CLUZH at SIGMORPHON 2022 Shared Tasks on Morpheme Segmentation and Inflection Generation

Silvan Wehrli **Simon Clematide** **Peter Makarov**
Department of Computational Linguistics
University of Zurich, Switzerland
silvan.wehrli@uzh.ch {simon.clematide,makarov}@cl.uzh.ch

Abstract

This paper describes the submissions of the team of the Department of Computational Linguistics, University of Zurich, to the SIGMORPHON 2022 Shared Tasks on Morpheme Segmentation and Inflection Generation. Our submissions use a character-level neural transducer that operates over traditional edit actions. While this model has been found particularly well-suited for low-resource settings, using it with large data quantities has been difficult. Existing implementations could not fully profit from GPU acceleration and did not efficiently implement mini-batch training, which could be tricky for a transition-based system. For this year's submission, we have ported the neural transducer to PyTorch and implemented true mini-batch training. This has allowed us to successfully scale the approach to large data quantities and conduct extensive experimentation. We report competitive results for morpheme segmentation (including sharing first place in part 2 of the challenge). We also demonstrate that reducing sentence-level morpheme segmentation to a word-level problem is a simple yet effective strategy. Additionally, we report strong results in inflection generation (the overall best result for large training sets in part 1, the best results in low-resource learning trajectories in part 2). Our code is publicly available.

1 Introduction

This paper describes our submissions to the following SIGMORPHON 2022 shared tasks:

SEGM Morpheme Segmentation (Batsuren et al., 2022):[1]

1. Word-level morpheme segmentation
2. Sentence-level morpheme segmentation

INFL Typologically Diverse and Acquisition-Inspired Morphological Inflection Generation:[2]

[1] https://github.com/sigmorphon/2022SegmentationST
[2] https://github.com/sigmorphon/2022InflectionST

Task	Input	Output
SEGM	hierarchisms	hierarch @@y @@ism @@s
INFL	sue V;PST	sued

Table 1: Examples of morpheme segmentation (SEGM) and inflection generation (INFL). SEGM involves predicting canonical forms of morphemes. The inputs for INFL consist of lemmas and UniMorph feature specifications.

1. Typologically diverse morphological inflection (Kodner et al., 2022)
2. Morphological acquisition trajectories (Kodner and Khalifa, 2022)

All our submissions rely on the same neural hard-attention transducer architecture that has shown strong language-independent performance in a variety of character-level transduction tasks in morphology, grapheme-to-phoneme conversion, and text normalization (Makarov and Clematide, 2018, 2020a,b).

1.1 Morpheme Segmentation

The goal of this task is to design a system that splits words into morphemes (Table 1). Part 1 focuses on word-level morpheme segmentation (inputs are word types), part 2 on sentence-level morpheme segmentation (inputs are tokenized sentences). In part 1, there is a unique segmentation for every input word. This track provides very large datasets (in hundreds of thousands of training examples per language), allowing us to test the scalability of our system. In part 2, a word form may be segmented differently depending on the context. It offers an interesting setup to study, on the example of three languages (English, Czech, Mongolian), how important it is for a system to recognize and correctly handle this ambiguity. Our submission for part 2 tests this by using a word-level model (developed for part 1), optionally with part-of-speech (POS) tags as side input.

19th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, pages 212 - 219

1.2 Inflection Generation

The SIGMORPHON–UniMorph 2022 shared task on typologically diverse and acquisition-inspired morphological inflection generation asks to predict an inflected word form given its lemma and a set of morphosyntactic features specified according to the UniMorph standard (Table 1). Part 1 consists of 32 languages with **small** training sets (mostly 700 items, but for 4 languages only 70 to 240 items) and 21 **large** training sets (exactly 7,000 items). Part 2 has an ablation-style setup for Arabic, English, and German: For each language, there is a dataset for each increment of 100, ranging from 100 to 600 (German) or 1,000 training samples (Arabic, English). The development set feature specifications are representative of the test set. Both tasks target the generalization capabilities of morphology learning systems by examining separately their test set performance on seen and unseen lemmas and feature specifications.

	training			**greedy decoding**	
	BL	**CLUZH**		**CLUZH**	
Batch size	GA	CPU	GPU	CPU	GPU
1	27.49	18.96	**5.02**	**6.49**	10.00
32	23.58	7.48	**0.25**	2.92	**0.73**
64	23.89	7.46	**0.16**	2.84	**0.47**
128	24.69	7.88	**0.13**	2.88	**0.33**
256	27.14	8.21	**0.12**	3.01	**0.26**
512	31.11	8.51	**0.12**	3.26	**0.23**

Table 2: Mini-batch training and greedy decoding speed for this year's implementation (*CLUZH*) vs the baseline (*BL*) of Makarov and Clematide (2020a) on the Armenian dataset of the SIGMOPRHON 2021 shared task on grapheme-to-phoneme conversion (Ashby et al., 2021). The BL models are trained on CPU using gradient accumulation (*GA*). All numbers are given in seconds and per 1,000 samples. The training times are averages of 20 epochs on the training set. The greedy decoding times are averages of 20 runs on the development set using a well-trained model. The CLUZH model hyper-parameters are identical to those of Makarov and Clematide (2020a).

2 Model Description

As a basis for all our submissions, we use a neural character-level transducer that edits the input string into the output string by a sequence of traditional edit actions: substitutions, insertions, deletion, and copy. The specific version of this approach was developed for grapheme-to-phoneme conversion (Makarov and Clematide, 2020a). Such neural transducers have typically performed well in morphological and related character-level transduction tasks in low to medium training data settings. Although they can be competitive in large-data regimes (Makarov and Clematide, 2018), their successful application to large data settings with appropriately large parameter sizes (cf. the Transformer-based models of Wu et al. (2021) have over 7M parameters) may also be limited by a specific implementation. In this year's submission, we scale the approach to large datasets by porting it to a different framework and making algorithmic improvements to training.

True mini-batch training. The training procedure for transition-based systems could be difficult to batch (Noji and Oseki, 2021; Ding and Koehn, 2019), which is why many systems are trained by gradient accumulation over individual samples (and possibly relying on library optimizations such as DyNet Autobatch (Neubig et al., 2017b)). This results in slow training for large data sets. In our implementation of true mini-batch training, we start by precomputing gold action sequences using an oracle character aligner. By doing so, alignments and gold actions for all decoding steps of all training samples are known a priori (as opposed to being computed on the fly, which would be useful when parameter updates are interleaved with sampling from the model distribution). This permits calling the unrolled version of the decoder. The resulting procedure dramatically speeds up training compared to gradient accumulation. Furthermore, our implementation supports batched greedy decoding. Table 2 gives an impression of these performance improvements: For a batch size of 32, training is around 3 times faster on a CPU and close to 100 times faster on a GPU. For a batch size of 512, training is faster by a factor of over 250 on a GPU. Additionally, the time needed for greedy decoding can be efficiently decreased on a GPU.[3]

Further model details. The latest implementation only uses teacher forcing. Specifically, it does not yet incorporate *roll-ins*, i.e. the model does not see its own predictions during training, which would improve generalizability by countering exposure bias (Pomerleau, 1989). We also add support

[3]Note that the precomputation of gold action sequences for the training data takes around 12 seconds per 1000 samples. However, this procedure is only required once per dataset as the precomputed output can be reused for any training run. In any case, the gains shown in Table 2 easily offset the additionally required time.

for features. Features are treated as atomic. For INFL, the features associated with an inflection input-output pair are passed through an embedding layer and then summed. For further details on the system and the oracle character aligner, we refer the reader to Makarov and Clematide (2020a).

3 Submission Details

For both tasks, we train separate models for each language and use the development set exclusively for model selection.

3.1 Morpheme Segmentation

Data preprocessing. Besides NFD normalization as a preprocessing step, we substitute the multi-character morpheme delimiter (" @@") by a single character unseen in the data to decrease the length of the output.

Sentence-level segmentation. We simplify part 2 of the SEGM task by reducing it to a word-level problem. Concretely, we split the input sentences into single word tokens and train the model on these word tokens, similarly to part 1. The single word predictions are then simply concatenated to form the original sentence. Since this completely neglects the context of the words, we have also experimented with POS tags as additional input features (Table 3). We use TreeTagger (Schmid, 1999) to obtain the features.[4] We also experimented with transducing entire sentences in one go, however this led to a substantial drop in accuracy.

Hyper-parameter search. For both parts, we have evaluated extensively various choices of optimizers, learning rate schedulers, batch size, encoder dropout. We found the Adam optimizer (Kingma and Ba, 2015) to work well, as well as the scheduler that reduces the learning rate whenever a development set metric plateaus. We settled on a batch size of 32 for all models, which offers a good trade-off between model performance and training speed.

Encoders. We use a 2-layer stacked LSTM as the encoder and experimented with encoder dropout. We also experimented extensively with a Transformer encoder (Vaswani et al., 2017). Despite considerable effort, we failed to make it work at the performance level of stacked LSTMs. Other hyperparameters (e.g. various embedding dimensions) are similar to the previous work (Makarov and Clematide, 2020a).

Decoding. For efficiency, we compute all the model outputs using mini-batch greedy decoding.

Ensembling. All our submissions are majority-vote ensembles. For part 1, we submit a 5-strong ensemble, **CLUZH**, composed of 3 models without encoder dropout and 2 models with encoder dropout of 0.1.[5]

For part 2, we submit three ensembles. All individual models have an encoder dropout probability of 0.25 and vary only in their use of features: **CLUZH-1** with 3 models without POS features, **CLUZH-2** with 3 models with POS tag features, and **CLUZH-3** with combines all the models from **CLUZH-1** and **CLUZH-2**.

3.2 Inflection Generation

Data preprocessing. For both parts, we apply NFD normalization to the input and split the UniMorph features at ";" by default. For languages that showed lower performance compared to the neural or non-neural baseline on the development set in part 1, we also computed models without NFD normalization and chose the best based on their development set performance. For Korean, we observed some Latin transliteration noise in the train/development set targets, which we removed before training. For Lamaholot (slp), we observed a very low accuracy (5%) on the development set compared to the neural baseline's 20% performance. By splitting UniMorph features at "+"

Гэрт	**эмээ**	хоол	хийв	.
Гэр @@т	**эмээ**	хоол	хийх @@в	.
NN	**NN**	VB	VB	.
Grandmother cooked at home.				
Би	өдөр	**эмээ**	уусан	.
Би	өдөр	**эм @@ээ**	уух @@сан	.
PR	NN	**VB**	VB	.
Today I took my medicine.				

Table 3: SEGM part 2 with POS features for Mongolian. The features inferred from the context using TreeTagger could help disambiguate the word form in bold.

[4] The parameter files are available at https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/.

[5] Due to a mistake, the predictions by the models with dropout 0.1 were included twice, and a prepared model with dropout 0.25 was not used at all. However, the F1 macro-average over all the languages for the intended ensemble on the development set is only 0.08 points higher.

as well as ";",[6] we achieved better generalization for this low-resource language (only 240 training examples available).

Hyper-parameters. For small datasets in both parts: batch size 1, a patience of 30 epochs, one-layer encoder and decoder with hidden size 200, character and action embeddings of size 100, feature embeddings of size 50, the AdamW optimizer (Loshchilov and Hutter, 2019) with a learning rate of 0.0005 (half of the default value), the reduce-learning-rate-on-plateau scheduler with factor 0.75, and beam decoding with beam width 4. For a few languages whose development set performance was lower than that of the baselines, we computed models without NFD normalization and used those in case of improved accuracy.[7]

For large datasets in part 1, we made the following changes from the above: batch size 32, a patience of 20 epochs, action embeddings of size 200, a two-layer encoder with a hidden size of 1,000, a one-layer decoder with a hidden size of 2,000. In case of the development set performance was below that of any of the official baselines, we used some alternative hyper-parameters:[8] no NFD normalization, batch size 16, a one-layer encoder with a hidden size of 2,000, a one-layer decoder with a hidden size of 4,000, and the Adadelta optimizer (Zeiler, 2012) with the default learning rate. Hyper-parameters were not chosen using a systematic grid search or experimentation.

Convergence. For the small datasets in part 1 with default hyper-parameters and NFD normalization, we observe large differences in the number of epochs to convergence (mean 27.3, SD 22.8). For some languages, e.g. Chukchi (ckt), Ket (ket), and Ludian (lud), we see the best results on the first epoch, which typically means the model has just learned to copy the input to the output. For other languages, much larger or highly varying numbers of epochs to convergence are observed: Slovak (15-93), Karelian (13-88), Mongolian, Khalkha (19-61), and Korean (12-143).

For the large datasets in part 1 (7,000 training examples) with default hyper-parameters and NFD normalization, we observe a mean of 17.3 epochs to convergence (SD 16.0). For Ludian, even in the large setting, the first epoch with copying gave the best results. In contrast, Georgian could generally profit from more epochs (mean 36.8, SD 17.9).

Ensembling. Our submission for part 1 is a 5-strong majority-voting ensemble, and it is a 10-strong ensemble for part 2.

4 Results and Discussion

4.1 Morpheme Segmentation

Table 4 and Table 5 show our results for parts 1 and 2, respectively. Based on the macro-average F1 score over all languages, our submission for part 1 ranks third out of 7 full submissions. For part 2, our submission CLUZH-3 was declared the winner out of 10 full submissions.[9]

Dropout. The results for part 1 suggest that encoder dropout can help improve model performance. For some languages, the performance can improve by as much as 1% F1 score absolute.

Ensembling. Ensembling brings a clear improvement over single-best results. On average, the improvement is +0.55% on the development set and +0.53% on the test set (compared to the best single model result). The improvement on the English dataset is substantial: +1.64% and +1.84% on the development and test sets, respectively.

Gains from POS tags. The results for part 2 suggest that treating a sentence-level problem as word-level may be a simple yet powerful strategy for morpheme segmentation. The success of this strategy depends on the language and the data. The more segmentation ambiguity a language has, the more important the context is. Mongolian has the highest segmentation ambiguity (Table 6). Around $1/5$ of the tokens in the training data have at least two possible segmentations, whereas Czech and English exhibit little to no ambiguity. This may partially explain why the performance on the Mongolian data is the lowest. This also explains why using POS tags as additional features bring the biggest improvement for Mongolian: +0.29% and +0.27% on the development and test sets, based on the average of individual models. Using POS tags improves the prediction of ambiguous segmentation by an absolute 1.1% and 0.6% on the development and

[6] For instance, `V;ARGAC2P+ARGNO2P;SBJV` would be split into 4 separate features.

[7] Arabic, Gothic, Hungarian, and Old Norse.

[8] Arabic, Assamese, Evenki, Hungarian, Kazakh, Mongolian, Khalkha, and Old Norse.

[9] Our submission performed the best on two out of three languages (Czech and Mongolian). As it was beaten by another submission based on the macro F1 average, two submissions were declared winners.

Language	dropout = 0.0 (avg. of 3 models) dev	dropout = 0.0 (avg. of 3 models) test	dropout = 0.1 (1 model) dev	dropout = 0.1 (1 model) test	dropout = 0.25 (1 model) dev	dropout = 0.25 (1 model) test	ensemble (5 models) dev	ensemble (5 models) test	best other test
Czech	92.96	93.31	93.35	93.60	93.32	93.49	94.07	93.81	**93.88**
English	90.33	90.33	91.01	90.86	90.91	90.68	92.65	92.70	**93.63**
French	93.22	93.02	93.95	93.85	93.72	93.48	94.94	94.80	**95.73**
Hungarian	99.40	98.28	99.15	98.09	99.63	98.57	99.61	98.54	**98.72**
Spanish	97.79	97.78	98.57	98.61	98.53	98.56	98.71	98.74	**99.04**
Italian	95.54	95.54	96.15	96.19	96.02	96.11	96.93	96.93	**97.47**
Latin	99.20	99.20	99.30	99.26	99.30	99.23	99.40	99.37	**99.38**
Russian	97.52	97.54	96.38	96.43	96.65	96.54	98.58	98.62	**99.35**
Mongolian	98.21	97.73	98.47	97.80	98.47	97.90	98.53	98.12	**98.51**
AVG	96.02	95.86	96.26	96.08	96.28	96.06	97.05	96.85	**97.30**

Table 4: F1 scores for SEGM part 1.

Language	without features: average (3 models) dev	without features: average (3 models) test	without features: ensemble (3 models) dev	without features: ensemble (3 models) test	with POS tags: average (3 models) dev	with POS tags: average (3 models) test	with POS tags: ensemble (3 models) dev	with POS tags: ensemble (3 models) test	combined: ensemble (6 models) dev	combined: ensemble (6 models) test	best other test
Czech	94.06	90.90	94.54	91.35	94.15	91.15	94.45	91.76	94.72	**91.99**	91.76
English	98.12	89.27	98.31	89.47	98.18	89.29	98.38	89.47	98.41	89.54	**96.31**
Mongolian	85.95	81.57	87.06	82.22	86.24	81.84	87.26	82.55	87.62	**82.88**	82.59
AVG	92.71	87.25	93.30	87.68	92.86	87.43	93.36	87.93	93.58	88.14	**90.22**

Table 5: F1 scores for SEGM part 2. All models are trained with a dropout probability of 0.25.

Language	train 1	train ≥2	dev 1	dev ≥2
Czech	100%	0%	100%	0%
English	99.58%	0.42%	99.75%	0.25%
Mongolian	77.91%	22.09%	90.00%	10.00%

Table 6: Segmentation ambiguity in SEGM part 2: Relative frequency of unambiguous (1) vs ambiguous (≥ 2) word tokens.

dev: ambiguous NF	dev: ambiguous POS	dev: all Δ	test: ambiguous NF	test: ambiguous POS	test: all Δ
63.0%	64.1%	+0.11%	59.5%	60.1%	+0.06%

Table 7: Impact of POS features on Mongolian, SEGM part 2. *ambiguous* shows the average percentage of correctly predicted ambiguous segmentations for Mongolian. *NF* denotes models without features, *POS* denotes models using POS tags. *all* shows the absolute improvement for POS compared to NF, in relation to the whole dataset.

test sets for Mongolian (Table 7). When looking at the whole dataset, using POS features increases the relative number of correct predictions by 0.11% (development set) and 0.06% (test set) compared to not using the features. Using POS tags brings slight improvements and helps mitigate the loss of context.

PyTorch reimplementation. This year's system is a close reimplementation in PyTorch (Paszke et al., 2019) of our earlier CPU codebase using DyNet (Neubig et al., 2017a). It fully supports GPU utilization, allowing for efficient processing of large amounts of training data. Our code is publicly available.[10]

[10] https://github.com/slvnwhrl/il-reimplementation

Token–type ratio. Another reason for the lower performance of Mongolian might lie in the high variance in the data: The Mongolian training dataset contains around 40% unique tokens (Table 8). This is around 4 times more than in the

Language	train total	train unique	dev total	dev unique
Czech	15,157	5,126	7,545	3,217
English	169,117	17,249	21,444	4,849
Mongolian	13,237	5,293	6,632	3,216

Table 8: Word counts in SEGM part 2: The total number of word forms and the number of unique words.

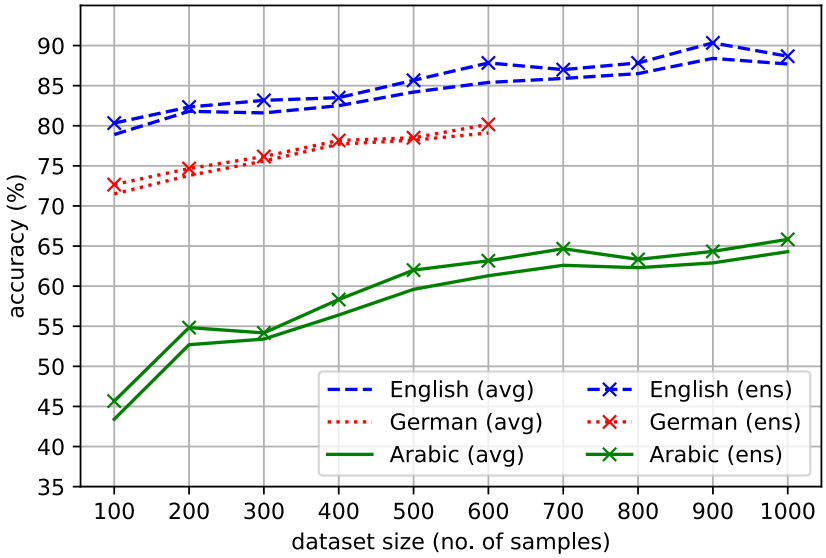

Figure 1: Test accuracy results for INFL part 2. avg=average, ens=10-strong ensemble.

System	Overall	**seen status** (± **L**emma/**F**eatures) +L +F	+L –F	-L +F	-L –F
Small dataset setting					
CLUZH	56.87	77.31	31.27	**77.97**	43.26
Best	**74.76**	**81.64**	72.91	**77.97**	70.87
Δ	-17.89	-4.33	-41.64	0.00	-27.62
Large dataset setting					
CLUZH	**67.85**	**90.99**	41.43	**87.17**	**60.30**
Best	62.39	89.57	**42.17**	85.31	55.56
Δ	5.46	1.43	-0.74	1.86	4.74

Table 9: Test results (accuracy macro-averaged over languages) for INFL part 1 split by training dataset size: large (7,000 training examples) vs small (up to 700 examples). Δ shows the difference between our submission and the best competitor covering the full set of languages.

English dataset. This makes the learning problem much harder, which is further exacerbated by the relatively small size of the data (compared to English).

4.2 Inflection Generation

The part 1 test set results are shown in Table 9. Given the large number of languages, we discuss the average accuracy on small and large training sets. An important goal for this shared task was to assess a system's performance on test data subsets defined by whether both the lemma and the feature specification were seen in the training data (+L +F in the Table), whether only the lemma (+L, -F), or only the feature specification (-L, +F) were seen, or whether neither of them (-L -F) appeared in the training data.

Small datasets. On the small datasets, our system only excels on the -L +F subset, meaning it is strong in modeling the behaviour of features. In the small dataset setting, the best competitor system, UBC, has an extremely strong performance in case the lemma is known (+L). It would be interesting to know what kind of information or data augmentation UBC uses: The neural baseline, which utilizes data augmentation, has a much lower performance (24.9%) than our submission. Overall, our submission with a 5-strong ensemble achieves the second-best result of the submissions covering all languages.

Large datasets. In the large dataset setting, our submission shows the best performance overall. On the subset with seen lemmas and unseen features (+L -F), the neural baseline is the only system with slightly better results. This indicates that our system's modeling of lemmas is not yet optimal. The information flow in our architecture maybe dominated by the features (they are fed into the decoder at every action prediction step) and the aligned input character, and it may not have the best representation of the input lemma as a whole.

Trajectories. The test set results for part 2 are shown in Figure 1. Our 10-strong ensemble was

the clear overall winner in this low-resource track. It beats the best competing approaches by a substantial margin on the per-language average: Arabic 59.6% accuracy (best competitor OSU 57.5%), German 76.7% (non-neural baseline 74.8%), English 85.7% (OSU 81.5%).

Individual model performance varies, and the majority-vote ensembling improved the scores by 1.4% absolute on average on the test set. Interestingly, the difference between the average model performance and the ensemble performance does not get smaller with larger training sets.

The correlation between the increasing number of training examples and the improving test set performance is almost perfect for the average performance. Ensembles are slightly less stable.

5 Conclusion

This paper presents the submissions of the Department of Computational Linguistics, University of Zurich, to the SIGMOPRHON 2022 morpheme segmentation and inflection generation shared tasks. We build on the previous architecture, the neural transducer over edit actions, porting it to a new deep learning framework and implementing GPU-optimized mini-batch training. This permits scaling the system to large training datasets, as demonstrated by strong performance in both shared tasks.

We show that reducing sentence-level morpheme segmentation to a word-level problem is a viable strategy. Conditioning on POS tags brings further improvements. We leave it to future work to explore more powerful representations of context. We experimented with a Transformer-based encoder for morpheme segmentation, and while the initial results were not satisfactory, we intent to pursue this further. In inflection generation, we note problems with capturing unseen lemmas, despite otherwise strong performance across data regimes.

References

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 Shared Task on Multilingual Grapheme-to-Phoneme Conversion. In *Proceedings of the 18th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*.

Khuyagbaatar Batsuren, Gábor Bella, Aryaman Arora, Viktor Martinović, Kyle Gorman, Zdeněk Žabokrtský, Amarsanaa Ganbold, Šárka Dohnalová, Magda Ševčíková, Kateřina Pelegrinová, Fausto Giunchiglia, Ryan Cotterell, and Ekaterina Vylomova. 2022. The sigmorphon 2022 shared task on morpheme segmentation. In *19th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*.

Shuoyang Ding and Philipp Koehn. 2019. Parallelizable stack long short-term memory. In *Proceedings of the Third Workshop on Structured Prediction for NLP*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Jordan Kodner and Salam Khalifa. 2022. SIGMORPHON-UniMorph 2022 Shared Task 0: Modeling Inflection in Language Acquisition. In *Proceedings of the SIGMORPHON 2022 Shared Task: Morphological Inflection*.

Jordan Kodner, Salam Khalifa, Khuyagbaatar Batsuren, Hossep Dolatian, Ryan Cotterell, Faruk Akkuş, Antonios Anastasopoulos, Taras Andrushko, Aryaman Arora, Nona Atanelov, Gábor Bella, Elena Budianskaya, Yustinus Ghanggo Ate, Omer Goldman, Simon Guriel, Silvia Guriel-Agiashvili, Jan Hajič, Jan Hric, Ritvan Karahodja, Witold Kieraś, Andrew Krizhanovsky, Natalia Krizhanovsky, Igor Marchenko, Magdalena Markowska, Polina Mashkovtseva, Maria Nepomniashchaya, Daria Rodionova, Elizabeth Salesky, Karina Sheifer, Alexandra Serova, Anastasia Yemelina, Jeremiah Young, and Ekaterina Vylomova. 2022. SIGMORPHON-UniMorph 2022 Shared Task 0: Generalization and Typologically Diverse Morphological Inflection. In *Proceedings of the SIGMORPHON 2022 Shared Task: Morphological Inflection*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Peter Makarov and Simon Clematide. 2018. Imitation learning for neural morphological string transduction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Peter Makarov and Simon Clematide. 2020a. 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*.

Peter Makarov and Simon Clematide. 2020b. Semi-supervised contextual historical text normalization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017a. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*.

Graham Neubig, Yoav Goldberg, and Chris Dyer. 2017b. On-the-fly Operation Batching in Dynamic Computation Graphs. In *Advances in Neural Information Processing Systems*, volume 30.

Hiroshi Noji and Yohei Oseki. 2021. Effective batching for recurrent neural network grammars. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*.

Dean A Pomerleau. 1989. Alvinn: An autonomous land vehicle in a neural network. In *Proceedings of the Conference on Neural Information Processing Systems*.

H. Schmid. 1999. Improvements in Part-of-Speech Tagging with an Application to German. In *Natural Language Processing Using Very Large Corpora*, Text, Speech and Language Technology.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30.

Shijie Wu, Ryan Cotterell, and Mans Hulden. 2021. Applying the transformer to character-level transduction. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*.

Matthew D Zeiler. 2012. ADADELTA: an adaptive learning rate method. arXiv:1212.5701.