CLUZH at SIGMORPHON 2020 Shared Task on Multilingual Grapheme-to-Phoneme Conversion

Peter Makarov **Simon Clematide**
Institute of Computational Linguistics
University of Zurich, Switzerland
makarov@cl.uzh.ch simon.clematide@cl.uzh.ch

Abstract

This paper describes the submission by the team from the Institute of Computational Linguistics, Zurich University, to the Multilingual Grapheme-to-Phoneme Conversion (G2P) Task of the SIGMORPHON 2020 challenge. The submission adapts our system from the 2018 edition of the SIGMORPHON shared task. Our system is a neural transducer that operates over explicit edit actions and is trained with imitation learning. It is well-suited for morphological string transduction partly because it exploits the fact that the input and output character alphabets overlap. The challenge posed by G2P has been to adapt the model and the training procedure to work with disjoint alphabets. We adapt the model to use substitution edits and train it with a weighted finite-state transducer acting as the expert policy. An ensemble of such models produces competitive results on G2P. Our submission ranks second out of 23 submissions by a total of nine teams.

1 Introduction

G2P requires mapping a sequence of characters in some language into a sequence of International Phonetic Alphabet (IPA) symbols, which represent the pronunciation of this input character sequence in some abstract way (not necessarily phonemic, despite the name of the task) (Figure 1).

Multilingual G2P is Task I of this year's SIGMORPHON challenge. It features fifteen languages from various phylogenetic families and written in different scripts. We refer the reader to Gorman et al. (2020) for an overview of the language data. Each language comes with 3,600 training and 450 development set examples. It is permitted to use external resources as well as to build a single multilingual model.

We participate in this shared task with an adaptation of our SIGMORPHON 2018 system (Makarov and Clematide, 2018b), which was particularly successful in type-level morphological inflection generation. Our system is a neural transducer that operates over explicit edit actions and is trained with imitation learning (Daumé III et al., 2009; Ross et al., 2011; Chang et al., 2015, IL). It has a number of useful inductive biases, one of which is the familiar bias towards copying the input (implemented as the traditional copy edit). This is particularly useful for morphological string transduction problems, which typically involve small and local edits and where most of the input is preserved in the output. This contrasts with models that rely purely on generating characters such as generic encoder-decoder models, which as a result suffer, particularly on smaller-sized datasets.

fathaigh ↦ /faː/ ("giants")
Irish of Cois Fhairrge (de Bhaldraithe, 1953)

Figure 1: Example of G2P.

Copying requires that the input and output character alphabets overlap, preferably substantially. This also allows our IL training to leverage a simple-to-implement expert policy (which during training provides demonstrations to the learner of how to optimally solve the task). The optimal completion of the target given the prediction generated so far during training requires finding edits that would extend the prediction so that the Levenshtein distance (Levenshtein, 1966) between the target and the partial prediction + the future suffix is minimized. Unfortunately, this objective alone would not discriminate between multiple edit action sequences that relate the input and the partial prediction + the future suffix. To address this spurious ambiguity, our IL training adds edit sequence scores, computed using traditional costs,[1] into the

[1] Copy costs zero, all other edits cost one.

Proceedings of the Seventeenth SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, pages 171–176

https://doi.org/10.18653/v1/P17

objective. This naturally encourages the system to copy, however this would fail on any editing problem with disjoint alphabets.

G2P poses an interesting challenge for a system like ours. On the one hand, G2P shares many similarities with morphological string transduction: The changes are mostly local, it would suffice to perform traditional left-to-right transduction, and a substantial part of the work is arguably applying equivalence rules (e.g. the German letter "g" most often converts to /g/, "a" to /a/ or /a:/), which is similar to copying. Yet, a general solution to G2P cannot rely on overlapping alphabets since many scripts do not share many symbols, if any at all, with IPA (e.g. Korean or Georgian).

Our solution adapts the model to use substitution edits and trains it with a weighted finite-state transducer acting as the expert policy.

2 Model description

The underlying model is a neural transducer introduced in Aharoni and Goldberg (2017). It defines a conditional distribution over traditional edits $p_\theta(\mathbf{y}, \mathbf{a} \mid \mathbf{x}) = \prod_{j=1}^{|\mathbf{a}|} p_\theta(a_j \mid a_{<j}, \mathbf{x})$, where $\mathbf{x}$ is an input sequence of graphemes and $\mathbf{a} = a_1 \dots a_{|\mathbf{a}|}$ is an edit action sequence. (The output sequence of IPA symbols $\mathbf{y}$ is deterministically computed from $\mathbf{x}$ and $\mathbf{a}$.) The model is equipped with a long short-term memory (LSTM) decoder and a bidirectional LSTM encoder (Graves and Schmidhuber, 2005). The challenge is training this model: Due to the recurrent decoder, it cannot be trained with exact marginal likelihood unlike the more familiar weighted finite-state transducer (Mohri, 2004; Eisner, 2002, WFST) or its neuralizations (Yu et al., 2016). For a more detailed description of the model, we refer the reader to Makarov and Clematide (2018a).[2]

IL training Makarov and Clematide (2018a) propose training the model using IL, a general model fitting framework for sequential problems over exponentially sized output spaces. IL has been applied successfully to natural language processing (NLP) problems, e.g. transition-based parsing (Goldberg and Nivre, 2012) and language generation (Welleck et al., 2019). IL relies on the availability of demonstrations of how the task can optimally be solved given any configuration. Due to the nature of many NLP problems, such demonstrations can often be provided by a rule-based program (known as expert policy).

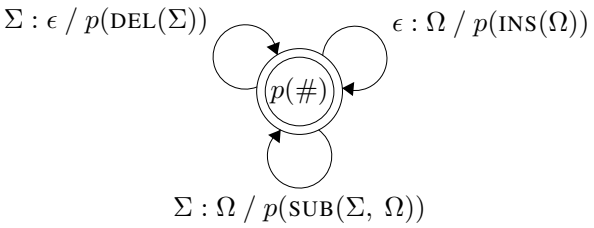

Figure 2: Stochastic edit distance (Ristad and Yianilos, 1998): A memoryless probabilistic FST. Σ and Ω stand for any input and output symbol, respectively.

Makarov and Clematide (2018a) use a combination of Levenshtein distance and edit sequence cost as the task objective ($\beta\,\text{ED}(\hat{\mathbf{y}}, \mathbf{y}) + \text{ED}(\mathbf{x}, \hat{\mathbf{y}}), \beta \geq 1$) and devise an expert policy for it. Given a target sequence $\mathbf{y}$, a partially completed prediction $\hat{\mathbf{y}}_{1:n}$, and the remaining input sequence $\mathbf{x}_{k:l}$, the expert needs to (1) identify the set of target suffixes $\mathbf{y}_{j:m}$ that when appended to $\hat{\mathbf{y}}_{1:n}$, lead to a prediction with minimum Levenshtein distance from the target, and (2) check which of the edit sequences producing those suffixes have the lowest cost, i.e. minimum Levenshtein distance from the remaining input.

The second part is crucial for training accurate models especially in the limited resource setting, as it reduces spurious ambiguity arising under the first part of the objective alone. It is also the second part of the training objective that hinges on the overlap of the input and output alphabets, as this permits minimization using the edit distance dynamic program with traditional costs.

2.1 Adaptation to G2P

The adaptation is two-fold: First, we introduce substitution edits, which have previously not been employed to keep the total number of edit actions to a minimum. For each output character c, there is now a substitution action $\text{SUBS}[c]$ which substitutes c for any input character x.

When the alphabets are disjoint, the completing edit sequences cannot be very informatively scored using traditional edit costs. For example, for the data sample кит $\mapsto$ /kʲit/ (Russian: "whale"), we would like the following most natural edit sequence to attain the lowest cost: $\text{SUBS}[\text{k}], \text{INS}[\text{ʲ}], \text{SUBS}[\text{i}], \text{SUBS}[\text{t}]$. Yet, it is clear

[2] The model uses shared input character / action embeddings of size 100 and one-layer LSTMs with hidden-state size 200.

that under traditional costs, this sequence attains the same cost as any other that consists of three substitutions and one insertion. Our solution to this is to learn costs from the training data to ensure an intuitive ranking of edit sequences.

SED policy Learning costs as well as computing string distance can be achieved with a very simple WFST: Stochastic Edit Distance (Ristad and Yianilos, 1998, SED), which is a probabilistic version of Levenshtein distance (Fig. 2). We use traditional multinomial parameterization.

Before starting training the neural transducer, we train a SED model using the Expectation–Maximization algorithm (Dempster et al., 1977). We use the following update in the M-step: $\theta^{(t+1)} \propto \max(0, \widetilde{\theta} + \alpha)$, where $\widetilde{\theta}$ is the unnormalized weight computed in the E-step and $0 < \alpha < 1$ is a sparse Dirichlet prior parameter associated with this edit. This corresponds to sparse regularization via Dirichlet prior (Johnson et al., 2007), which results in many edits having zero probability. We found this training to lead to more accurate SED models. Furthermore, it dramatically reduces the size of the edit action set that the neural transducer is defined over.

SED is integrated into the expert policy. During training, given a configuration consisting of a partial prediction, a remainder of the input, and the target, we query the expert policy for next optimal edits. We minimize the first part of the objective much like before, and we minimize the second part by decoding SED with the Viterbi algorithm.

Suppose we transduce the French word $\mathbf{x} = \textit{abject}$ ("vile") into the target $\mathbf{y} =$ a b ʒ ɛ k t. Suppose also that the neural transducer currently attends to character $x_4 = e$ and the prediction built so far during training is $\hat{\mathbf{y}}_{1:7} =$ a b ʒ e (note the error). We query the SED policy to get the optimal edit action whose likelihood we will maximize. First, much like before, we find that the following edits are optimal with respect to the first term of the training objective (call them *permissible*) as they do not increase the Levenshtein distance of the prediction from the target (assuming all subsequent edits are permissible too): SUBS[ɛ], INS[ɛ], DEL, SUBS[ ], INS[ ]. (This can be verified by looking at the Levenshtein distance prefix matrix for strings $\hat{\mathbf{y}}_{1:7}$ and $\mathbf{y}$.) Each such edit starts a suffix that completes the target, e.g. it is "ɛ k t" for SUBS[ɛ] and " k t" for SUBS[ ]. Next, we use SED to rank the permissible edits by cost-to-go. For each of the edits and their corresponding suffixes, the expert needs to execute the edit (e.g. SUBS[ɛ] writes ɛ and moves the attention to $x_5 = c$) and then decode SED with Viterbi on the the remaining input and the suffix (both possibly modified by the edit). In this way, we obtain that SUBS[ ] is the optimal action with the lowest cost-to-go (=negative sum of the log probabilities of the edit and of the Viterbi path) of 15.28 (vs 17.65 for SUBS[ɛ], 21.09 for INS[ɛ], 17.31 for DEL, and 17.31 for INS[ ]).[3]

Exploration This time, we also train the transducer with an aggressive exploration schedule: $p_{\text{sampling}}(i) = \frac{1}{1+\exp(i)}$, where i is the training epoch number. After a couple of training epochs, training configurations are generated entirely by executing edit actions sampled from the model.

3 Submission details

We train separate models for each language on the official training data and use the development set for model selection.[4] Our submission does not use any additional lexical resources.

For most of the models, we employ Unicode decomposition normalization (NFKD)[5] as a data preprocessing step. Importantly, this helps decomposing Unicode syllable blocks used e.g. in Hangul.

The size of the development set is rather small (450 examples), and having examined the data, we suspect that overly relying on the development set for model selection might hurt generalization. For example, the French development set contains three exceptions to the "ill"–/j/ equivalence; thus, a single model that achieves a high score on the development set might, in fact, be overfitting. To counter this, we build an eleven-model–strong majority-vote ensemble. Fortunately, training a neural transducer is fast as one epoch takes just about four minutes on average on a single CPU, due to the relatively small number of model parameters.

[3] This particular SED is trained on the French training data for 3 EM epochs with Dirichlet prior $\alpha =$ 1e-05 for all edits.

[4] We train the SED model for 20 epochs of EM with $\alpha = 0.25$ for insertions and 0.5 for all other edits. We train the neural transducer for a maximum of 60 epochs with a patience of 12 epochs. We use mini-batches of size 5. We decode using beam search with beam width 4.

[5] Using NFKD instead of NFD was a bit unfortunate because some superscript diacritics get normalized to their regular size. Luckily, as pointed out to us by Kyle Gorman, there is a unique mapping from NFKD to NFC for the spaced output format of this task. See http://www.unicode.org/reports/tr15/ for Unicode normalization forms.

	CLUZH		ENS.		CLUZH WER AVG				LSTM	TF	BEST BY OTHERS			
LNG	WER	PER	#C	#D	WER	±	Δ,%	⊥	WER	WER	WER	Δ,%	PER	Δ,%
ady	27.11	6.27	0	11	30.32	1.97	-12	16.89	28.00	28.44	**24.67**	9	**5.76**	8
arm	**12.22**	**2.82**	0	11	14.73	0.76	-21	8.89	14.67	14.22	12.67	-4	2.91	-3
bul	23.33	**4.70**	0	11	30.81	2.78	-32	13.78	31.11	34.00	**22.22**	5	**4.70**	0
dut	14.44	2.51	9	2	18.30	1.44	-27	9.33	16.44	15.78	**13.56**	6	**2.36**	6
fre	6.89	1.56	2	9	8.12	0.54	-18	3.56	6.22	6.89	**5.11**	26	**1.16**	26
geo	27.33	4.83	0	11	29.11	0.86	-7	8.89	26.44	28.00	**24.89**	9	**4.57**	5
gre	16.44	2.68	11	0	19.60	1.80	-19	7.33	18.89	18.89	**14.44**	12	**2.42**	10
hin	**5.11**	**1.20**	0	11	7.13	0.55	-40	2.67	6.67	9.56	**5.11**	0	**1.20**	0
hun	**4.00**	1.02	0	11	4.77	0.60	-19	2.89	5.33	5.33	**4.00**	0	**0.92**	10
ice	**9.11**	1.90	0	11	10.00	0.53	-10	5.78	10.00	10.22	**9.11**	0	**1.83**	4
jpn	6.00	1.58	0	11	7.19	0.30	-20	4.89	7.56	7.33	**4.89**	19	**1.16**	27
kor	28.44	4.88	0	11	28.26	1.39	1	11.78	46.89	43.78	**24.00**	16	**4.05**	17
lit	**18.67**	**3.27**	0	11	21.54	0.82	-15	14.22	19.11	20.67	**18.67**	0	3.38	-3
rum	11.33	2.68	0	11	13.66	1.11	-21	7.11	10.67	12.00	**9.78**	14	**2.23**	17
vie	1.56	0.35	0	11	1.60	0.21	-2	0.89	4.67	7.56	**0.89**	43	**0.27**	23
AVG	14.13	2.82	1.5	9.5	16.34	1.05	-16	7.93	16.84	17.51	**12.93**	8	**2.59**	8

Table 1: Overview of the test results. Δ gives relative error difference compared to our submission CLUZH. #C=number of NFC models in the ensemble. #D=number of NFKD models in the ensemble. CLUZH WER AVG=average WER, standard deviation, and relative error difference of the average computed over individual models. ⊥=lower-bound on WER: correct if predicted by any individual model. LSTM=official seq2seq LSTM baseline. TF=official seq2seq Transformer baseline. BEST BY OTHERS=best results of other systems for each language.

4 Results and Discussion

Our system ranks second among 23 submissions by a total of nine teams (Table 1). It ties for first place on four languages (Hindi, Hungarian, Icelandic, Lithuanian) and outperforms every other submission for Armenian. It achieves strong gains over the neural baselines.

Ensembling gains us 16% in error reduction compared to test set averages—a substantial improvement. We leave it for future work to see whether dropout and a larger model size could be used instead as effectively as ensembling. Unicode decomposition normalization boosts the performance of our Korean models.[6] On average, at least one model predicts the output correctly for all but 7.93% of all the words (⊥)—Adyghe, Lithuanian, and Bulgarian being the most difficult languages. For some languages, WER standard deviation is high, likely confirming our hypothesis that model selection on the small-sized development set would lead to poor generalization.

Error analysis Table 2 shows the most frequent errors of our system for each language and helps to qualitatively assess their strongly varying error profiles. We take a closer look at the errors in French and Korean. Additional lexical information could improve our French models. E.g. the word's lexical category feature and/or morphological segmentation would probably help correctly transduce the word-final "-ent" (adverb "vraiement" (*truly*) /...ɑ̃/ vs verb "viennent" (*they come*), where the ending is silent). Many errors in French are in English borrowings.

We look in some detail at the errors on the Korean test data that all or almost all of the individual models of the ensemble make. As expected, lexicalized phenomena contribute most of the errors: vowel length (which is neither phonemic nor phonetically realized in the speech of all except elderly speakers (Sohn, 2001)) and tensification. Vowel length is not indicated in Korean orthography, and neither is tensification (with some exceptions). Knowing whether a word is an English borrowing (e.g. 섹스 seksŭ[7] (*sex*)) or whether a word is a compound and where the morpheme boundary lies (초승달 ch'osŭng-tal (*new moon*)) could help predict non-automatic tensification correctly in a small number of cases ([s̬e̬ks̬ɯ] vs *[sʰ...] and [t͡ɕʰo̬sʰɯŋta[] vs *[...da[]).

[6] In fact, in a post-submission analysis, we see a strong gain from decomposition only for Korean (17 percentage points on average). For the other languages, it has no impact on performance on average.

[7] This uses McCune-Reischauer transliteration of Korean.

ady	’/ϵ/17	ə•/ϵ/9	ʃ/ʂ/8	ϵ/•ə/7	j•/ϵ/6	ϵ/’/6	ϵ/ə•/5	ɮ/l/5	:/ϵ/5	a/ə/5
arm	ɔ/o/17	ϵ/ə•/12	◌͡/•/12	ə•/ϵ/3	t/d/3	g/kʰ/2	ʃʰ/ʒ/2	ɛ/j/2	χ/ʁ/2	t͡ʃ/d•ʒ/1
bul	r/ɾ/26	o/ɔ/22	ə/a/14	a/ə/12	◌̪/ϵ/9	ϵ/◌̪/9	a/ɐ/7	ɬ/l/5	ɐ/ə/5	ϵ/ʲ/5
dut	ə/ɛ/9	ϵ/j•/4	aː/ɑ/4	eː/ə/4	ə/eː/3	t/d/3	:/ϵ/3	oː/ɔ/2	ϵ/ɛ•/2	n/m/2
fre	ϵ/•ɑ̃/2	ϵ/•s/2	a/ɑ/2	o/ɔ/2	w/ɔ/2	•j•ɑ̃/ϵ/1	ϵ/•k•s/1	ɔ•p/o/1	•ɑ̃/ϵ/1	e/ɛ•ʁ/1
geo	ɪ/i/103	i/ɪ/48	χ/x/5	ɣ/ʁ/4	ʁ/ɣ/3	x/χ/3	ɑ/a/2	•s/ϵ/1		
gre	ɾ/r/27	o/ɔ/19	r/ɾ/15	e/ɛ/9	j̩/i/3	n•/ϵ/2	ç/i/2	m/ɱ/2	•m•e/ϵ/1	ϵ/s•/1
hin	ϵ/ə•/10	ə•/ϵ/5	ϵ/•ə/2	ɛː/ə/2	ϵ/◡•/2	ɑ/a/2	ɪ/iː/1	•ɦ/ɦ/1	ɪ/i/1	ə/j/1
hun	ʃ/ʒ/3	ϵ/:/3	eː/i•n•t/1	ϵ/ɱ•v•/1	m/eːʲ/1	t͡s/xː/1	sː/ʃ•s/1	h•/ϵ/1	•h/◌͡ʃ/1	◌͡/•/1
ice	:/ϵ/11	ϵ/:/9	t•/ϵ/4	v/f/3	ϵ/◌̥/3	t/d/2	ʰ/ϵ/2	ϵ/ʰ/2	ʏ•ʏ/uː/1	cʰ/k/1
jpn	ϵ/◌̊/8	ϵ/◌̥/6	ϵ/:/3	:/•ɯ̟ᵝ/2	:/•o̞/2	ɯ̟ᵝ/j•o̞/1	ϵ/•e̞/1	ɯ̟ᵝ/e̞/1	o̞ː/ã/1	s•i/ɯ̃/1
kor	ϵ/ː/72	:/ϵ/18	ɜː/ʌ̹/11	ʑ/ɕ/4	ʌ̹/ɜː/4	d/t/4	g/k̟/3	ϵ/ɲ•/3	d/t̟/2	l/n/2
lit	ϵ/◌̪/15	n/ŋ/14	ɐ/aː/12	:/ϵ/8	ʲ/ϵ/7	o/ɔ/7	ϵ/:/6	ϵ/ʲ/5	ɛ/æː/3	aː/ɐ/3
rum	◌͡/•/8	•/◌͡/8	e̯/j/6	r/ɾ/5	ʲ/•i/4	:/•j/3	i/j/3	ϵ/e•/2	o/e/2	j/i/2
vie	ϵ/ː/2	⊣•/ϵ/1	ϵ/e•/1	w•/ϵ/1	ˀ•/ϵ/1	◌̄m/ϵ/1	ə/e/1	ʔ/n/1	˥/ϵ/1	a/ɔ/1

Table 2: Ten most frequent errors per language. Notation: prediction / gold / error frequency. • denotes whitespace. Computed using the UTF-8 aware version of the ISRI Analytic Tools for OCR Evaluation.[8]

How good is SED policy? Somewhat surprisingly, using SED as part of the expert policy results in competitive performance. Yet, SED is a very crude model (e.g. because of the lack of context, when used as a conditional model, SED assigns less probability to any edit sequence containing insertions than the same sequence but with all the insertions removed; this e.g. makes it unusable as a standalone model for G2P). On top of this, we also do not use learned roll-out, which would be recommended when training with a sub-optimal expert (Chang et al., 2015). We leave it for future work to examine whether the neural transducer's performance on G2P would improve from replacing SED with a more powerful model.

5 Conclusion

This presents the approach taken by the CLUZH team to solving the SIGMORPHON 2020 Multilingual Grapheme-to-Morpheme Conversion challenge. Our submission is based on our successful SIGMORPHON 2018 system, which is a majority-vote ensemble of neural transducers trained with imitation learning. We adapt the 2018 system to work on transduction problems with disjoint input and output alphabets. We add substitution actions (not available in previous versions of the system) and employ a memoryless probabilistic finite-state transducer to define the expert policy for the imitation learning. We use majority-vote ensembling to counter the overfitting to the small development sets. These simple modifications result in a highly competitive performance even without the use of any exernal resources or learning a single multilingual model. Our ensemble ranks second out of 23 submissions by a total of nine teams. Our error analysis indicates that addressing many of the errors requires additional information such as knowing the word's lexical category, morphological segmentation, or etymology. We will make our code publicly available.

[8] https://github.com/eddieantonio/ocreval

Acknowledgment

We would like to thank the organizers for their great effort in these turbulent times. We thank Kyle Gorman for taking the time to help us with our Unicode normalization problem. This work has been supported by the Swiss National Science Foundation under grant CR-SII5_173719.

References

Roee Aharoni and Yoav Goldberg. 2017. Morphological inflection generation with hard monotonic attention. In *ACL*.

Tomás de Bhaldraithe. 1953. *Gaeilge Chois Fhairrge: An Deilbhíocht*. Institiúid Ard-Léinn Bhaile Átha Cliath.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume III, and John Langford. 2015. Learning to search better than your teacher. In *ICML*.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning*.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1).

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

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *COLING*.

Kyle Gorman, Lucas F.E. Ashby, Aaron Goyzueta, Arya D. McCarthy, Shijie Wu, and Daniel You. 2020. The SIGMORPHON 2020 shared task on multilingual grapheme-to-phoneme conversion. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5).

Mark Johnson, Thomas L Griffiths, and Sharon Goldwater. 2007. Bayesian inference for PCFGs via Markov Chain Monte Carlo. In *NAACL-HLT*.

Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8).

Peter Makarov and Simon Clematide. 2018a. Imitation learning for neural morphological string transduction. In *EMNLP*.

Peter Makarov and Simon Clematide. 2018b. UZH at CoNLL-SIGMORPHON 2018 shared task on universal morphological reinflection. *Proceedings of the CoNLL SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection*.

Mehryar Mohri. 2004. Weighted finite-state transducer algorithms. An overview. In *Formal Languages and Applications*, volume 148 of *Studies in Fuzziness and Soft Computing*. Springer Berlin Heidelberg.

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

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*.

Ho-Min Sohn. 2001. *The Korean language*. Cambridge University Press.

Sean Welleck, Kianté Brantley, Hal Daumé III, and Kyunghyun Cho. 2019. Non-monotonic sequential text generation. In *ICML*.

Lei Yu, Jan Buys, and Phil Blunsom. 2016. Online segment to segment neural transduction. In *EMNLP*.