrent neural sequence to sequence models (Sutskever et al., 2014). Note that this model will also serve as an experimental baseline in §8.

The soft-attention transduction model defines a distribution over the output $\Sigma_{\mathbf{y}}^*$, much like the hard-attention model, with the following expression:

$$p(\mathbf{y} \mid \mathbf{x}) = \prod_{i=1}^{|\mathbf{y}|} p(y_i \mid \mathbf{y}_{<i}, \mathbf{x}) \tag{11}$$

where we define each conditional distribution as

$$p(y_i \mid \mathbf{y}_{<i}, \mathbf{x}) \tag{12}$$
$$= \text{softmax}\left(\mathbf{W}\, \mathbf{f}(\mathbf{h}_i^{(dec)}, \mathbf{c}_i)\right)$$

We reuse the function $\mathbf{f}$ in eq. (9). The hidden state $\mathbf{h}_i^{(dec)}$, as before, is the $i^{\text{th}}$ state of a target-side language model that summarizes the prefix of the string decoded so far; this is explained in §4.3. And, finally, we define the **context vector**

$$\mathbf{c}_i = \sum_{j=1}^{|\mathbf{x}|} \alpha_j(i)\, \mathbf{h}_j^{(enc)} \tag{13}$$

using the same alignment distribution as in §4.2. In the context of the soft-attention model, this distribution is referred to as the **attention weights**.

Inspection shows that there is only a small difference between the soft-attention model presented here and and our hard non-monotonic attention model. The difference is where we place the probabilities $\alpha_j(i)$. In the soft-attention version, we place them inside the softmax (and the function $\mathbf{f}$), as in eq. (12), and we have a mixture of the encoder's hidden states, the context vector, that we feed into the model. On the other hand, if we place them outside the softmax, we have a mixture of softmaxes, as shown in eq. (3). Both models have *identical* set of parameters.

### 5.1 Input Feeding: What's That?

The equations in eq. (10), however, are not the only approach. **Input-feeding** is another popular approach that is, perhaps, standard at this point (Luong et al., 2015). Input feeding refers to the setting where the architecture designer additionally feeds the attention weights into the update for the decoder's hidden state. This yields the recursion

$$\mathbf{h}_i^{(dec)} = \tanh\Big(\mathbf{U}^{(dec)}\,(\mathbf{e}^{(dec)}(y_{i-1}) \oplus \bar{\mathbf{c}}_{i-1}) + \mathbf{V}^{(dec)}\,\mathbf{h}_{i-1}^{(dec)} + \mathbf{b}^{(dec)}\Big) \tag{14}$$

where $\bar{\mathbf{c}}_{i-1} = \mathbf{f}(\mathbf{h}_{i-1}^{(dec)}, \mathbf{c}_{i-1})$. Note that this requires that $\mathbf{U}^{(dec)} \in \mathbb{R}^{d_h \times (d_e + d_s)}$. This is the architecture discussed in Bahdanau et al. (2015, §3.1). In contrast to the architecture above, this architecture has attention weights that do depend on previous attention weights due to the feeding in of the context vector $\mathbf{c}_i$. See Cohn et al. (2016) for an attempt to incorporate structural biases into the manner in which the attention distribution is influenced by previous attention distributions.

### 5.2 Combining Hard Non-Monotonic Attention with Input Feeding

To combine hard attention with input feeding, Xu et al. (2015) derive a variational lower bound on the log-likelihood though Jensen's inequality:

$$\begin{aligned}
&\log p(\mathbf{y} \mid \mathbf{x}) = \log \sum_{\mathbf{a} \in A(\mathbf{x},\mathbf{y})} p(\mathbf{y}, \mathbf{a} \mid \mathbf{x}) \\
&= \log \sum_{\mathbf{a} \in A(\mathbf{x},\mathbf{y})} p(\mathbf{a} \mid \mathbf{x}) \cdot p(\mathbf{y} \mid \mathbf{x}, ) \\
&\geq \sum_{\mathbf{a} \in A(\mathbf{x},\mathbf{y})} p(\mathbf{a} \mid \mathbf{x}) \log p(\mathbf{y} \mid \mathbf{x}, \mathbf{a})
\end{aligned}$$

Note that we have omitted the dependence of $p(\mathbf{a} \mid \mathbf{x})$ on the appropriate prefix of $\mathbf{y}$; this was done for notational simplicity. Using this bound, Xu et al. (2015) derive an efficient approximation to the gradient using the REINFORCE trick of Williams (1992). This sampling-based gradient estimator is then used for learning, but suffers from high variance. We compare to this model in §8.

## 6 Future Work

Just as Brown et al. (1993) started with IBM Model 1 and build up to richer models, we can do the same. Extensions, resembling those of IBM Model 2 and the HMM aligner (Vogel et al., 1996)

| | source | target |
|---|---|---|
| **Grapheme-to-phoneme conversion** | a c t i o n | AE K SH AH N |
| **Named-entity transliteration** | A A C H E N | 아 헨 |
| **Morphological inflection** | N AT+ALL SG l i p u k e | l i p u k k e e l l e |

Table 1: Example of source and target string for each task as processed by the model

that generalize IBM Model 1, are easily bolted on to our proposed model as well. If we are willing to perform approximate inference, we may also consider fertility as found in IBM Model 4.

In order to extend our method to machine translation (MT) in any practical manner, we require an approximation to the softmax. Given that the softmax is already the bottleneck of neural MT models (Shi and Knight, 2017), we can not afford ourselves a $\mathcal{O}(|\mathbf{x}|)$ slowdown during training. Many methods have been proposed for approximating the softmax (Goodman, 2001; Bengio et al., 2003; Gutmann and Hyvärinen, 2010). More recently, Chen et al. (2016) compared methods on neural language modeling and Grave et al. (2017) proposed a GPU-friendly method.

## 7 The Tasks

The empirical portion of the paper focuses on character-level string-to-string transduction problems. We consider three tasks: G: grapheme-to-phoneme conversion, T: named-entity transliteration, and I: morphological inflection. We describe each briefly in turn and we give an example of a source and target string for each task in Tab. 1.

**Grapheme-to-Phoneme Conversion.** We use the standard grapheme-to-phoneme conversion (G2P) dataset: the Sphinx-compatible version of CMUDict (Weide, 1998) and NetTalk (Sejnowski and Rosenberg, 1987). G2P transduces a word, a string of graphemes, to its pronunciation, a string of phonemes. We evaluate with word error rate (WER) and phoneme error rate (PER) (Yao and Zweig, 2015). PER is equal to the edit distance divided by the length of the string of phonemes.

**Named-Entity Transliteration.** We use the NEWS 2015 shared task on machine transliteration (Zhang et al., 2015) as our named-entity transliteration dataset. It contains 14 language pairs. Transliteration transduces a named entity from its source language to a target language—in other words, from a string in the source orthography to a string in the target orthography. We evaluate with word accuracy in percentage (ACC) and mean F-score

| input-fed | soft attention | hard alignment |
|---|---|---|
| yes | ① Bahdanau et al. (2015); Luong et al. (2015) | ② Xu et al. (2015) |
| no | ③ Luong et al. (2015) *without input feeding* | ④ *This work* |

Table 2: The 4 architectures considered in the paper.

(MFS) (Zhang et al., 2015). For completeness, we include the definition of MFS in App. A.

**Morphological Inflection.** We consider the high-resource setting of task 1 in the CoNLL–SIGMORPHON 2017 shared task (Cotterell et al., 2017) as our morphological inflection dataset. It contains 51 languages in the high resource setting. Morphological inflection transduces a lemma (a string of characters) and a morphological tag (a sequence of subtags) to an inflected form of the word (a string of characters). We evaluate with word accuracy (ACC) and average edit distance (MLD) (Cotterell et al., 2017).

## 8 Experiments

The goal of the empirical portion of our paper is to perform a controlled study of the different architectures and approximations discussed up to this point in the paper. §8.1 exhibits the neural architectures we compare and the main experimental results[1] are in Tab. 3. In §8.2, we present the experimental minutiae, e.g. hyperparameters. In §8.3, we analyze our experimental findings. Finally, in §8.4, we perform error analysis and visualize the soft attention weight and hard alignment distribution.

### 8.1 The Architectures

The four architectures we consider in controlled comparison are: ①: soft attention *with* input feeding, ②: hard attention *with* input feeding, ③: soft attention *without* input feeding and ④: **hard attention *without* input feeding** (our system). They are also shown in Tab. 2. As a fifth system, we compare to the monotonic system Ⓜ: Aharoni and Goldberg (2017). Additionally, we present Ⓤ, a variant of ① where the number of parameters is not

[1] Because we do not have access to the test set of T, we only report development performance.

| | Grapheme-to-Phoneme Conversion (G) | | | | Named-Entity Transliteration (T) | | | | Morphological Inflection (I) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Small** | | **Large** | | **Small** | | **Large** | | **Small** | | **Large** | |
| | **WER** | **PER** | **WER** | **PER** | **ACC** | **MFS** | **ACC** | **MFS** | **ACC** | **MLD** | **ACC** | **MLD** |
| (1) | 33.7 | 0.080 | 30.8 | 0.074 | 38.9 | 0.890 | 39.9 | 0.893 | 91.4 | 0.183 | 91.1 | 0.201 |
| (U) | 30.6 | 0.074 | 30.4 | 0.073 | 39.8 | 0.891 | 40.3 | 0.894 | 91.0 | 0.185 | 91.0 | 0.212 |
| (2) | 32.3 | 0.079 | 33.1 | 0.081 | 36.3 | 0.881 | 30.8 | 0.837 | 91.0 | 0.193 | 89.3 | 0.322 |
| (3) | 30.3 | 0.074 | 28.6 | 0.070 | **40.1** | 0.891 | 40.5 | 0.894 | 92.0 | 0.163 | 92.2 | 0.166 |
| (4) | **29.6** | **0.072** | **28.2** | **0.068** | 39.8 | 0.891 | **41.1** | 0.894 | **92.6** | **0.151** | **93.6** | **0.128** |
| (R) | 30.7 | 0.076 | 29.7 | 0.074 | 37.1 | 0.882 | 36.9 | 0.863 | 91.2 | 0.190 | 92.8 | 0.151 |
| (M) | 33.9 | 0.082 | 29.9 | 0.072 | 38.8 | **0.959** | 40.1 | **0.960** | 91.7 | 0.160 | 92.8 | 0.141 |

Table 3: Average test performance on G, T and I averaged across datasets and languages. See App. B fCor full breakdown.

| | **Small** | **Large** | **Search range** |
|---|---|---|---|
| **Emb. dim.** | 100 | 200 | {50,100,200,300} |
| **Enc. dim.** | 200 | 400 | {100,200,400,600} |
| **Enc. layer** | 1 | 2 | {1,2,3} |
| **Dec. dim.** | 200 | 400 | {100,200,400,600} |
| **Dec. layer** | 1 | 1 | {1,2,3} |
| **Dropout** | 0.2 | 0.4 | {0,0.1,0.2,0.4} |
| **# param.** | 1.199M | 8.621M | N/A |

Table 4: Model hyperparameters and search range

controlled for, and (R), a variant of (4) trained using REINFORCE instead of exact marginalization.

### 8.2 Experimental Details

We implement the experiments with PyTorch (Paszke et al., 2017) and we port the code of Aharoni and Goldberg (2017) to admit batched training. Because we did not observe any improvements in preliminary experiments when decoding with beam search [2], all models are decoded greedily.

**Data Preparation.** For G, we sample 5% and 10% of the data as development set and test set, respectively. For T, we only run experiments with 11 out of 14 language pairs[3] because we do not have access to all the data.

**Model Hyperparameters.** The hyperparameters of all models are in Tab. 4. The hyperparameters of the large model are tuned using the baseline (3) on selected languages in I, and the search range is shown in Tab. 4. All three tasks have the same two sets of hyperparameters. To ensure that (1) has the same number of parameters as the other models, we decrease $d_s$ in eq. (9) while for the rest of the models $d_s = 3d_h$. Additionally, we use a linear mapping to merge $\mathbf{e}^{(dec)}(y_{i-1})$ and $\bar{\mathbf{c}}_{i-1}$ in eq. (14) instead of concatenation. The output of the linear mapping has the same dimension as $\mathbf{e}^{(dec)}(y_{i-1})$, ensuring that the RNN has the same size.

(M) has quite a different architecture: The input of the decoder RNN is the concatenation of the previously predicted word embedding, the encoder's hidden state at a specific step, and in the case of I, the encoding of the morphological tag. Differing from Aharoni and Goldberg (2017), we concatenate all attributes' embeddings (0 for attributes that are not applicable) and merge them with a linear mapping. The dimension of the merged vector and attributes vector are $d_e$. To ensure that it has the same number of parameters as the rest of the model, we increase the hidden size of the decoder RNN.

**Optimization.** We train the model with Adam (Kingma and Ba, 2014) with an initial learning rate of 0.001. We halve the learning rate whenever the development log-likelihood doesn't improve. We stop after the learning rate dips to $1 \times 10^{-5}$. We save all models after each epoch and select the model with best development performance. We train the model for at most 50 epochs, though all the experiments stop early. We train on G, T, and I with batch sizes of 20, 50 and 20, respectively. We notice in the experiments that the training of (1) and (U) is quite unstable with the large model, probably because of the longer chain of gradient information flow. We apply gradient clipping to the large model with maximum gradient norm 5.

**REINFORCE.** In the REINFORCE training of (R) and (2), we sample 2 and 4 positions at each time step for the small and large model, respectively. The latter is tuned on selected languages in I with search range {2,3,4,5}. To stabilize the training, we apply a baseline with a moving average reward and

[2] Compared to greedy decoding with an average error rate of 20.1% and an average edit distance of 0.385, beam search with beam size 5 gets a slightly better edit distance of 0.381 while hurting the error rate with 20.2%.

[3] Ar–En, En–Ba, En–Hi, En–Ja, En–Ka, En–Ko, En–Pe, En–Ta, En–Th, Jn–Jk and Th–En.

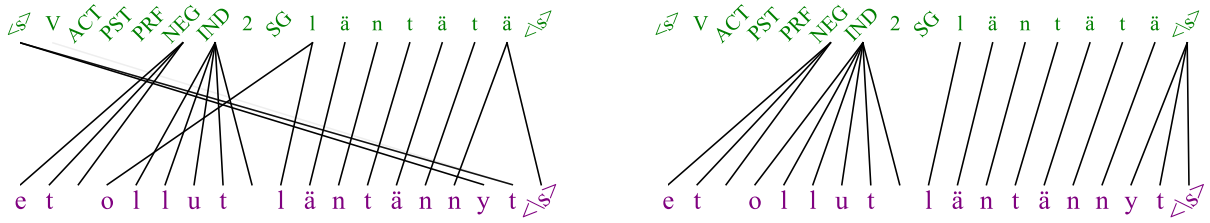


Figure 3: Attention-weight (③; left) and alignment distribution (④; right) of Finnish in I. Both models predict correctly.

discount factor of 0.9, similar to Xu et al. (2015).

### 8.3 Experimental Findings

**Finding #1: Effect of Input Feeding.** By comparing ③ and ④ against ① and ② in Tab. 3, we find input feeding hurts performance in all settings and all tasks. This runs in contrast to the reported results of Luong et al. (2015), but they experiment on machine translation, rather than character-level transduction. This validates our independence assumption about the alignment distribution.

**Finding #2: Soft Attention vs. Hard Attention.** Training with REINFORCE hurts the performance of the hard attention model; compare ① and ② (trained with REINFORCE), in Tab. 3. On the other hand, training with exact marginalization causes the hard attention model to outperform soft attention model in nearly all settings; compare ③ and ④ in Tab. 3. This comparison shows that hard attention outperforms soft attention in character-level string transduction when trained with exact marginalization.

**Finding #3: Non-monotonicity vs. Monotonicity.** The monotonic model Ⓜ underperforms compared to non-monotonic models ③ in Tab. 3 except for one setting. It performs slightly worse on T and G due to the many-to-one alignments in the data and the fact that Aharoni and Goldberg (2017) can only use the hidden vector of the final element of the span in a many-to-one alignment to directly predict the one target element. The current state-of-the-art systems for character-level string transduction are non-monotonic models, despite the tasks' seeming monotonicity; see §8.4.

**Finding #4: Approximate Hard Attention.** Given our development of an exact marginalization method for neural models with hard attention, a natural question to ask is how much exact marginalization helps during learning. By comparing ④ and Ⓡ in Tab. 3, we observe that training with exact marginalization clearly outperforms training under stochastic approximation in every setting and on every dataset. We also observe that exact marginalization allows faster convergence, since training with REINFORCE is quite unstable where some runs seemingly to get stuck.

| | NETtalk | | | | CMUDict | | | |
|---|---|---|---|---|---|---|---|---|
| | ③ | | ④ | | ③ | | ④ | |
| | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ |
| **Monotonic** | 18742 | 1230 | 18823 | 1172 | 95824 | 17294 | 96176 | 17159 |
| **Non-monotonic** | 31 | 5 | 12 | 1 | 158 | 162 | 37 | 66 |

Table 5: Breakdown of correct and incorrect predictions of monotonic and non-monotonic alignments of ③ and ④ in G, derived from the soft attention weights and the hard alignment distribution

**Finding #5: Controlling for Parameters.** Input feeding yields a more expressive model, but also leads to an increase in the number of parameters. Here, we explore what effect this has on the performance of the models. In their ablation, Luong et al. (2015) did not control the number of parameters when adding input feeding. The total number of parameters of Ⓤ is 1.679M for the small setting and 10.541M for the large setting, which has 40% and 22.3% more parameters than the controlled setting. By comparing ① and Ⓤ in Tab. 3, we find that the increase in parameters, rather than the increase in expressivity explains the success of input feeding.

### 8.4 Visualization and Error Analysis

We hypothesize that even though the model is non-monotonic, it can learn monotonic alignment with flexibility if necessary, giving state-of-the-art results on many seemingly monotonic character-level string transduction task. To show more insights, we compare the best soft attention model (③) against the best hard alignment model (④) on G by showing the confusion matrix of each model in Tab. 5. An alignment is non-monotonic when alignment edges predicted by the model cross. There is an edge connecting $x_j$ and $y_i$ if the attention weight or hard alignment distribution $\alpha_j(i)$ is

larger than 0.1. We find that the better-performing transducers are more monotonic, and most learned alignments are monotonic. The results indicate that there are a few transductions that are indeed non-monotonic in the dataset. However, the number is so few that this does not entirely explain why non-monotonic models outperform the monotonic models. We speculate this lies in the architecture of Aharoni and Goldberg (2017), which does not permit many-to-one alignments, while monotonic alignment learned by the non-monotonic model is more flexible. Future work will investigate this.

In Fig. 3, we visualize the soft attention weights (③) and the hard alignment distribution (④) side by side. We observe that the hard alignment distribution is more interpretable, with a clear boundary when predicting the prefixes.

## 9 Conclusion

We exhibit an efficient dynamic program for the exact marginalization of all non-monotonic alignments in a neural sequence-to-sequence model. We show empirically that exact marginalization helps over approximate inference by REINFORCE and that models with hard, non-monotonic alignment outperform those with soft attention.

**Acknowledgements.** The last author was supported by an NDSEG fellowship and a Facebook Fellowship.

## References

Roee Aharoni and Yoav Goldberg. 2017. Morphological inflection generation with hard monotonic attention. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2004–2015, Vancouver, Canada. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, volume abs/1409.0473.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Wenlin Chen, David Grangier, and Michael Auli. 2016. Strategies for training large vocabulary neural language models. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1975–1985.

Trevor Cohn, Cong Duy Vu Hoang, Ekaterina Vymolova, Kaisheng Yao, Chris Dyer, and Gholamreza Haffari. 2016. Incorporating structural alignment biases into an attentional neural translation model. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 876–885, San Diego, California. Association for Computational Linguistics.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, Géraldine Walther, Ekaterina Vylomova, Patrick Xia, Manaal Faruqui, Sandra Kübler, David Yarowsky, Jason Eisner, and Mans Hulden. 2017. The CoNLL-SIGMORPHON 2017 shared task: Universal morphological reinflection in 52 languages. In *Proceedings of the CoNLL-SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, Vancouver, Canada. Association for Computational Linguistics.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science*, 14(2):179–211.

Joshua Goodman. 2001. Classes for fast maximum entropy training. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, volume 1, pages 561–564. IEEE.

Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2017. Efficient softmax approximation for gpus. In *ICML*.

Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. *Neural Computation*, 9(8):1735–1780.

Katharina Kann and Hinrich Schütze. 2016. Single-model encoder-decoder with explicit morphological representation for reinflection. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 555–560, Berlin, Germany. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Philipp Koehn. 2009. *Statistical Machine Translation*. Cambridge University Press.

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 *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *EMNLP*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Lawrence R. Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. 2016. Weighting finite-state transductions with neural context. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 623–633, San Diego, California. Association for Computational Linguistics.

Kenneth L. Rehg and Damian G. Sohl. 1981. *Ponapean Reference Grammar*. University of Hawaii Press.

Mihaela Rosca and Thomas Breuel. 2016. Sequence-to-sequence neural network models for transliteration. *arXiv preprint arXiv:1610.09565*.

Terrence J. Sejnowski and Charles R. Rosenberg. 1987. Parallel networks that learn to pronounce english text. *Complex Systems*, 1.

Xing Shi and Kevin Knight. 2017. Speeding up neural machine translation decoding by shrinking run-time vocabulary. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 574–579.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27 (NIPS)*, pages 3104–3112.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. 1996. HMM-based word alignment in statistical translation. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 2*, COLING '96, pages 836–841, Stroudsburg, PA, USA.

R.L. Weide. 1998. The carnegie mellon pronouncing dictionary.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, pages 2048–2057.

Kaisheng Yao and Geoffrey Zweig. 2015. Sequence-to-sequence neural net models for grapheme-to-phoneme conversion. In *INTERSPEECH 2015*, pages 3330–3334, Dresden, Germany.

Min Zhang, Haizhou Li, Rafael E. Banchs, and A. Kumaran. 2015. Whitepaper of news 2015 shared task on machine transliteration. In *NEWS@ACL*.

## A MFS

$$
\begin{aligned}
LCS(c, r) &= \frac{1}{2}(|c| + |r| - ED(c, r)) \\
R_i &= \frac{LCS(c_i, r_i)}{|r_i|} \\
P_i &= \frac{LCS(c_i, r_i)}{|c_i|} \\
FS_i &= 2\frac{R_i \times P_i}{R_i + P_i}
\end{aligned}
$$

where $r_i$ and $c_i$ is the i-th reference and prediction,

## B Full breakdown of experiments

A full breakdown of G and T can be found in Tab. 6 and Tab. 7, respectively. A full breakdown of I can be found in Tab. 8 and Tab. 9.

| | Small | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ① | | Ⓤ | | ② | | ③ | | ④ | | Ⓡ | | Ⓜ | |
| | **WER** | **PER** | **WER** | **PER** | **WER** | **PER** | **WER** | **PER** | **WER** | **PER** | **WER** | **PER** | **WER** | **PER** |
| CMUDict | 36.2 | 0.086 | 31.0 | 0.074 | 35.1 | 0.083 | 30.8 | 0.073 | 30.5 | 0.072 | 31.2 | 0.074 | 32.0 | 0.075 |
| NETtalk | 31.2 | 0.075 | 30.2 | 0.075 | 29.6 | 0.074 | 29.8 | 0.074 | 28.8 | 0.073 | 30.3 | 0.078 | 35.7 | 0.088 |
| | **Large** | | | | | | | | | | | | | |
| | ① | | Ⓤ | | ② | | ③ | | ④ | | Ⓡ | | Ⓜ | |
| | **WER** | **PER** | **WER** | **PER** | **WER** | **PER** | **WER** | **PER** | **WER** | **PER** | **WER** | **PER** | **WER** | **PER** |
| CMUDict | 32.3 | 0.076 | 31.4 | 0.073 | 36.7 | 0.087 | 30.5 | 0.073 | 29.8 | 0.071 | 31.8 | 0.077 | 30.5 | 0.072 |
| NETtalk | 29.3 | 0.071 | 29.4 | 0.072 | 29.5 | 0.075 | 26.8 | 0.068 | 26.6 | 0.066 | 27.7 | 0.071 | 29.3 | 0.072 |

Table 6: Full breakdown of G2P

| | Small | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ① | | Ⓤ | | ② | | ③ | | ④ | | Ⓡ | | Ⓜ | |
| | **ACC** | **MFS** | **ACC** | **MFS** | **ACC** | **MFS** | **ACC** | **MFS** | **ACC** | **MFS** | **ACC** | **MFS** | **ACC** | **MFS** |
| ArEn | 54.9 | 0.954 | 54.8 | 0.953 | 53.9 | 0.954 | 53.5 | 0.951 | 56.6 | 0.954 | 53.9 | 0.950 | 60.1 | 0.980 |
| EnBa | 38.4 | 0.916 | 38.9 | 0.915 | 38.5 | 0.914 | 37.6 | 0.918 | 39.7 | 0.918 | 38.0 | 0.909 | 37.4 | 0.961 |
| EnHi | 42.4 | 0.922 | 44.0 | 0.925 | 40.8 | 0.921 | 46.1 | 0.927 | 43.8 | 0.926 | 43.6 | 0.924 | 43.0 | 0.967 |
| EnJa | 40.5 | 0.871 | 40.6 | 0.868 | 35.4 | 0.853 | 41.2 | 0.872 | 41.3 | 0.872 | 35.2 | 0.852 | 39.4 | 0.952 |
| EnKa | 34.8 | 0.910 | 35.5 | 0.912 | 33.1 | 0.909 | 37.8 | 0.913 | 36.0 | 0.909 | 34.9 | 0.907 | 35.6 | 0.960 |
| EnKo | 52.4 | 0.861 | 51.8 | 0.857 | 49.1 | 0.850 | 54.4 | 0.861 | 54.9 | 0.867 | 48.9 | 0.849 | 47.5 | 0.958 |
| EnPe | 28.3 | 0.899 | 32.6 | 0.908 | 30.8 | 0.903 | 34.8 | 0.911 | 30.5 | 0.901 | 29.6 | 0.898 | 34.7 | 0.964 |
| EnTa | 36.8 | 0.921 | 38.7 | 0.925 | 36.7 | 0.918 | 38.5 | 0.923 | 36.2 | 0.923 | 37.5 | 0.921 | 37.5 | 0.963 |
| EnTh | 42.4 | 0.909 | 42.6 | 0.907 | 37.0 | 0.892 | 42.8 | 0.907 | 42.1 | 0.906 | 37.1 | 0.890 | 40.1 | 0.954 |
| JnJk | 18.1 | 0.717 | 18.1 | 0.717 | 13.4 | 0.693 | 15.3 | 0.706 | 18.1 | 0.716 | 15.5 | 0.705 | 12.7 | 0.933 |
| ThEn | 39.2 | 0.912 | 40.2 | 0.913 | 31.1 | 0.882 | 38.9 | 0.911 | 38.6 | 0.912 | 33.6 | 0.897 | 38.3 | 0.960 |
| | **Large** | | | | | | | | | | | | | |
| | ① | | Ⓤ | | ② | | ③ | | ④ | | Ⓡ | | Ⓜ | |
| | **ACC** | **MFS** | **ACC** | **MFS** | **ACC** | **MFS** | **ACC** | **MFS** | **ACC** | **MFS** | **ACC** | **MFS** | **ACC** | **MFS** |
| ArEn | 52.2 | 0.954 | 54.3 | 0.954 | 0.7 | 0.682 | 55.4 | 0.954 | 55.6 | 0.953 | 57.1 | 0.955 | 59.7 | 0.979 |
| EnBa | 39.0 | 0.914 | 39.2 | 0.918 | 40.4 | 0.917 | 38.5 | 0.916 | 38.2 | 0.917 | 37.8 | 0.912 | 37.7 | 0.962 |
| EnHi | 42.0 | 0.923 | 43.0 | 0.926 | 38.9 | 0.914 | 45.7 | 0.929 | 46.1 | 0.928 | 40.7 | 0.916 | 45.0 | 0.968 |
| EnJa | 40.8 | 0.873 | 40.9 | 0.872 | 37.8 | 0.860 | 41.6 | 0.875 | 40.6 | 0.872 | 39.1 | 0.864 | 41.1 | 0.953 |
| EnKa | 36.2 | 0.913 | 37.9 | 0.914 | 35.0 | 0.909 | 37.5 | 0.913 | 38.6 | 0.915 | 38.2 | 0.913 | 39.2 | 0.961 |
| EnKo | 53.3 | 0.868 | 53.1 | 0.865 | 50.7 | 0.858 | 53.9 | 0.866 | 55.3 | 0.868 | 49.7 | 0.850 | 50.1 | 0.961 |
| EnPe | 34.0 | 0.911 | 34.6 | 0.913 | 32.5 | 0.906 | 34.2 | 0.912 | 35.3 | 0.911 | 33.4 | 0.911 | 34.3 | 0.964 |
| EnTa | 39.1 | 0.925 | 37.6 | 0.922 | 32.5 | 0.901 | 38.5 | 0.925 | 40.2 | 0.927 | 37.1 | 0.919 | 40.3 | 0.965 |
| EnTh | 43.6 | 0.910 | 43.7 | 0.909 | 32.5 | 0.869 | 43.7 | 0.909 | 43.9 | 0.910 | 40.0 | 0.897 | 41.3 | 0.955 |
| JnJk | 18.1 | 0.721 | 18.4 | 0.721 | 0.1 | 0.483 | 17.2 | 0.721 | 17.6 | 0.720 | 0.2 | 0.458 | 12.6 | 0.934 |
| ThEn | 40.2 | 0.915 | 40.3 | 0.915 | 37.7 | 0.909 | 39.2 | 0.915 | 40.3 | 0.916 | 32.9 | 0.897 | 39.8 | 0.962 |

Table 7: Full breakdown of NEWS2015

| | Small | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ① | | Ⓤ | | ② | | ③ | | ④ | | Ⓡ | | Ⓜ | |
| | **ACC** | **MLD** | **ACC** | **MLD** | **ACC** | **MLD** | **ACC** | **MLD** | **ACC** | **MLD** | **ACC** | **MLD** | **ACC** | **MLD** |
| albanian-high | 97.2 | 0.048 | 98.5 | 0.023 | 97.9 | 0.043 | 98.1 | 0.031 | 98.1 | 0.045 | 98.5 | 0.029 | 94.5 | 0.150 |
| arabic-high | 89.2 | 0.396 | 79.1 | 0.792 | 91.0 | 0.419 | 90.4 | 0.360 | 91.7 | 0.377 | 90.2 | 0.438 | 89.1 | 0.340 |
| armenian-high | 94.6 | 0.106 | 95.3 | 0.086 | 90.1 | 0.214 | 95.2 | 0.080 | 95.5 | 0.080 | 93.7 | 0.126 | 94.4 | 0.108 |
| basque-high | 100.0 | 0.000 | 100.0 | 0.000 | 100.0 | 0.000 | 99.0 | 0.010 | 97.0 | 0.060 | 100.0 | 0.000 | 95.0 | 0.140 |
| bengali-high | 98.0 | 0.060 | 98.0 | 0.060 | 99.0 | 0.030 | 99.0 | 0.050 | 97.0 | 0.090 | 98.0 | 0.080 | 98.0 | 0.040 |
| bulgarian-high | 88.9 | 0.165 | 88.3 | 0.188 | 96.1 | 0.067 | 94.1 | 0.101 | 93.9 | 0.115 | 95.4 | 0.077 | 96.5 | 0.058 |
| catalan-high | 96.8 | 0.083 | 96.9 | 0.083 | 96.8 | 0.075 | 97.5 | 0.063 | 97.2 | 0.073 | 97.3 | 0.068 | 96.3 | 0.076 |
| czech-high | 90.3 | 0.170 | 92.3 | 0.145 | 90.6 | 0.167 | 91.9 | 0.157 | 92.2 | 0.139 | 78.2 | 0.468 | 91.4 | 0.152 |
| danish-high | 88.9 | 0.166 | 88.9 | 0.174 | 90.1 | 0.151 | 89.1 | 0.170 | 90.2 | 0.148 | 91.4 | 0.132 | 92.6 | 0.118 |
| dutch-high | 93.7 | 0.112 | 94.8 | 0.100 | 95.2 | 0.090 | 95.2 | 0.090 | 94.9 | 0.086 | 94.2 | 0.097 | 94.8 | 0.093 |
| english-high | 96.1 | 0.077 | 96.1 | 0.071 | 90.4 | 0.203 | 96.5 | 0.074 | 95.7 | 0.078 | 95.3 | 0.087 | 96.3 | 0.062 |
| estonian-high | 95.2 | 0.109 | 96.0 | 0.078 | 96.4 | 0.070 | 96.3 | 0.090 | 96.8 | 0.079 | 96.2 | 0.091 | 93.0 | 0.145 |
| faroese-high | 79.9 | 0.420 | 79.2 | 0.390 | 78.4 | 0.449 | 79.5 | 0.413 | 82.9 | 0.365 | 81.6 | 0.392 | 82.5 | 0.333 |
| finnish-high | 86.6 | 0.318 | 81.9 | 0.278 | 86.5 | 0.216 | 88.2 | 0.202 | 90.2 | 0.271 | 84.1 | 0.311 | 86.1 | 0.227 |
| french-high | 84.5 | 0.291 | 85.3 | 0.270 | 85.2 | 0.292 | 83.8 | 0.317 | 85.7 | 0.262 | 82.3 | 0.332 | 86.5 | 0.253 |
| georgian-high | 97.6 | 0.039 | 95.4 | 0.083 | 97.8 | 0.037 | 97.9 | 0.113 | 97.5 | 0.046 | 98.2 | 0.039 | 97.3 | 0.038 |
| german-high | 89.6 | 0.244 | 88.4 | 0.272 | 88.1 | 0.282 | 89.6 | 0.257 | 89.3 | 0.179 | 83.7 | 0.381 | 88.9 | 0.276 |
| haida-high | 97.0 | 0.040 | 98.0 | 0.030 | 98.0 | 0.030 | 99.0 | 0.020 | 97.0 | 0.040 | 98.0 | 0.030 | 92.0 | 0.150 |
| hebrew-high | 99.0 | 0.010 | 98.8 | 0.013 | 99.1 | 0.010 | 97.5 | 0.027 | 97.8 | 0.027 | 97.8 | 0.026 | 98.7 | 0.016 |
| hindi-high | 95.1 | 0.482 | 99.9 | 0.002 | 100.0 | 0.000 | 100.0 | 0.000 | 100.0 | 0.000 | 100.0 | 0.000 | 99.4 | 0.014 |
| hungarian-high | 83.4 | 0.338 | 83.9 | 0.336 | 85.2 | 0.333 | 82.5 | 0.372 | 82.3 | 0.381 | 83.2 | 0.444 | 83.0 | 0.367 |
| icelandic-high | 82.2 | 0.333 | 82.0 | 0.350 | 84.1 | 0.305 | 84.5 | 0.304 | 86.3 | 0.286 | 84.4 | 0.296 | 84.5 | 0.300 |
| irish-high | 87.4 | 0.387 | 84.4 | 0.454 | 89.0 | 0.333 | 87.9 | 0.351 | 90.6 | 0.289 | 88.3 | 0.332 | 88.5 | 0.335 |
| italian-high | 96.1 | 0.101 | 87.2 | 0.251 | 96.0 | 0.099 | 95.5 | 0.111 | 95.7 | 0.106 | 95.5 | 0.105 | 94.6 | 0.120 |
| khaling-high | 99.2 | 0.008 | 98.9 | 0.018 | 98.7 | 0.016 | 98.0 | 0.030 | 98.7 | 0.018 | 98.7 | 0.024 | 98.1 | 0.028 |
| kurmanji-high | 94.2 | 0.123 | 93.7 | 0.101 | 92.3 | 0.184 | 93.2 | 0.126 | 93.8 | 0.098 | 93.1 | 0.143 | 94.0 | 0.074 |
| latin-high | 65.4 | 0.578 | 65.9 | 0.591 | 69.6 | 0.516 | 70.1 | 0.476 | 72.1 | 0.458 | 68.6 | 0.503 | 70.7 | 0.450 |
| latvian-high | 94.7 | 0.084 | 95.3 | 0.071 | 93.7 | 0.104 | 95.1 | 0.090 | 95.5 | 0.081 | 94.2 | 0.101 | 93.6 | 0.100 |
| lithuanian-high | 87.0 | 0.178 | 87.8 | 0.247 | 84.9 | 0.233 | 86.9 | 0.196 | 89.1 | 0.162 | 87.4 | 0.201 | 80.9 | 0.258 |
| lower-sorbian-high | 94.6 | 0.111 | 93.7 | 0.112 | 94.8 | 0.100 | 93.4 | 0.138 | 95.2 | 0.096 | 94.8 | 0.103 | 94.2 | 0.108 |
| macedonian-high | 94.1 | 0.088 | 94.2 | 0.089 | 95.3 | 0.073 | 90.7 | 0.164 | 93.6 | 0.102 | 94.7 | 0.087 | 94.9 | 0.094 |
| navajo-high | 84.9 | 0.446 | 84.6 | 0.461 | 81.2 | 0.468 | 86.2 | 0.332 | 88.5 | 0.268 | 85.1 | 0.356 | 79.8 | 0.450 |
| northern-sami-high | 93.9 | 0.112 | 94.2 | 0.099 | 94.8 | 0.125 | 93.6 | 0.145 | 95.4 | 0.089 | 93.2 | 0.143 | 91.8 | 0.154 |
| norwegian-bokmal-high | 86.4 | 0.220 | 87.6 | 0.293 | 88.4 | 0.193 | 89.7 | 0.172 | 90.0 | 0.158 | 88.2 | 0.198 | 90.9 | 0.156 |
| norwegian-nynorsk-high | 71.8 | 0.454 | 78.1 | 0.363 | 76.5 | 0.392 | 77.9 | 0.378 | 77.4 | 0.379 | 81.0 | 0.324 | 88.4 | 0.197 |
| persian-high | 99.4 | 0.013 | 99.4 | 0.013 | 99.4 | 0.012 | 99.1 | 0.016 | 98.9 | 0.017 | 99.2 | 0.017 | 96.8 | 0.064 |
| polish-high | 89.8 | 0.245 | 86.5 | 0.306 | 82.2 | 0.424 | 90.7 | 0.237 | 89.9 | 0.258 | 88.9 | 0.297 | 90.8 | 0.198 |
| portuguese-high | 98.0 | 0.032 | 96.3 | 0.060 | 96.4 | 0.056 | 98.5 | 0.034 | 98.9 | 0.024 | 97.9 | 0.036 | 98.0 | 0.034 |
| quechua-high | 99.4 | 0.013 | 97.9 | 0.045 | 98.7 | 0.040 | 98.2 | 0.053 | 99.6 | 0.020 | 98.2 | 0.058 | 96.4 | 0.087 |
| romanian-high | 83.3 | 0.649 | 83.6 | 0.454 | 75.6 | 0.711 | 81.6 | 0.482 | 84.8 | 0.432 | 83.8 | 0.473 | 84.8 | 0.404 |
| russian-high | 89.4 | 0.263 | 86.8 | 0.303 | 71.2 | 0.960 | 90.1 | 0.271 | 90.1 | 0.242 | 87.7 | 0.312 | 89.8 | 0.233 |
| serbo-croatian-high | 89.6 | 0.209 | 86.8 | 0.241 | 89.0 | 0.244 | 87.2 | 0.256 | 89.5 | 0.236 | 89.9 | 0.225 | 91.8 | 0.159 |
| slovak-high | 90.5 | 0.155 | 89.8 | 0.168 | 87.2 | 0.212 | 91.3 | 0.145 | 89.6 | 0.163 | 90.2 | 0.158 | 92.6 | 0.128 |
| slovene-high | 94.7 | 0.101 | 94.1 | 0.110 | 95.3 | 0.094 | 96.3 | 0.070 | 96.2 | 0.080 | 94.9 | 0.114 | 95.9 | 0.068 |
| sorani-high | 89.3 | 0.131 | 90.0 | 0.124 | 89.2 | 0.127 | 87.7 | 0.148 | 88.4 | 0.149 | 88.9 | 0.143 | 84.0 | 0.222 |
| spanish-high | 95.9 | 0.080 | 94.3 | 0.104 | 96.0 | 0.083 | 95.1 | 0.105 | 95.5 | 0.099 | 94.6 | 0.118 | 94.2 | 0.099 |
| swedish-high | 87.1 | 0.212 | 87.2 | 0.214 | 87.8 | 0.208 | 88.9 | 0.167 | 88.7 | 0.185 | 69.6 | 0.880 | 90.2 | 0.165 |
| turkish-high | 97.1 | 0.069 | 96.9 | 0.078 | 95.6 | 0.104 | 96.8 | 0.073 | 95.4 | 0.099 | 95.5 | 0.127 | 93.3 | 0.146 |
| ukrainian-high | 90.4 | 0.171 | 91.1 | 0.157 | 89.0 | 0.188 | 90.3 | 0.168 | 91.8 | 0.141 | 91.6 | 0.139 | 92.9 | 0.111 |
| urdu-high | 99.4 | 0.009 | 99.6 | 0.007 | 99.2 | 0.012 | 99.5 | 0.009 | 99.4 | 0.009 | 99.7 | 0.005 | 98.2 | 0.027 |
| welsh-high | 96.0 | 0.070 | 96.0 | 0.080 | 98.0 | 0.040 | 96.0 | 0.070 | 99.0 | 0.030 | 98.0 | 0.040 | 98.0 | 0.040 |

Table 8: Full breakdown of SIGMORPHON2017 with small model

| | Large | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ① | | Ⓤ | | ② | | ③ | | ④ | | Ⓡ | | Ⓜ | |
| | **ACC** | **MLD** | **ACC** | **MLD** | **ACC** | **MLD** | **ACC** | **MLD** | **ACC** | **MLD** | **ACC** | **MLD** | **ACC** | **MLD** |
| albanian-high | 97.9 | 0.037 | 96.3 | 0.066 | 98.5 | 0.024 | 98.0 | 0.030 | 98.5 | 0.028 | 98.8 | 0.021 | 96.1 | 0.123 |
| arabic-high | 90.4 | 0.496 | 89.8 | 0.397 | 88.4 | 0.456 | 89.8 | 0.358 | 92.3 | 0.352 | 91.9 | 0.437 | 90.8 | 0.267 |
| armenian-high | 94.3 | 0.112 | 95.6 | 0.166 | 94.3 | 0.106 | 94.9 | 0.078 | 95.8 | 0.075 | 94.6 | 0.095 | 93.6 | 0.110 |
| basque-high | 100.0 | 0.000 | 100.0 | 0.000 | 99.0 | 0.030 | 100.0 | 0.000 | 100.0 | 0.000 | 100.0 | 0.000 | 99.0 | 0.020 |
| bengali-high | 99.0 | 0.020 | 99.0 | 0.050 | 99.0 | 0.050 | 99.0 | 0.050 | 99.0 | 0.050 | 95.0 | 0.150 | 98.0 | 0.050 |
| bulgarian-high | 95.0 | 0.079 | 91.9 | 0.129 | 93.6 | 0.095 | 96.5 | 0.059 | 96.8 | 0.052 | 96.5 | 0.061 | 96.8 | 0.057 |
| catalan-high | 95.8 | 0.102 | 96.9 | 0.074 | 97.1 | 0.068 | 96.9 | 0.064 | 97.9 | 0.056 | 97.4 | 0.064 | 96.3 | 0.074 |
| czech-high | 89.2 | 0.184 | 87.4 | 0.209 | 87.8 | 0.241 | 90.7 | 0.176 | 92.7 | 0.133 | 89.7 | 0.188 | 92.0 | 0.140 |
| danish-high | 88.3 | 0.167 | 87.0 | 0.282 | 89.3 | 0.166 | 88.6 | 0.159 | 91.9 | 0.121 | 91.7 | 0.127 | 92.2 | 0.121 |
| dutch-high | 93.9 | 0.109 | 93.3 | 0.120 | 94.2 | 0.107 | 94.6 | 0.099 | 95.7 | 0.082 | 95.8 | 0.075 | 95.6 | 0.078 |
| english-high | 95.9 | 0.082 | 95.7 | 0.239 | 93.1 | 0.151 | 95.5 | 0.081 | 96.3 | 0.069 | 96.0 | 0.078 | 96.4 | 0.055 |
| estonian-high | 95.5 | 0.235 | 96.9 | 0.069 | 96.7 | 0.069 | 96.3 | 0.082 | 97.6 | 0.064 | 96.1 | 0.086 | 92.7 | 0.140 |
| faroese-high | 80.8 | 0.388 | 80.7 | 0.468 | 80.1 | 0.374 | 83.0 | 0.365 | 84.3 | 0.327 | 82.3 | 0.371 | 84.5 | 0.314 |
| finnish-high | 89.7 | 0.323 | 86.5 | 0.196 | 26.2 | 4.562 | 88.5 | 0.212 | 92.2 | 0.137 | 90.4 | 0.169 | 88.0 | 0.197 |
| french-high | 82.2 | 0.329 | 83.3 | 0.316 | 84.0 | 0.305 | 82.9 | 0.335 | 85.5 | 0.274 | 84.7 | 0.296 | 85.6 | 0.273 |
| georgian-high | 97.5 | 0.043 | 96.8 | 0.060 | 96.9 | 0.056 | 96.8 | 0.064 | 98.2 | 0.027 | 98.4 | 0.022 | 98.6 | 0.018 |
| german-high | 87.7 | 0.459 | 82.4 | 0.505 | 76.5 | 0.751 | 87.4 | 0.309 | 91.3 | 0.141 | 88.7 | 0.233 | 89.4 | 0.244 |
| haida-high | 98.0 | 0.030 | 98.0 | 0.030 | 98.0 | 0.030 | 97.0 | 0.040 | 98.0 | 0.030 | 98.0 | 0.030 | 95.0 | 0.100 |
| hebrew-high | 98.5 | 0.019 | 99.1 | 0.011 | 98.5 | 0.017 | 98.6 | 0.015 | 98.4 | 0.018 | 98.7 | 0.016 | 98.5 | 0.018 |
| hindi-high | 100.0 | 0.000 | 99.7 | 0.099 | 99.9 | 0.001 | 99.8 | 0.002 | 99.9 | 0.003 | 100.0 | 0.000 | 99.8 | 0.006 |
| hungarian-high | 82.7 | 0.361 | 82.0 | 0.362 | 81.7 | 0.400 | 79.0 | 0.437 | 84.1 | 0.347 | 82.0 | 0.386 | 84.7 | 0.329 |
| icelandic-high | 82.6 | 0.341 | 84.4 | 0.309 | 82.9 | 0.338 | 84.7 | 0.301 | 88.0 | 0.244 | 87.6 | 0.248 | 87.6 | 0.245 |
| irish-high | 85.8 | 0.377 | 86.4 | 0.391 | 81.8 | 0.512 | 88.7 | 0.423 | 90.5 | 0.254 | 88.3 | 0.344 | 89.6 | 0.304 |
| italian-high | 95.8 | 0.101 | 96.1 | 0.089 | 95.6 | 0.113 | 96.6 | 0.081 | 96.5 | 0.084 | 96.2 | 0.091 | 96.0 | 0.104 |
| khaling-high | 99.4 | 0.006 | 99.2 | 0.009 | 99.4 | 0.006 | 99.3 | 0.009 | 99.5 | 0.005 | 99.4 | 0.009 | 97.8 | 0.031 |
| kurmanji-high | 93.0 | 0.162 | 93.2 | 0.268 | 91.3 | 0.183 | 93.1 | 0.128 | 93.0 | 0.089 | 92.7 | 0.146 | 94.0 | 0.076 |
| latin-high | 71.6 | 0.518 | 69.6 | 0.513 | 70.4 | 0.485 | 78.0 | 0.371 | 78.4 | 0.361 | 76.2 | 0.392 | 74.7 | 0.378 |
| latvian-high | 80.8 | 0.524 | 91.5 | 0.133 | 94.6 | 0.095 | 94.3 | 0.097 | 96.3 | 0.056 | 95.7 | 0.078 | 95.2 | 0.068 |
| lithuanian-high | 86.4 | 0.368 | 85.4 | 0.394 | 87.9 | 0.173 | 89.7 | 0.150 | 90.6 | 0.126 | 90.7 | 0.139 | 89.4 | 0.149 |
| lower-sorbian-high | 93.9 | 0.118 | 93.7 | 0.208 | 95.2 | 0.094 | 95.2 | 0.094 | 95.1 | 0.100 | 96.3 | 0.073 | 95.6 | 0.083 |
| macedonian-high | 90.8 | 0.126 | 92.7 | 0.111 | 94.4 | 0.085 | 93.7 | 0.097 | 95.9 | 0.067 | 94.6 | 0.085 | 93.8 | 0.110 |
| navajo-high | 88.1 | 0.268 | 86.9 | 0.312 | 90.8 | 0.198 | 88.3 | 0.435 | 91.3 | 0.201 | 88.6 | 0.279 | 84.5 | 0.359 |
| northern-sami-high | 94.9 | 0.129 | 95.8 | 0.082 | 95.3 | 0.121 | 95.9 | 0.087 | 97.5 | 0.075 | 96.4 | 0.103 | 95.0 | 0.090 |
| norwegian-bokmal-high | 84.4 | 0.331 | 85.1 | 0.403 | 88.4 | 0.272 | 88.1 | 0.199 | 88.7 | 0.190 | 89.6 | 0.178 | 91.0 | 0.152 |
| norwegian-nynorsk-high | 73.3 | 0.440 | 76.2 | 0.408 | 78.2 | 0.376 | 79.4 | 0.354 | 80.4 | 0.345 | 82.6 | 0.314 | 89.3 | 0.182 |
| persian-high | 99.6 | 0.006 | 99.2 | 0.011 | 99.3 | 0.108 | 99.5 | 0.014 | 99.3 | 0.014 | 99.6 | 0.011 | 96.7 | 0.066 |
| polish-high | 85.3 | 0.406 | 85.7 | 0.369 | 88.5 | 0.282 | 88.4 | 0.366 | 89.7 | 0.248 | 89.2 | 0.251 | 90.2 | 0.193 |
| portuguese-high | 97.4 | 0.041 | 97.4 | 0.042 | 98.2 | 0.034 | 97.9 | 0.037 | 98.3 | 0.032 | 98.9 | 0.023 | 98.8 | 0.028 |
| quechua-high | 98.9 | 0.019 | 97.9 | 0.059 | 97.8 | 0.048 | 98.6 | 0.037 | 98.9 | 0.032 | 98.8 | 0.037 | 97.7 | 0.057 |
| romanian-high | 84.9 | 0.594 | 83.4 | 0.568 | 47.1 | 2.794 | 85.1 | 0.457 | 86.7 | 0.400 | 86.8 | 0.422 | 86.4 | 0.450 |
| russian-high | 88.0 | 0.353 | 85.8 | 0.458 | 86.9 | 0.324 | 89.8 | 0.432 | 90.5 | 0.244 | 90.3 | 0.244 | 91.2 | 0.220 |
| serbo-croatian-high | 84.9 | 0.307 | 85.0 | 0.277 | 88.8 | 0.323 | 88.8 | 0.236 | 90.9 | 0.187 | 82.0 | 0.426 | 91.4 | 0.185 |
| slovak-high | 87.7 | 0.214 | 89.4 | 0.270 | 89.2 | 0.264 | 89.6 | 0.186 | 92.1 | 0.126 | 90.4 | 0.163 | 92.6 | 0.129 |
| slovene-high | 94.6 | 0.096 | 93.2 | 0.222 | 94.6 | 0.102 | 95.2 | 0.078 | 96.1 | 0.073 | 94.9 | 0.097 | 96.4 | 0.063 |
| sorani-high | 88.8 | 0.138 | 88.4 | 0.144 | 88.9 | 0.133 | 89.4 | 0.123 | 90.3 | 0.121 | 90.1 | 0.120 | 85.9 | 0.180 |
| spanish-high | 94.3 | 0.092 | 95.8 | 0.163 | 96.1 | 0.072 | 96.1 | 0.075 | 96.7 | 0.056 | 96.2 | 0.072 | 95.4 | 0.079 |
| swedish-high | 85.7 | 0.237 | 85.0 | 0.329 | 79.4 | 0.529 | 85.9 | 0.235 | 90.9 | 0.157 | 89.6 | 0.170 | 92.2 | 0.132 |
| turkish-high | 95.9 | 0.103 | 94.9 | 0.226 | 94.8 | 0.118 | 92.4 | 0.172 | 97.0 | 0.063 | 94.8 | 0.121 | 94.3 | 0.128 |
| ukrainian-high | 90.4 | 0.170 | 90.3 | 0.161 | 90.8 | 0.148 | 92.5 | 0.126 | 92.9 | 0.116 | 93.1 | 0.114 | 91.4 | 0.144 |
| urdu-high | 99.3 | 0.012 | 99.4 | 0.010 | 99.0 | 0.015 | 99.6 | 0.007 | 99.3 | 0.014 | 99.6 | 0.006 | 99.1 | 0.013 |
| welsh-high | 96.0 | 0.060 | 98.0 | 0.050 | 98.0 | 0.040 | 97.0 | 0.050 | 97.0 | 0.070 | 97.0 | 0.060 | 97.0 | 0.060 |

Table 9: Full breakdown of SIGMORPHON2017 with large model