Training Strategies for Neural Multilingual Morphological Inflection

Adam Ek and Jean-Philippe Bernardy
Centre for Linguistic Theory and Studies in Probability
Department of Philosophy, Linguistics and Theory of Science
University of Gothenburg
{adam.ek,jean-philippe.bernardy}@gu.se

Abstract

This paper presents the submission of team GUCLASP to SIGMORPHON 2021 Shared Task on Generalization in Morphological Inflection Generation. We develop a multilingual model for Morphological Inflection and primarily focus on improving the model by using various training strategies to improve accuracy and generalization across languages.

1 Introduction

Morphological inflection is the task of transforming a *lemma* to its *inflected form* given a set of *grammatical features* (such as `tense` or `person`). Different languages have different strategies, or morphological processes such as affixation, circumfixation, or reduplication among others (Haspelmath and Sims, 2013). These are all ways to make a lemma express some grammatical features. One way to characterize how languages express grammatical features is a spectrum of morphological typology ranging from agglutinative to isolating. In agglutinative languages, grammatical features are encoded as bound morphemes attached to the lemma, while in isolating languages each grammatical feature is represented as a lemma. Thus, languages in different parts of this spectrum will have different strategies for expressing information given by grammatical features.

In recent years, statistical and neural models have been performing well on the task of morphological inflection (Smit et al., 2014; Kann and Schütze, 2016; Makarov et al., 2017; Sharma et al., 2018). We follow this tradition and implement a neural multilingual model for morphological inflection. In a multilingual system, a single model is developed to handle input from several different languages: we can give the model either a word in Evenk or Russian and it perform inflection.

This is a challenging problem for several reasons. For many languages resources are scarce, and a multilingual system must balance the training signals from both high-resource and low-resource languages such that the model learns something about both. Additionally, different languages employ different morphological processes to inflect words. In addition to languages employing a variety of different morphological processes, different languages use different scripts (for example Arabic, Latin, or Cyrillic), which can make it hard to transfer knowledge about one language to another. To account for these factors a model must learn to recognize the different morphological processes, the associated grammatical features, the script used, and map them to languages.

In this paper, we investigate how far these issues can be tackled using different training strategies, as opposed to focusing on model design. Of course, in the end, an optimal system will be a combination of a good model design and good training strategies. We employ an LSTM encoder-decoder architecture with attention, based on the architecture of Anastasopoulos and Neubig (2019), as our base model and consider the following training strategies:

- Curriculum learning: We tune the order in which the examples are presented to the model based on the loss.
- Multi-task learning: We predict the formal operations required to transform a lemma into its inflected form.
- Language-wise label smoothing: We smooth the loss function to not penalize the model as much when it predicts a character from the correct language.
- Scheduled sampling: We use a probability distribution to determine whether to use the

previous output or the gold as input when decoding.

2 Data

The data released cover 38 languages of varying typology, genealogy, grammatical features, scripts, and morphological processes. The data for the different languages vary greatly in size, from 138 examples (Ludic) to 100310 (Turkish). For the low-resourced languages[1] we extend the original dataset with *hallucinated data* (Anastasopoulos and Neubig, 2019) to train on.

With respect to the work of Anastasopoulos and Neubig (2019), we make the following changes. We identify all subsequences of length 3 or more that overlap in the lemma and inflection. We then randomly sample one of them, denoted R, as the sequence to be replaced. For each language, we compile a set $\mathcal{C}_{\mathcal{L}}$ containing all (1,2,3,4)-grams in the language. We construct a string G to replace R with by uniformly sampling n-grams from $\mathcal{C}_{\mathcal{L}}$ and concatenating them $G = \mathsf{cat}(g_0, ..., g_m)$ until we have a sequence whose length satisfy: $|R| - 2 \leq |G| \leq |R| + 2$.

Additionally, we do not consider subsequences which include a phonological symbol.[2] A schematic of the hallucination process is shown in Figure 1.

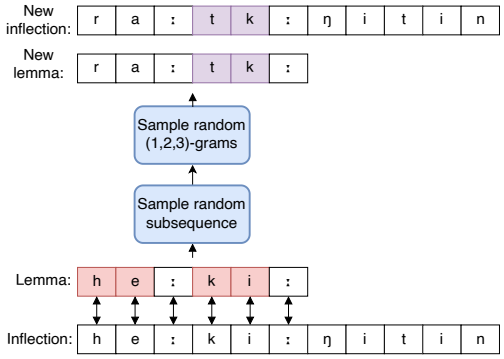

Figure 1: A example of the data hallucination process. The sequence $R =$ ki is replace by $G =$ tk.

Sampling n-grams instead of individual characters allow us to retain some of the orthographical information present in the language. We generate a set of 10 000 hallucinated examples for each of the low-resource languages.

[1]We consider languages with less than 10 000 training examples as low-resource in this paper.

[2]Thus in Figure 1 a subsequence of length 2 is selected as the sequence to be replaced, since the larger subsequences would include the phonological symbol :

3 Method

In this section, the multilingual model and training strategies used are presented. [3] We employ a single model with shared parameters across all languages.

3.1 Model

To account for different languages in our model we prepend a language embedding to the input (similarly to Johnson et al. (2017); Raffel et al. (2019)). To model inflection, we employ an encoder-decoder architecture with attention. The first layer in the model is comprised of an LSTM, which produces a contextual representation for each character in the lemma. We encode the tags using a self-attention module (equivalent to a 1-head transformer layer) (Vaswani et al., 2017). This layer does not use any positional data: indeed the order of the tags does not matter (Anastasopoulos and Neubig, 2019).

To generate inflections, we use an LSTM decoder with two attention modules. One attending to the lemma and one to the tags. For the lemma attention, we use a content-based attention module (Graves et al., 2014; Karunaratne et al., 2021) which uses cosine similarity as its scoring method. However, we found that only using content-based attention causes attention to be too focused on a single character, and mostly ignores contextual cues relevant for the generation.

To remedy this, we combine the content-based attention with additive attention as follows, where superscript cb indicate content-based attention, add additive attention and k the key:

$$a^{add} = \mathsf{softmax}(w^\top \mathsf{tanh}(\mathsf{W}_a k + \mathsf{W}_b h))$$

$$att^{add} = \sum_{t=1}^{T} a_t^{add} h_t^{add}$$

$$a^{cb} = \mathsf{softmax}(cos(k, h))$$

$$att^{cb} = \sum_{t=1}^{T} a_t^{cb} h_t^{cb}$$

$$att = \mathsf{W}[att^{cb}; att^{add}]$$

In addition to combining content-based attention and additive attention we also employ regularization on the attention modules such that for each decoding step, the attention is encouraged to distribute the attention weights a uniformly across

[3]Our code is available here: https://github.com/adamlek/multilingual-morphological-inflection/

the lemma and tag hidden states (Anastasopoulos and Neubig, 2019; Cohn et al., 2016). We employ additive attention for the tags.

In each decoding step, we pass the gold or predicted character embedding to the decoding LSTM. We then take the output as the key and calculate attention over the lemma and tags. This representation is then passed to a two-layer perceptron with ReLU activations.

3.2 Multi-task learning

Instead of predicting the characters in the inflected form, one can also predict the Levenshtein operations needed to transform the lemma into the inflected form; as shown by Makarov et al. (2017).

A benefit of considering operations instead of characters needed to transform a lemma to its inflected form is that the script used is less of a factor, since by considering the operations only we abstract away from the script used. We find that making *both* predictions, as a multi-task setup, improves the performance of the system.

The multi-task setup operates on the character level, thus for each contextual representation of a character we want to predict an operation among *deletion* (`del`), *addition/insertion* (`add`), *substitution* (`sub`) and *copy* (`cp`). Because `add` and `del` change the length, we predict two sets of operations, the **lemma-reductions** and the **lemma-additions**. To illustrate, the Levenshtein operations for the word pair (*valatas*, *ei valate*) in Veps (uralic language related to Finnish) is shown in Figure 2.

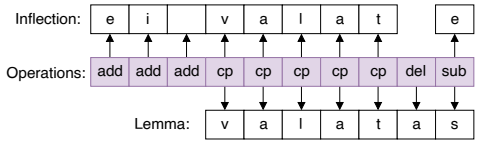

Figure 2: Levenshtein operations mapped to characters in the lemma and inflection.

In our setup, the task of lemma-reductions is performed by predicting the `cp`, `del`, and `sub` operations based on the encoded hidden states in the lemma. The task of lemma-additions then is performed by predicting the `cp`, `add`, and `sub` operations on the characters generated by the decoder. We use a single two-layer perceptron with ReLU activation to predict both lemma-reduction and lemma-additions. [4]

[4] In the future, we'd like to experiment with including the representations of tags in the input to the operation classifier.

3.3 Curriculum Learning

We employ a competence-based curriculum learning strategy (Liu et al., 2020; Platanios et al., 2019). A competence curriculum learning strategy constructs a learning curriculum based on the *competence* of a model, and present examples which the model is deemed to be able to handle. The goal of this strategy is for the model to transfer or apply the information it acquires from the easy examples to the hard examples.

To estimate an initial difficulty for an example we consider the character unigram log probability of the lemma and inflection. For a word (either the lemma or inflection) $w = c_0, ..., c_K$, the unigram log probability is given by:

$$\log(P_U(w)) = \sum_{k=0}^{K} \log(p(c_k)) \quad (1)$$

To get a score for a lemma and inflection pair (henceforth (x, y)), we calculate it as the sum of the log probabilities of x and y:

$$score(x, y) = P_U(x) + P_U(y) \quad (2)$$

Note that here we do not normalize by the length of the inflection and lemma. This is because an additional factor in how difficult an example should be considered is its length, i.e. longer words are harder to model. We then sort the examples and use a cumulative density function (CDF) to map the unigram probabilities to a score in the range $(0, 1]$, we denote the training set of pairs and their scores $((x, y), s)_0, \ldots, ((x, y), s)_m$, where m indicate the number of examples in the dataset, as $\mathcal{D}$.

To select appropriate training examples from $\mathcal{D}$ we must estimate the competence c of our model. The competence of the model is estimated by a function of the number of training steps t taken:

$$c(t) = \min\left(1, \sqrt{t\frac{1 - c(1)^2}{c(1)^2} + c(1)^2}\right) \quad (3)$$

During training, we employ a probabilistic approach to constructing batches from our corpus, we uniformly draw samples $((x, y), s)$ from the training set $\mathcal{D}$ such that the score s is lower than the model competence $c(t)$. This ensures that for each training step, we only consider examples that the model can handle according to our curriculum schedule.

However, just because an example has low unigram probability doesn't ensure that the example is easy, as the example may contain frequent characters but also include rare morphological processes (or rare combinations of Levenshtein operations), to account for this we recompute the example scores at each training step. We sort the examples in each training step according to the **decoding loss**, then assign a new score to the examples in the range $(0, 1]$ using a CDF function.

We also have to take into account that as the model competence grows, "easy" (low loss or high unigram probability) examples will be included more often in the batches. To ensure that the model learns more from examples whose difficulty is close to its competence we compute a weight w for each example in the batch. We then scale the loss by dividing the score s by the model competence at the current time-step:

$$\text{weighted loss}(x, y) = \text{loss}(x, y) \times \frac{score(x, y)}{c(t)} \tag{4}$$

Because the value of our model competence is tied to a specific number of training steps, we develop a probabilistic strategy for sampling batches when the model has reached full competence. When the model reaches full competence we construct language weights by dividing the number of examples in a language by the total number of examples in the dataset and taking the inverse distribution as the language weights. Thus for each language, we get a value in the range $(0, 1]$ where low-resource languages receive a higher weight. To construct a batch we continue by sampling examples, but now we only add an example if $r \sim \rho$, where ρ is a uniform Bernoulli distribution, is less than the language weight of the example. This strategy allows us to continue training our model after reaching full competence without neglecting the low-resource languages.

In total we train the model for $240\,000$ training steps, and consider the model to be fully competent after $60\,000$ training steps.

3.4 Scheduled Sampling

Commonly, when training an encoder-decoder RNN model, the input at time-step t is not the output from the decoder at $t - 1$, but rather the gold data. It has been shown that models trained with this strategy may suffer at inference time. Indeed, they have never been exposed to a partially incorrect input in the training phase. We address this issue using scheduled sampling (Bengio et al., 2015).

We implement a simple schedule for calculating the probability of using the gold characters or the model's prediction by using a global sample probability variable which is updated at each training step. We start with a probability ρ of 100% to take the gold. At each training step, we decrease ρ by $\frac{1}{\text{totalsteps}}$. For each character, we take a sample from the Bernoulli distribution of parameter ρ to determine the decision to make.

3.5 Training

We use cross-entropy loss for the character generation loss and for the operation predictions tasks. Our final loss function consists of the character generation loss, the lemma-reduction, and the lemma-addition losses summed. We use a cosine annealing learning rate scheduler (Loshchilov and Hutter, 2017), gradually decreasing the learning rate. The hyperparameters used for training are presented in Table 1.

HYPERPARAMETER	VALUE
Batch Size	256
Embedding dim	128
Hidden dim	256
Training steps	240000
Steps for full competence	60000
Initial LR	0.001
Min LR	0.0000001
Smoothing-α	2.5%

Table 1: Hyperparameters used. As we use a probabilistic approach to training we report number of training steps rather than epochs. In total, the number of training steps we take correspond to about 35 epochs.

Language-wise Label smoothing We use language-wise label smoothing to calculate the loss. This means that we remove a constant α from the probability of the correct character and distribute the same α uniformly across the probabilities of the characters belonging to the language of the word. The motivation for doing label smoothing this way is that we know that all incorrect character predictions are not *equally* incorrect. For example, when predicting the inflected form of a Modern Standard Arabic (ara) word, it is more correct to select *any* character from the Arabic alphabet than a character from

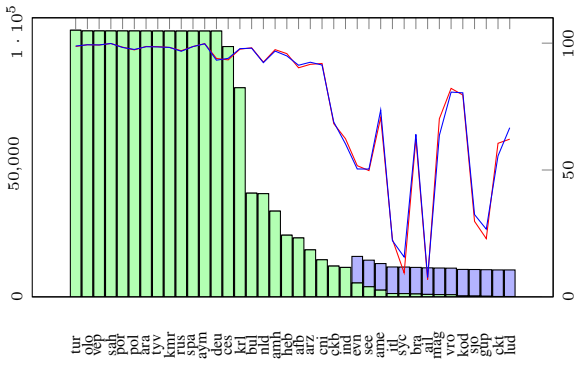

Figure 3: Number of examples (green indicate natural and blue hallucinated examples, left x-axis) plotted against the exact match accuracy (right x-axis) of our system on the development data (blue) and the test data (red).

the Latin or Cyrillic alphabet. A difficulty is that each language potentially uses a different set of characters. We calculate this set using the training set only— so it is important to make α not too large, so that there is not a too big difference between characters seen in the training set and those not seen. Indeed, if there were, the model might completely exclude unseen characters from its test-time predictions. (We found that $\alpha = 2.5\%$ is a good value.)

4 Results

The results from our system using the four straining strategies presented earlier are presented in Table 2. Each language is evaluated by two metrics, exact match, and average Levenshtein distance. The average Levenshtein distance is on average, how many operations are required to transform the system's guess to the gold inflected form. One challenging aspect of this dataset for our model is balancing the information the model learns about low- and high-resource languages. We plot the accuracy the model achieved against the data available for that language in Figure 3.

We note that for all languages with roughly more than 30 000 examples our model performs well, achieving around 98% accuracy. When we consider languages that have around 10 000 natural examples and no hallucinated data the accuracy drops closer to round 50%. For the languages with hallucinated data, we would expect this trend to continue as the data is synthetic and does not take into account orthographic information as natural language examples do. That is, when constructing hallucinated examples, orthography is taken into account only indirectly because we consider n-grams instead of characters when finding the replacement sequence. However, we find that for many of the languages with hallucinated data the exact match accuracy is above 50%, but varies a lot depending on the language.

	Test		Dev	
LANG	EM	LEV	EM	LEV
afb	90.29	0.17	91.29	0.15
ail	6.84	3.6	7.69	3.62
ame	70.72	0.75	73.67	0.64
amh	97.44	0.04	96.87	0.04
ara	98.69	0.03	98.59	0.04
arz	91.65	0.14	92.48	0.14
aym	99.8	0.0	99.75	0.01
bra	62.38	0.71	64.05	0.59
bul	98.16	0.03	98.02	0.03
ces	93.41	0.12	94.01	0.12
ckb	68.27	0.77	68.91	0.73
ckt	60.53	1.37	55.56	1.72
cni	91.99	0.11	91.38	0.12
deu	93.95	0.09	93.28	0.1
evn	51.7	1.47	50.41	1.5
gup	22.95	3.92	26.67	3.1
heb	95.86	0.09	94.97	0.1
ind	62.32	1.3	60.28	1.33
itl	22.63	2.89	22.16	3.11
kmr	98.19	0.02	98.32	0.02
kod	79.57	0.58	80.43	0.37
krl	97.62	0.05	97.83	0.04
lud	62.16	0.73	66.67	0.44
mag	70.2	0.53	63.64	0.71
nld	92.51	0.12	92.31	0.12
olo	99.39	0.01	99.36	0.01
pol	97.34	0.04	97.51	0.04
por	98.41	0.03	98.3	0.03
rus	97.02	0.05	96.8	0.05
sah	99.86	0.0	99.86	0.0
see	49.77	1.7	50.37	1.51
sjo	29.76	1.73	32.43	1.83
spa	98.66	0.02	98.65	0.02
syc	9.43	5.25	15.7	5.41
tur	98.69	0.03	98.86	0.03
tyv	98.61	0.03	98.51	0.03
vep	99.26	0.01	99.3	0.01
vro	82.17	0.31	80.7	0.42

Table 2: Results on the development data.

Two of the worst languages in our model is Classical Syriac (syc) and Xibe (sjo). An issue with Classical Syriac is that the language uses a unique script, the Syriac abjad, which makes it difficult for the model to transfer information about operations and common character combinations/transformations into Classical Syriac from related languages such as Modern Standard Arabic (spoken in the region). For Xibe there is a similar story: it uses the Sibe alphabet which is a variant of Manchu script, which does not occur elsewhere in our dataset.

5 Language similarities

Our model process many languages simultaneously, thus it would be encouraging if the model also was able to find similarities between languages. To explore this we investigate whether the language embeddings learned by the model produce clusters of language families. A t-SNE plot of the language embeddings is shown in Figure 4.

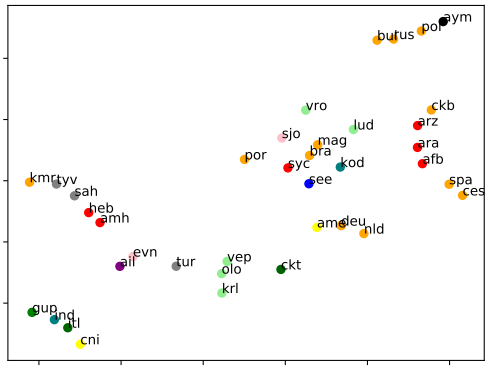

Figure 4: t-SNE plot of the language embeddings. Different colors indicate different language families.

The plot shows that the model can find some family resemblances between languages. For example, we have a Uralic cluster consisting of the languages Veps (vep), Olonets (olo), and Karelian (krl) which are all spoken in a region around Russia and Finland. However, Ludic (lud) and Võro (vro) are not captured in this cluster, yet they are spoken in the same region.

We can see that the model seem to separate language families somewhat depending on the script used. The Afro-Asiatic languages are split into two smaller clusters, one cluster containing the languages that use Standard Arabic (ara, afv and arz) script and one cluster that use Amharic and Hebrew (amh, heb) script. As mentioned earlier Classical Syriac uses its another script and seems to consequently appear in another part of the map.

In general, our model's language embeddings appear to learn some relationships between languages, but certainly not all of them. However, that we find some patterns in encouraging for future work.

6 Scheduled Sampling

We note that during the development all of our training strategies showed a stronger performance for the task, except one: scheduled sampling. We hypothesize this is because the low-resource languages benefit from using the gold as input when predicting the next character, while high-resource languages do not need this as much. The model has seen more examples from high-resource languages and thus can model them better, which makes using the previous hidden state more reliable as input when predicting the next token. Indeed, the scheduled sampling degrade the overall performance by 3.04 percentage points, increasing our total average accuracy to 83.3 percentage points, primarily affecting low-resource languages.

7 Conclusions and future Work

We have presented a single multilingual model for morphological inflection in 38 languages enhanced with different training strategies: curriculum learning, multi-task learning, scheduled sampling and language-wise label smoothing. The results indicate that our model to some extent capture similarities between the input languages, however, languages that use different scripts appears problematic. A solution to this would be to employ transliteration (Murikinati et al., 2020).

In future work, we plan on exploring curriculum learning in more detail and move away from estimating the competence of our model linearly, and instead, estimate the competence using the accuracy on the batches. Another interesting line of work here is instead of scoring the examples by model loss alone, but combine it with insights from language acquisition and teaching, such as sorting lemmas based on their frequency in a corpus (Ionin and Wexler, 2002; Slabakova, 2010).

We also plan to investigate language-wise label smoothing more closely, specifically how the value of α should be fine-tuned with respect to the number of characters and languages.

Acknowledgments

The research reported in this paper was supported by grant 2014-39 from the Swedish Research Council, which funds the Centre for Linguistic Theory and Studies in Probability (CLASP) in the Department of Philosophy, Linguistics, and Theory of Science at the University of Gothenburg.

References

Antonios Anastasopoulos and Graham Neubig. 2019. Pushing the limits of low-resource morphological inflection. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 984–996. Association for Computational Linguistics.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1171–1179.

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 *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 876–885. The Association for Computational Linguistics.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *CoRR*, abs/1410.5401.

Martin Haspelmath and Andrea Sims. 2013. *Understanding morphology*. Routledge.

Tania Ionin and Kenneth Wexler. 2002. Why is 'is' easier than '-s'?: acquisition of tense/agreement morphology by child second language learners of english. *Second language research*, 18(2):95–136.

Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Katharina Kann and Hinrich Schütze. 2016. Med: The lmu system for the sigmorphon 2016 shared task on morphological reinflection. In *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 62–70.

Geethan Karunaratne, Manuel Schmuck, Manuel Le Gallo, Giovanni Cherubini, Luca Benini, Abu Sebastian, and Abbas Rahimi. 2021. Robust high-dimensional memory-augmented neural networks. *Nature communications*, 12(1):1–12.

Xuebo Liu, Houtim Lai, Derek F. Wong, and Lidia S. Chao. 2020. Norm-based curriculum learning for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 427–436. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2017. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Peter Makarov, Tatiana Ruzsics, and Simon Clematide. 2017. Align and copy: UZH at SIGMORPHON 2017 shared task for morphological reinflection. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection, Vancouver, BC, Canada, August 3-4, 2017*, pages 49–57. Association for Computational Linguistics.

Nikitha Murikinati, Antonios Anastasopoulos, and Graham Neubig. 2020. Transliteration for cross-lingual morphological inflection. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 189–197, Online. Association for Computational Linguistics.

Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabas Poczos, and Tom M Mitchell. 2019. Competence-based curriculum learning for neural machine translation. *arXiv preprint arXiv:1903.09848*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Abhishek Sharma, Ganesh Katrapati, and Dipti Misra Sharma. 2018. IIT(BHU)–IIITH at CoNLL–SIGMORPHON 2018 shared task on universal morphological reinflection. In *Proceedings of the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection*, pages 105–111, Brussels. Association for Computational Linguistics.

Roumyana Slabakova. 2010. What is easy and what is hard to acquire in a second language?

Peter Smit, Sami Virpioja, Stig-Arne Grönroos, and Mikko Kurimo. 2014. Morfessor 2.0: Toolkit for statistical morphological segmentation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics,*

EACL 2014, April 26-30, 2014, Gothenburg, Sweden, pages 21–24. The Association for Computer Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.