CLUZH at SIGMORPHON 2021 Shared Task on Multilingual Grapheme-to-Phoneme Conversion: Variations on a Baseline

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

Abstract

This paper describes the submission by the team from the Department of Computational Linguistics, Zurich University, to the Multilingual Grapheme-to-Phoneme Conversion (G2P) Task 1 of the SIGMORPHON 2021 challenge in the low and medium settings. The submission is a variation of our 2020 G2P system, which serves as the baseline for this year's challenge. The system is a neural transducer that operates over explicit edit actions and is trained with imitation learning. For this challenge, we experimented with the following changes: a) emitting phoneme segments instead of single character phonemes, b) input character dropout, c) a mogrifier LSTM decoder (Melis et al., 2019), d) enriching the decoder input with the currently attended input character, e) parallel BiLSTM encoders, and f) an adaptive batch size scheduler. In the low setting, our best ensemble improved over the baseline, however, in the medium setting, the baseline was stronger on average, although for certain languages improvements could be observed.

1 Introduction

The SIGMORPHON Grapheme-to-Phoneme Conversion task consists of mapping a sequence of characters in some language into a sequence of whitespace delimited International Phonetic Alphabet (IPA) symbols, which represent the pronunciation of this input character sequence (not necessarily a phonemic transcription, despite the name of the task) according to the language-specific conventions used in the English Wiktionary.[1] The data was collected and post-processed by the WikiPron project (Lee et al., 2020). Post-processing removes stress and syllable markers and applies IPA segmentation for combining and modifier diacritics as well as contour information. See Figure 1 for the post-processed shared task entries and the original entries from the Wiktionary pronunciation section. For more information, we refer the reader to the shared task overview paper (Ashby et al., 2021).

Lang.	Grapheme	Phoneme	Wiktionary
ice	persóna	pʰ ɛ ˌɾ s o u: n a	/ˈpʰɛˌɾ.sou:na/
fra	williams	w i l j a m z	/wi.ljamz/
bul	засичайки	z ɛ s ʧ ʃ ə j kʲ i	/zɛˈsiʧəjkʲi/
kor	검출	k ɘ: m ʧ͡ɕʰ u l	[ˈkʌ̹(:)mʧ͡ɕʰul]

Figure 1: Examples of the original G2P shared task data from four different languages and their pronunciation entries in Wiktionary.

In the low and medium data setting, the 2021 SIGMORPHON multilingual G2P challenge features ten different languages from various phylogenetic families and written in different scripts. The low setting comes with 800 training, 100 development and 100 test examples. In the medium setting, the data splits are 10 times larger. Although it is permitted to use external resources for the medium setting, all our models used exclusively the official training material.

Our system is a neural transducer with pointer network-like monotonic hard attention (Aharoni and Goldberg, 2017) that operates over explicit character edit actions and is trained with imitation learning (Daumé III et al., 2009; Ross et al., 2011; Chang et al., 2015). It is an adaptation of our type-level morphological inflection generation system that proved its data efficiency and performance in the SIGMORPHON 2018 shared task (Makarov and Clematide, 2018). G2P shares many similarities with traditional morphological string transduction: The changes are mostly local and often simple depending on how close the spelling of a language reflects pronunciation. For most languages, a substantial part of the work is actually

[1] https://en.wiktionary.org/

Proceedings of the Seventeenth SIGMORPHON Workshop on Computational Research

in Phonetics, Phonology, and Morphology,pages 148–153

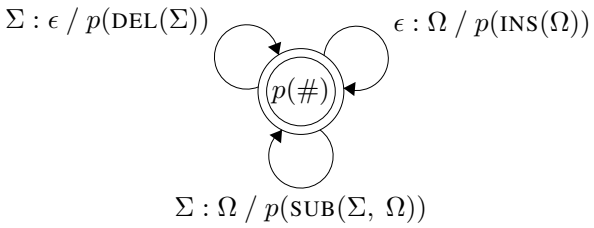

Figure 2: Stochastic edit distance (Ristad and Yianilos, 1998): A memoryless probabilistic FST. Σ and Ω stand for any input and output symbol, respectively. Transition weights are to the right of the slash and $p(\#)$ is the final weight.

applying character-by-character substitutions. An extreme case is Georgian, which features an almost deterministic one-to-one mapping between graphemes and IPA segments that can be learned almost perfectly from little training data.[2]

The main goal of our submission was to test whether our last year's system, which is the baseline for this year's G2P challenge, already exhausts the potential of its architecture, or whether changes to the output representation (IPA segments vs. IPA Unicode codepoints; input character dropout), to the LSTM decoder (the mogrifier steps and the additional input of the attended character), to the BiLSTM encoder (parallel encoders), or to other hyper-parameter settings (adaptive batch size) can improve the results without replacing the LSTM-based encoder/decoder setup by a Transformer-based architecture (see e.g. Wu et al. (2021) for Transformer-based state-of-the-art results).

2 Model description

The model defines a conditional distribution over substitution, insertion and deletion edits $p_\theta(\mathbf{a} \mid \mathbf{x}) = \prod_{j=1}^{|\mathbf{a}|} p_\theta(a_j \mid a_{<j}, \mathbf{x})$, where $\mathbf{x} = x_1 \dots x_{|\mathbf{x}|}$ 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 an LSTM decoder and a bidirectional LSTM encoder (Graves and Schmidhuber, 2005). At each decoding step j, the model attends to a single grapheme $\mathbf{x}_i$. The attention steps monotonically through the input sequence, steered by the edits that consume input (e.g. a deletion shifts the attention to the next grapheme $\mathbf{x}_{i+1}$).

[2]Even a reduced training set of only 100 items allows a single model to achieve over 90% accuracy on the Georgian test set.

The imitation learning algorithm relies on an expert policy for suggesting intuitive and appropriate character substitution, insertion and deletion actions. For instance, for the data sample КИТ $\mapsto$ /kʲit/ (Russian: "whale"), we would like the following most natural edit sequence to attain the lowest cost: SUBS[k], INS[ʲ], SUBS[i], SUBS[t]. The cost function for these actions is estimated by fitting a Stochastic Edit Distance (SED) model (Ristad and Yianilos, 1998) on the training data, which is a memoryless weighted finite-state transducer shown in Figure 2. The resulting SED model is integrated into the expert policy, the SED policy, that uses Viterbi decoding to compute optimal edit action sequences for any point in the action search space: Given a transducer configuration of partially processed input, find the best edit actions to generate the remaining target sequence suffix. During training, an aggressive exploration schedule $p_{\text{sampling}}(i) = \frac{1}{1+\exp(i)}$ where i is the training epoch number, exposes the model to configurations sampled by executing edit actions from the model. For an extended description of the SED policy and IL training, we refer the reader to the last year's system description paper (Makarov and Clematide, 2020).

2.1 Changes to the baseline model

This section describes the changes that we implemented in our submissions.

IPA segments vs. IPA Unicode characters: Emitting IPA segments in one action (including its whitespace delimiter), e.g., for the Russian example from above SUBS[kʲ•],[3] instead of producing the same output by three actions SUBS[k], INS[ʲ], INS[•] reduces the number of action predictions (and potential errors) considerably, which is beneficial. On the other hand, this might lead to larger action vocabularies and sparse training distributions. Therefore, we experimented with character (CHAR) and IPA segment (SEG) edit actions in our submission. Table 1 shows statistics on the resulting vocabulary sizes if CHAR or SEG actions are used. Some caution is needed though because some segments might only appear once in the training data, e.g. English has an IPA segment sːː that only appears in the word "psst".

Input character dropout: To prevent the model from memorizing the training set and to force it to learn about syllable contexts, we randomly replace

[3]• denotes whitespace symbol.

S	Language	NFD$^<$	SEG	C^{NFC}	C^{NFD}
L	ady	0.5%	67	37	37
L	gre	4.3%	33	33	33
L	ice	30.3%	60	36	36
L	ita	0.8%	32	29	29
L	khm	0.5%	47	36	34
L	lav	12.4%	73	51	36
L	mlt_latn	9.0%	41	29	29
L	rum	0.3%	45	31	31
L	slv	4.3%	48	38	30
L	wel_sw	2.4%	43	37	37
M	arm_e	0.0%	54	31	31
M	bul	3.5%	46	34	34
M	dut	0.8%	49	39	39
M	fre	0.1%	39	36	36
M	geo	0.0%	33	27	27
M	hbs_latn	3.7%	63	43	33
M	hun	42.5%	66	37	37
M	jpn_hira	36.1%	64	42	39
M	kor	99.8%	60	46	46
M	vie_hanoi	88.2%	49	44	44
H	eng_us	0.0%	124	83	80
Average		16.2%	54.1	39.0	37.0

Table 1: Statistics on Unicode normalization for low (L), medium (M), and high (H) settings (column S). Column NFD$^<$ specifies the percentage of training items where NFD normalized graphemes had smaller length difference to phonemes than in NFC normalization. Column SEG gives the vocabulary size of IPA segments (the counts are the same for NFC and NFD). Column C^{NFC} reports the phoneme vocabulary size in NFC Unicode characters (CHAR) and C^{NFD} in NFD.

an input character with the UNK symbol according to a linearly decaying schedule.[4]

Mogrifier LSTM decoder: Mogrifier LSTMs (Melis et al., 2019) iteratively and mutually update the hidden state of a previous time step with the current input before feeding the modified hidden state and input into a standard LSTM cell. On language modeling tasks with smaller corpora, this technique closed the gap between LSTM and Transformer models. We apply a standard mogrifier with 5 rounds of updates in our experiments. We expect the mogrifier decoder to profit from IPA segmentation because in this setup the decoder mogrifies neighboring IPA phoneme segments and not space and IPA characters.

Enriching the decoder input with the currently attended input character: The autoregressive decoder of the baseline system uses the LSTM decoder output of the previous time step and the BiLSTM encoded representation of the currently attended input character as input. Intuitively, by feeding the input character embedding directly into the decoder (as a kind of skip connection), we want to liberate the BiLSTM encoder from transporting the hard attention information to the decoder, thereby motivating the sequence encoder to focus more on the contextualization of the input character.

Multiple parallel BiLSTM encoders: Convolutional encoders typically use many convolutional filters for representation learning and Transformer encoders similarly feature multi-head attention. Using several LSTM encoders in parallel has been proposed by Zhu et al. (2017) for language modeling and translation and was e.g. also successfully used for named entity recognition (Žukov-Gregorič et al., 2018). Technically, the same input is fed though several smaller LSTMs, each with its own parameter set, and then their output is concatenated for each time step. The idea behind parallel LSTM encoders is to provide a more robust ensemble-style encoding with lower variance between models. For our submission, there was not enough time to systematically tune the input and hidden state sizes as well as the number of parallel LSTMs.

Adaptive batch size scheduler: We combine the ideas of "Don't Decay the Learning Rate, Increase the Batch Size" (Smith et al., 2017) and cyclical learning schedules by dynamically enlarging or reducing the batch size according to development set accuracy: Starting with a defined minimal batch size m threshold, the batch size for the next epoch is set to $\lfloor m - 0.5 \rfloor$ if the development set performance improved, or $\lfloor m + 0.5 \rfloor$ otherwise.[5] If a predefined maximum batch size is reached, the batch size is reset in one step to the minimum threshold. The motivation for the reset comes from empirical observations that going back to a small batch size can help overcome local optima. With larger training sets, we subsample the training sets per epoch randomly in order to have a more dynamic behavior.[6]

[4]For all experiments, we start with a probability of 50% for UNKing a character in a word and reduce this rate over 10 epochs to a minimal probability of 1%. Light experimentation on a few languages led to this cautious setting, which might leave room for further improvement.

[5]See also the recent discussion on learning rates and batch sizes by Wu et al. (2021).

[6]The subsample size is set to 3,000 items per epoch in all our experiments.

2.2 Unicode normalization

For some writing systems, e.g. for Korean or Vietnamese, applying Unicode NFD normalization to the input has a great impact on the input sequence length and consequently on the G2P character correspondences. The decomposition of diacritics and other composing characters for all languages, as performed in the baseline, has the disadvantage of longer input sequences. We apply a simple heuristic to decide on NFD normalization based on a criterion for the minimum length distance between graphemes and phonemes: If more than 50% of the training grapheme sequences in NFD normalization have a smaller length difference compared to the phoneme sequence than their corresponding NFC variants, then NFD normalization is applied. See Table 1 for statistics, which indicate a preference for NFD for only 2 languages.

3 Submission details

Modifications such as mogrifier LSTMs, additional input character skip connections, or parallel encoders increase the number of model parameters and make it difficult to compare the baseline system directly with its variants. Additionally, we did not have enough time before the submission to systematically explore and fine-tune for the best combination of model modifications and hyper-parameters. In the end, after some light experimentation we had to stick to settings that might not be optimal.

We train separate models for each language on the official training data and use the development set exclusively for model selection. As beam decoding for mogrifier models sometimes suffered compared to greedy decoding, we built all ensembles from greedy model prediction. Like the baseline system (B), we train the SED model for 10 epochs, use one-layer LSTMs, hidden state dimension 200 for the decoder LSTMs and action embedding dimension 100. For the low (L) and medium (M) setting, we have the following specific hyper-parameters:

- patience: 12 (B), 24 (L), 18 (M)
- maximal epochs: 60 (B), 80 (L/M)
- minimal batch size:[7] 3 (L), 5 (M)
- maximal batch size: 10 (L/M)
- character embedding dimension:[8] 100 (B), 50(L/M)
- LSTM encoder hidden state dimension: 200 (B), 300 (L/M) divided by 6 parallel encoders.

We submit 3 ensemble runs for the low setting:
CLUZH-1: 15 models with CHAR input,
CLUZH-2: 15 models with SEG input,
CLUZH-3: 30 models with CHAR or SEG input.
We submit 4 ensemble runs for the medium setting:
CLUZH-4: 5 models with CHAR input,
CLUZH-5: 10 models with SEG input,
CLUZH-6: 5 models with SEG input,
CLUZH-7: 15 models with CHAR or SEG input.
Due to a configuration error, medium results were actually computed without two add-ons: mogrifier LSTMs and the additional input character. In post-submission experiments, we computed runs that enabled these features and report their results as well (CLUZH-4m/5m).

4 Results and discussion

Table 2 shows a comparison of results for the low setting. We report the development and test set average word error rate (WER) performance to illustrate the sometimes dramatic differences between these sets (e.g. Greek). Both runs containing CHAR action emitting models (CLUZH-1, CLUZH-3) have second best results (the best system reaches 24.1). The SEG models with IPA segmentation actions excel on some languages (Adyghe, Latvian), but fail badly on Slovene and Maltese. Only for Romanian and Italian, we see an improvement for the 30-strong mixed ensemble. The expectation that the size difference between the SEG and CHAR vocabulary correlates with language-specific performance differences cannot be confirmed given the numbers in Table 1. E.g. Latvian features 73 different IPA segments but only 51 IPA characters, still, the SEG variant shows only 49% WER.

Table 3 shows a comparison of results for the medium setting. We report selected development and test set average performance to illustrate that also in this larger setting, the expectation of a slightly higher development set performance does not always hold (e.g. Korean or Japanese). On the other hand, Bulgarian and Dutch have a sharp increase in errors on the test set compared to the development set. The comparison between runs with the mogrifier LSTM decoder and the attended character input (CLUZH-*N*m) or without (C-*N*) suggest that these changes are not beneficial. In the medium setting, C-4 (CHAR) and C-6 (SEG) can be directly

[7] The baseline system's batch size is 5.

[8] The motivation for lowering the character embedding size comes from adding the input character to the mogrifier decoder LSTM, which increases the parameter size for each of the 5 update weight matrices.

	CLUZH-1 (CHAR)				CLUZH-2 (SEG)				C-3	OUR BASELINE				BSL	Other
	AVERAGE			E	AVERAGE			E	E	AVERAGE			E	E	
LNG	dev	test	sd	test	dev	test	sd	test	test	dev	test	sd	test	test	test
ady	25.0	27.8	3.3	24	25.6	26.2	1.8	**22**	**22**	26	25.2	2.8	21	**22**	**22**
gre	6.5	22.2	2.3	**20**	5.1	22.8	2.8	22	**20**	5	26.0	3.3	25	21	21
ice	14.8	12.4	2.4	**10**	16.1	14.5	2.2	12	**10**	21	15.8	2.1	12	12	11
ita	24.5	27.0	2.2	23	24.4	26.3	3.2	24	21	25	22.7	3.5	19	**19**	20
khm	39.8	38.2	3.4	32	40.3	36.9	2.2	33	32	39	40.4	2.5	34	34	**28**
lav	47.2	53.7	2.8	53	46.9	55.3	3.7	**49**	**49**	44	56.5	2.2	54	55	**49**
mlt	17.0	18.0	2.4	**12**	19.7	21.2	2.9	16	14	23	21.8	5.1	17	19	18
rum	11.1	13.7	1.8	13	10.3	14.1	1.0	13	12	11	12.5	2.1	10	**10**	**10**
slv	46.4	56.4	2.7	50	48	60.2	3.4	59	55	44	54.2	2.1	51	49	**47**
wel	18.0	14.9	3.5	**10**	15.6	15.7	1.8	13	12	19	14.8	2.0	12	**10**	12
AVG	25.0	28.4	2.7	24.7	25.2	29.3	2.5	26.3	24.7	25.7	29.0	2.8	25.5	25.1	23.8

Table 2: Overview of the dev and test results in the low setting. C-3 is CLUZH-3 ensemble. OUR BASELINE shows the results for our own run of the baseline configuration. They are different from the official baseline results (BSL) due to different random seeds. Column sd always reports the test set standard deviation. E means ensemble results.

	C-4	CLUZH-4m (CHAR)			C-5	CLUZH-5m (SEG)				C-5l	C-6	C-7	OUR BASELINE				BSL
	E_5	AVERAGE			E_5	E_{10}	AVERAGE			E_{10}	E_{10}	E_5	E_{15}	AVERAGE			E_{10}	E_{10}
LNG	test	dev	test	sd	test	test	dev	test	sd	test	test	test	test	dev	test	sd	test	test
arm	7.1	5.4	7.9	0.7	6.4	6.6	5.1	7.2	0.5	**6.2**	7.1	6.6	**6.4**	5.8	7.8	0.7	6.5	7.0
bul	20.1	12.2	20.4	2.0	19.9	19.2	11.9	23.3	2.1	22.4	16.2	18.8	19.7	12.5	19.7	1.7	19.3	**18.3**
dut	15.0	13.1	18.3	1.2	14.8	14.9	12.4	16.8	0.6	14.6	14.5	15.6	**14.7**	13.1	17.7	1.3	**14.3**	**14.7**
fre	**7.5**	8.4	9.7	0.6	8.2	**7.5**	8.5	9.5	0.7	8.1	8.1	**7.5**	7.6	8.9	9.1	0.5	7.8	8.5
geo	**.0**	.0	.0	.0	.0	**.0**	.0	.0	.0	.0	.0	**.0**	**.0**	.0	.0	.0	.0	**.0**
hbs	38.4	43.2	44.5	1.1	39.1	35.6	42.4	44.3	1.5	36.8	35.7	37.0	35.3	39.1	38.9	1.2	33.6	**32.1**
hun	1.5	1.8	1.8	0.1	1.6	1.2	1.7	1.5	0.3	1.0	**0.9**	**1.0**	**1.0**	1.7	2.0	0.3	1.8	1.8
jpn	5.9	6.9	6.8	0.2	5.5	5.3	6.8	6.5	0.3	5.4	5.2	5.5	**5.0**	6.8	6.4	0.5	5.5	5.2
kor	**16.2**	21.3	18.6	0.7	17.4	16.9	19.6	18.3	0.8	16.2	**16.1**	17.2	16.3	20.4	18.9	0.8	16.5	16.3
vie	2.3	1.2	2.4	0.1	2.3	**2.0**	1.2	2.1	0.1	2.1	2.2	2.1	**2.0**	1.4	2.5	0.2	2.4	2.5
AVG	11.4	11.4	13.0	0.7	11.5	10.9	11.0	12.9	0.7	11.3	**10.6**	11.1	10.8	11.0	12.3	0.7	10.8	**10.6**

Table 3: Overview of the development and test results in the medium setting. C-*N* is CLUZH-*N* ensemble. CLUZH-*N*m runs use the mogrifier decoder and additional input character in decoder (these are post-submisson runs). C-5l uses larger parameterization and reaches WER 10.60 (BSL: 10.64). OUR BASELINE shows the results for our own run of the baseline configuration. Boldface indicates best performance in official shared task runs; underline marks the best performance in post-submission configurations. Column sd always reports the test set standard deviation. E_n means n-strong ensemble results.

compared because they feature the same ensemble size: The results suggest that IPA segmentation (SEG) for higher resource settings (and the specific medium languages) seems to be slightly better than CHAR. C-5l is a post-submission run with a larger parametrization.[9] This post-submission ensemble outperforms the baseline system by a small margin, but still struggles with Serbo-Croatian (hbs) compared to the official baseline results.

In a post-submission experiment on the high setting, we built a large[10] 5-strong SEG-based ensemble. It achieves an impressive low word error rate of 38.7 compared to the official baseline (41.94) and the best other submission (37.43).

Future work: Performance variance between different runs of our LSTM-based architecture makes it difficult to reliably assess the actual usefulness of the small architectural changes; extensive experimentation, e.g. in the spirit of Reimers and Gurevych (2017), is needed for that. One should also investigate the impact of the official data set splits: The observed differences between the development set and test set performance in the low

[9]Three parallel encoders with 200 hidden units each; character embedding dimension of 200; no mogrifier; no input character added to the decoder.

[10]Character embedding dimension: 200; action embedding dimension: 100; 10 parallel encoders with hidden state dimension 100; decoder hidden state dimension: 500; minimal batch size: 5; maximal batch size: 20; epochs: 200 (subsampled to 3,000 items); patience: 24; no mogrifier; no input character added to the decoder.

setting for Slovene or Greek are extreme. Cross-validation experiments might help assess the true difficulty of the WikiPron datasets.

5 Conclusion

This paper presents the approach taken by the CLUZH team to solving the SIGMORPHON 2021 Multilingual Grapheme-to-Phoneme Conversion challenge. Our submission for the low and medium settings is based on our successful SIGMORPHON 2020 system, which is a majority-vote ensemble of neural transducers trained with imitation learning. We add several modifications to the existing LSTM architecture and experiment with IPA segment vs. IPA character action predictions. For the low setting languages, our IPA character-based run outperforms the baseline and ranks second overall. The average performance of segment-based action edits suffers from performance outliers for certain languages. For the medium setting languages, we note small improvements on some languages, but the overall performance is lower than the baseline. Using a mogrifier LSTM decoder and enriching the encoder input with the currently attended input character did not improve performance in the medium setting. Post-submission experiments suggest that network capacity for the submitted systems was too small. A post-submission run for the high-setting shows considerable improvement over the baseline.

References

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

Lucas F.E Ashby, Travis M. Bartley, Simon Clematide, Luca Del Signore, Cameron Gibson, Kyle Gorman, Yeonju Lee-Sikka, Peter Makarov, Aidan Malanoski, Sean Miller, Omar Ortiz, Reuben Raff, Arundhati Sengupta, Bora Seo, Yulia Spektor, and Winnie Yan. 2021. Results of the Second SIGMORPHON 2021 Shared Task on Multilingual Grapheme-to-Phoneme Conversion. In *Proceedings of 18th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*.

Kai-Wei Chang, Akshay Krishnamurthy, Hal Daumé, III, and John Langford. 2015. Learning to search better than your teacher. In *PMLR*, volume 37 of *Proceedings of Machine Learning Research*.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning*, 75(3).

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

Jackson L. Lee, Lucas F.E. Ashby, M. Elizabeth Garza, Yeonju Lee-Sikka, Sean Miller, Alan Wong, Arya D. McCarthy, and Kyle Gorman. 2020. Massively multilingual pronunciation modeling with WikiPron. In *LREC*.

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

Peter Makarov and Simon Clematide. 2020. CLUZH at 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*.

Gábor Melis, Tomás Kociský, and Phil Blunsom. 2019. Mogrifier LSTM. *CoRR*, abs/1909.01792.

Nils Reimers and Iryna Gurevych. 2017. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. In *EMNLP*.

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

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *PMLR*, volume 15 of *Proceedings of Machine Learning Research*.

Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. 2017. Don't decay the learning rate, increase the batch size. *CoRR*, abs/1711.00489.

Shijie Wu, Ryan Cotterell, and Mans Hulden. 2021. Applying the transformer to character-level transduction. In *EACL*.

Danhao Zhu, Si Shen, Xin-Yu Dai, and Jiajun Chen. 2017. Going wider: Recurrent neural network with parallel cells. *CoRR*, abs/1705.01346.

Andrej Žukov-Gregorič, Yoram Bachrach, and Sam Coope. 2018. Named entity recognition with parallel recurrent neural networks. In *ACL*.