Quantifying the Hyperparameter Sensitivity of Neural Networks for Character-level Sequence-to-Sequence Tasks

Adam Wiemerslage[†] **Kyle Gorman**[‡] **Katharina von der Wense**[†♈]
[†]University of Colorado Boulder
[♈]Johannes Gutenberg University Mainz
[‡]Graduate Center, City University Of New York
adam.wiemerslage@colorado.edu kgorman@gc.cuny.edu katharina.kann@colorado.edu

Abstract

Hyperparameter tuning, the process of searching for suitable hyperparameters, becomes more difficult as the computing resources required to train neural networks continue to grow. This topic continues to receive little attention and discussion—much of it hearsay—despite its obvious importance. We attempt to formalize *hyperparameter sensitivity* using two metrics: similarity-based sensitivity and performance-based sensitivity. We then use these metrics to quantify two such claims: (1) transformers are more sensitive to hyperparameter choices than LSTMs and (2) transformers are particularly sensitive to batch size. We conduct experiments on two different character-level sequence-to-sequence tasks and find that, indeed, the transformer is slightly more sensitive to hyperparameters according to both of our metrics. However, we do not find that it is more sensitive to batch size in particular.

1 Introduction

Neural networks are famously hard to interpret and slightly mysterious to researchers and practitioners alike. Hyperparameter tuning, typically an important part of developing a neural model, is often perceived as black magic—based on trial and error or inherited recipes. This has resulted in urban legends within the speech and NLP communities regarding deep learning models and their hyperparameters. At the same time, actual hyperparameter tuning is not commonly performed. Larger models, along with the increasing multilinguality of NLP models and tasks, make hyperparameter tuning more expensive, and the prevalence of the pretraining–fine-tuning paradigm has many convinced that hyperparameter tuning is unimportant.

We consider two specific claims regarding hyperparameters. First, we test whether transformers (Vaswani et al., 2017) are more sensitive to specific hyperparameter choices than LSTMs (Hochreiter and Schmidhuber, 1997). This can be found in different paraphrased versions in multiple ACL-published studies; we provide examples in Table 1. Second, we investigate the claim that transformers are particularly sensitive to batch size when compared to LSTMs (e.g., Popel and Bojar, 2018; Wu et al., 2021). Such statements are poorly defined since there is no accepted definition of *hyperparameter sensitivity*.

"...(Transformer) neural networks are very sensitive to architecture and hyperparameter settings." (Murray et al., 2019)
"Unlike the LSTM-based model...the Transformer-based architecture was found to be sensitive to such changes." (Stengel-Eskin et al., 2021)
"The Transformer models were even tuned longer, since they were more sensitive to small hyperparameter changes." (van Noord et al., 2022)
"It is acknowledged that Transformer [sic] model is extremely sensitive to the hyper-parameters..." (Inaguma et al., 2020)
"We...find (i) framing the task...and (ii) several additional techniques...can mitigate the (pretrained transformer)'s extreme sensitivity to hyperparameters." (Liu et al., 2020)

Table 1: Excerpts from ACL papers mentioning the hyperparameter sensitivity of transformers.

We first motivate and describe task- and architecture-agnostic metrics for quantifying hyperparameter sensitivity. Then, we run a study where we apply our metrics to two character transduction tasks (i.e., character-level sequence-to-sequence tasks that typically have very short sequences; Wu et al., 2021): morphological inflection and grapheme-to-phoneme conversion. We perform an extensive hyperparameter search for both LSTM and transformer models for the two tasks, together covering ten languages.

We find that transformers are typically more sensitive to hyperparameters according to both of our

Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 674–689

metrics. While, for batch size in particular, the transformer is more sensitive, we do not find that this sensitivity is any greater than all parameters together, which contradicts the claims of previous work. In addition, we also define a new state of the art for both tasks and each language as a side effect of our extensive tuning: surprisingly, we show that, in contrast to the results of Wu et al. (2021), transformers do *not* outperform LSTMs with attention on either task with equivalent random tuning budget. Finally, we analyze the most successful hyperparameter ranges for each architecture, and find that typical ranges used previously for LSTMs may be sub-optimal.

2 Related Work

Hyperparameter sensitivity is frequently discussed in the NLP literature beyond the claims in Table 1. Olsen and Plank (2021) investigate the hyperparameter sensitivity of convolutional neural networks (CNNs) by looking at the mean and max performance resulting from a hyperparameter search. Britz et al. (2017) discuss hyperparmeter sensitivity of recurrent neural networks (RNNs) for machine translation, and perform a large ablation study of which hyperparameters work best. Zhang and Wallace (2017) discuss hyperparameter sensitivity of CNNs and make tuning suggestions to practitioners. In contrast, our goal is to propose reproducible metrics for comparing hyperparameter sensitivity across architectures. Popel and Bojar (2018) present suggestions for training transformers, and Wu et al. (2021) do the same for character transduction tasks in particular. Both works claim transformers are particularly sensitive to batch size. Zhou et al. (2022) propose a knowledge distillation method that they claim is insensitive to hyperparameters. Similar in spirit to our work, Dodge et al. (2019) define the *expected validation performance* of an architecture, given some computation budget for hyperparameter search. Instead, we define metrics that are concerned with the *variance* of performance as a function of hyperparameter settings.

3 Hyperparameter Sensitivity

We define hyperparameter sensitivity as the extent to which an architecture changes in performance due to changes in hyperparameters. We distinguish between two types of sensitivity: **performance-based sensitivity** describes how likely high performance is across a random hyperparameter search—a common technique often resulting in good hyperparameters (Bergstra and Bengio, 2012). In contrast, **similarity-based sensitivity** refers to how close performance is for similar hyperparameter configurations, a metric more relevant for structured hyperparameter optimization (e.g., Bergstra et al., 2011). Both types of sensitivity are quantified in terms of changes in model accuracy.

We present multiple metrics for each type of hyperparameter sensitivity. Figure 1 provides a visual intuition of the two types of hyperparameter sensitivity and the relationship to accuracy that the metrics measures. In Case 1, we have an architecture that is fully robust to any changes in hyperparameters. This is not sensitive at all according to the performance-based metrics. In Case 2, we have an architecture that linearly increases in performance as we make small changes to hyperparameters. Although this may be a sensitive model according to performance-based metrics, it is not sensitive according to similarity-based metrics. Finally, Case 3 is more sensitive in terms of both types of metrics due to its inconsistent changes in performance, though still less sensitive than a completely random distribution of accuracy. Given the large hyperparameter space that we study, we expect our outcomes to be closest to Case 3.

3.1 Preliminaries

Random Hyperparameter Search Hyperparameter search requires training several models with different configurations. The goal is to find the optimal configuration that leads to the best learning algorithm. Traditionally, for neural networks, this has often been performed by searching over every combination of finite sets of manually chosen hyperparameters—typically referred to as *grid search*. Bergstra and Bengio (2012) find that *random search* leads to better outcomes. In a random hyperparameter search, we set a distribution for the values of each hyperparameter, and sample from these distributions for each configuration. Our metrics are defined in terms of a random hyperparameter search, which allows us to cover a large hyperparameter space for comparing sensitivity and also samples configurations that we as researchers may not typically search over in a more manual setup.

Notation To compute the proposed metrics, we assume a random hyperpameter search with a budget of n "runs" for a given architecture and dataset, with r_i denoting the hyperparameter configuration

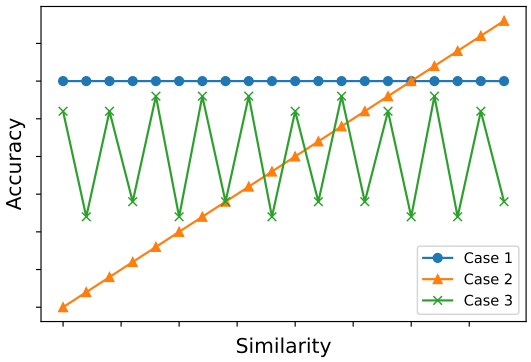

Figure 1: Possible relationships between hyperparameter similarity (defined in §3.3) and accuracy.

of the ith run. In our experiments we define a dataset as the combination of a task (e.g., morphological inflection) and a language (e.g., Dutch). The performance of r_i is denoted by $\text{acc}_i \in A$, where A is the vector of all accuracies from one search.[1] The set of varied hyperparameters is denoted by H, and a particular hyperparameter, e.g., batch size, is h_j. The value of hyperparameter h_j in run r_i is h_j^i.

3.2 Performance-based Sensitivity

Each performance-based metric compares architectures in terms of how many hyperparameter configurations in a random search attain a relatively high performance, i.e., a performance that is close to the best one found during the search. A practical question that this sheds light on is: *in a random hyperparameter search, how likely am I to find an optimally*[2] *performing model?* Sensitivity is more concerned with measuring variance in performance than finding the best run or architecture, so performance-based sensitivity considers *relative* accuracy to the best performing system.

Relative Performance at k This measures the difference in performance between the best performing run r_1, and the kth best run in the hyperparameter search. To compute this, we first sort all runs by performance, and then we simply compute $\frac{\text{acc}_k}{\text{acc}_1}$. Intuitively, a higher value at a larger k means that there is a larger subset of runs that achieve high accuracy, and, thus, implies lower sensitivity.

Relative Mean Performance of First k In order to capture more nuance in the change across runs from the best to the kth, we additionally compute the mean performance of the first k runs and report it as relative to acc_1. This reduces the impact of the choice of a particular k. We first rank all runs by performance, and then, for a given k, we compute

$$\frac{\sum_{i=2}^{k} \text{acc}_i}{(k-1) \cdot \text{acc}_1}. \quad (1)$$

Percentage of Best-equivalent Runs The percentage of systems within a *region of practical equivalence* (ROPE; Benavoli et al., 2017) of the best model measures how likely an optimal run is in a random hyperparameter search with a fixed-size window of optimal values. We define the ROPE as the $[1, -1]$ interval, meaning that we report the percentage of runs that achieve at most 1% accuracy less than acc_1. A higher percentage means that a larger number of runs achieve roughly optimal performance and, thus, implies lower sensitivity.

Expected Size of Equivalent Runs In a large hyperparameter search, there are likely to be unreasonable hyperparameter combinations. To account for this, we characterize the expected percentage of equivalent runs to any given run. To this end, for each acc_i, we compute the number of $\text{acc}_j \in A$ within a ROPE of $[1, -1]$, which we denote by δ_i. Then, we report the mean δ over all runs. This metric says less about the likelihood of finding an optimally perfomant run, but illustrates the robustness of a particular architecture to hyperparameter changes in general.

3.3 Similarity-based Sensitivity

An architecture for which small changes in hyperparameter values are associated with large changes in performance is sensitive according to similarity-based sensitivity. A practical question that this sheds light on is: *in a **structured** hyperparameter search, how predictably will changes to hyperparameters lead to better results?*

Measuring the similarity between runs is not straightforward because each h_j samples values from a different distribution, which could be continuous, discrete, or categorical. To resolve this, we quantify similarity with a non-parametric measure. Each run r_i is assigned a rank, denoted by rank_i, as follows. First, all $h_j^i \in h_j$ are sorted from the largest to smallest. We define γ_j^i as the index of h_j^i in the sorted vector, where $\gamma_j^i = \gamma_j^k$ if $h_j^i = h_j^k$.

[1] Although we use accuracy in this work, any metric that is positively correlated with system performance could be used.

[2] We refer to the best performance obtained during a search as *optimal*, though there may exist better configurations not found during search.

	Hyperparameter	values
Optimization	Batch size	{16, 32, . . . , 2048}
	Learning rate	$[1e^{-6}, 0.01]$
	β_1	[.8, .999]
	β_2	[.98, .999]
	Label smoothing	[0, .2]
	Scheduler	{reduceonplateau, warmupinvsqrt, (none)}
	Warmup samples*	{100, 200, . . . , 5M}
	Factor*	[.1, .9]
	Min. learning rate*	$[1e^{-7}, .001]$
	Learning rate patience*	{1, 2, . . . , 5}
Architectural	Embedding Size	{16, 32, . . . , 512}
	Hidden layer size	{64, 128, . . . , 2048}
	Encoder layers	Transformer: {2, 4, 6, 8}; LSTM: {1, 2}
	Decoder layers	Transformer: {2, 4, 6, 8}; LSTM: {1}
	Attention heads†	{2, 4, 8}
	Dropout	[0, .5]

Table 2: Hyperparameters and the distributions we sample from. We write continuous distributions as an interval [. . .] and discrete distributions by demonstrating the step size between samples: {1, 2, . . . , max}. We sample all values uniformly at random, with the exception of learning rate, for which we sample from a log uniform distribution. *: Conditional hyperparameters; ignored if the related scheduler is not chosen. †: Specific to the transformer.

That is, γ^i_j is the rank of r_i for the hyperpameter h_j. Then we compute the average rank of every hyperparameter in r_i:

$$\text{rank}_i = \frac{1}{n}\sum_{j=1}^{n}\gamma^i_j. \tag{2}$$

This process leaves the possibility of equal ranks, which complicates the requirement of some sensitivity metrics to sort all runs by rank. In order to account for this when sorting, we sort such that $\text{acc}_k > \text{acc}_{k+1}$ within equal ranks. When computing similarity with respect to a single hyperparameter, as in batch size, we compute Equation 2 by setting h_j to the batch size. In the case of categorical variables, as in learning rate scheduler, we need to manually choose a ranking based on intuitions about that hyperparameter. The subjectivity here is undesirable, but such cases are rare. For example, in our experiments, the learning rate scheduler is the only categorical variable. We describe how the learning rate scheduler is treated in Appendix A.

Performance Correlation We compute Spearman's ρ between similarity and accuracy relative to the best run. More formally, we build two vectors $|\text{rank}_1 - \text{rank}_i|$ and $\text{acc}_1 - \text{acc}_i$ for all $1 < i \leq n$, respectively, and then compute ρ between the vectors. This measures the extent to which hyperparameters and performance vary in the same direction. An architecture with a high ρ is less sensitive in terms of similarity-based sensitivity.

Number of Maxima To further characterize performance in terms of changes in hyperparameters, we compute the number of performance maxima when sorting by rank. Each maximum is an outlier for which a small change in hyperparameters reduces performance. Consider Case 3 in Figure 1: each maximum is a point where a small change in hyperparameters in either direction will reduce performance. We first sort A by rank, resulting in A^*. Then, we count the number of maxima using this ordering: that is, a_i such that $a_{i-1} < a_i > a_{i+1}$, where i refers to rank_i. Thus, more maxima indicate higher sensitivity.

Average Change in Performance Performance maxima can occur from small fluctuations in performance, which do not actually represent high sensitivity. We additionally compute the average change in performance between adjacent runs:

$$\frac{1}{n}\sum_{a_i \in \{A^* \setminus a_n\}} |a_{i+1} - a_i|. \tag{3}$$

A higher value means that adjacent points are fur-

Task	Arch	Rel@25 ↑		Rel@50 ↑		Rel@100 ↑		Rel@150 ↑	
Infl.	LSTM	99.31	(0.75)	98.86	(1.16)	98.29	(1.61)	96.75	(2.84)
	Trans.	99.42	(0.35)	99.03	(0.58)	96.91	(1.73)	74.54	(15.26)
G2P	LSTM	99.70	(0.30)	99.57	(0.40)	99.25	(0.63)	98.13	(1.65)
	Trans.	99.39	(0.54)	98.90	(1.01)	97.11	(1.97)	83.38	(10.36)

Table 3: Performance-based sensitivity metrics: Rel@k (and standard deviation) averaged over each language.

Task	Arch	μ@25 ↑		μ@50 ↑		μ@100 ↑		μ@150 ↑		% Best ↑		$\mathbb{E}$[%Equiv.] ↑	
Infl.	LSTM	99.59	(0.45)	99.31	(0.69)	98.96	(1.02)	98.56	(1.33)	44.85	(34.33)	47.08	(20.96)
	Trans.	99.62	(0.23)	99.42	(0.35)	98.81	(0.66)	96.00	(1.87)	29.71	(13.83)	22.67	(6.11)
G2P	LSTM	99.81	(0.19)	99.71	(0.28)	99.57	(0.40)	99.34	(0.58)	64.07	(21.56)	58.29	(15.69)
	Trans.	99.64	(0.30)	99.41	(0.53)	98.80	(0.96)	97.05	(1.81)	34.67	(26.91)	27.00	(18.35)

Table 4: Performance-based sensitivity metrics: Mean @k, Percentage of Best-equivalent Runs, and Expected Size of Equivalent Runs where each metric is averaged over all languages. Standard deviations are given in parentheses.

ther in accuracy from their most similar hyperparameters, which implies higher sensitivity.

4 Case Study: Character Transduction

We perform a case study to explore the hyperparameter sensitivity of transformers and LSTMs for character transduction tasks. We choose those tasks as the vocabularies and sequence lengths are typically quite short, and models train faster than for many other tasks, which makes it possible to run a large set of experiments within a few weeks. Furthermore, there are several high-quality benchmarks available. We also believe that hyperparameter sensitivity is of particular interest to the community working on these problems because most character transduction datasets are highly multilingual—this is true of transliteration (e.g., Roark et al., 2020) and text normalization (e.g., Bollmann, 2019) in addition to the tasks we report results on—and typically one model is trained for each language in the task. While our findings about the relative hyperparameter sensitivity of LSTMs vs. transformers may not generalize to the many tasks which lack these properties, the metrics we introduce can be applied to virtually any machine learning system. We submit that even for large pretrained models, where hyperparameter tuning is difficult and prohibitively expensive, sensitivity in the fine-tuning stage should be considered.

Morphological Inflection We experiment with morphological inflection (Infl.), the task of producing a word form given a lemma and inflectional tags. For example, given the English verb *be* and the tags for first person, singular, and present tense, a system should generate *am*. Following standard practice, we evaluate using exact match accuracy of the generated word.

Grapheme-to-Phoneme Conversion Grapheme-to-phoneme conversion (G2P) is the task of predicting the sequence of *phonemes* representing the sounds of a word from its orthographic form. Unlike inflection, the input and output vocabularies here are typically disjoint. We again use exact string match accuracy.

Data We consider a subset of languages from benchmarks for each task. We sample eight languages from the CoNLL-SIGMORPHON 2017 shared task on morphological reinflection (Cotterell et al., 2017) in the high setting for the inflection experiments. For G2P, we sample four languages from the SIGMORPHON 2021 shared task (Ashby et al., 2021) in the medium setting. For both tasks, we use accuracy on the development set to quantify sensitivity metrics. We additionally report the test set performance of the best model in each sweep according to development accuracy. See Table 7 for dataset details.

4.1 Models

We use two architectures. Following Kann and Schütze (2016), we use an **LSTM** encoder-decoder model (Cho et al., 2014; Sutskever et al., 2014) with soft attention (Bahdanau et al., 2015). Following Wu et al. (2021), we further use a **trans-**

Task	Arch	$\rho \uparrow$		Maxima $\downarrow$		μ length $\downarrow$		BS $\rho \uparrow$		BS Maxima $\downarrow$		BS μ length $\downarrow$	
Infl.	LSTM	.09	(.16)	53.88	(18.67)	15.93	(4.24)	.04	(.07)	45.75	(19.32)	15.11	(3.46)
	Trans.	.05	(.07)	58.75	(9.36)	31.56	(4.77)	−.02	(.10)	56.12	(8.06)	30.16	(4.88)
G2P	LSTM	−.04	(.04)	46.25	(27.29)	15.22	(5.36)	−.01	(.05)	43.25	(26.06)	15.87	(5.68)
	Trans.	.07	(.04)	60.75	(7.93)	27.35	(4.17)	.07	(.07)	56.75	(6.34)	27.00	(3.01)

Table 5: Similarity-based sensitivity metrics averaged over each language. Standard deviations are given in parentheses. We additionally present similarity-based sensitivity for just the batch size denoted by BS.

former encoder-decoder model (Vaswani et al., 2017). Each run trains for up to 800 epochs with a patience of 50 and a fixed random seed. Development set accuracy is evaluated every four epochs; we report best development accuracy.

Hyperparameters We report all sixteen hyperparameters and the distributions their values are sampled from in Table 2. All but three hyperparameter distributions are the same for both architectures. As is standard practice, we use a single attention head for the LSTM, though this is a varied hyperparameter for transformers in our study. Similarly, we do not consider deep (i.e., many-layer) LSTM encoders or decoders, though we do for transformers. All sweeps consist of 200 runs.

5 Results

All results are discussed with respect to the average over all languages. We report per-language values in the appendix, in Table 10, Table 11 and Table 12.

5.1 Performance-based Sensitivity

Relative Performance Table 3 presents the relative performance at k for each task and architecture, averaged over all languages. At smaller relative $k = 100$ or lower, both models have extremely high scores, indicating that they are within a few points of accuracy of the best system. However, there is a significant drop in performance at $k = 150$ by the transformer, while the LSTM scores remain stable. This suggests that while neither architecture highly sensitive, if we look at the longer tail of suboptimal runs, transformers are more sensitive than LSTMs. Table 4 presents the mean relative performance of the first k, where even at $k = 150$, the scores between both architectures are comparable. This shows that the difference in sensitivity is less drastic than the rel@150 score implies.

Percentage of Best Equivalent Systems Table 4 also shows that, for both tasks, the percentage of best-equivalent runs for LSTMs is much higher, though with a much higher standard deviation across languages for inflection. We interpret this as showing that, with a ROPE interval of $[1, -1]$, there are some languages for which LSTMs will find an optimal accuracy for many hyperparameter configurations, though this is not true in other (perhaps more challenging) languages. Though transformers seem to, on average, attain very similar relative performance for the best 50–100 performing systems, LSTMs are much less sensitive overall.

Expected Size of Equivalent Runs The last column of Table 4 similarly shows that LSTMs consistently have a much higher expected size of equivalent runs. This shows that LSTMs are more robust to changing hyperparameters. That is, there is a much larger cluster of runs that result in roughly equivalent accuracy. Transformers are much more sensitive according to this metric.

5.2 Similarity-based Sensitivity

Table 5 presents results for the similarity-based sensitivity metrics for both tasks and architectures averaged over all languages.

Number of Maxima The number of maxima are very high for both architectures, though LSTM results vary more. Still, on average, LSTMs result in fewer maxima on both tasks. This implies that small changes to hyperparameters do not lead to systematic increases in accuracy in either architecture. That is, both architectures are sensitive to small changes in hyperparameters, but transformers, again, are more sensitive than LSTMs.

Average Change in Performance LSTMs have much lower average changes in performance than transformers. Despite the high number of maxima, this shows that transformers are quite a bit more sensitive than LSTMs. LSTM runs tend to be around two times closer in accuracy to their most similar runs than transformers.

Task	Lang	Max		Mean		% Zero		Test Acc.	
		Trans.	LSTM	Trans.	LSTM	Trans.	LSTM	Trans.	LSTM
Infl.	alb	99.90	99.80	77.65	92.68	13.50	5.50	99.30	99.30
	ara	97.70	97.50	76.75	85.67	13.00	9.50	95.40	95.30
	cat	99.20	98.80	80.75	88.61	11.50	8.00	98.50	98.60
	dut	98.70	98.50	78.82	91.36	10.50	5.00	97.10	97.00
	gle	94.40	94.80	67.77	81.16	16.00	10.00	92.40	93.10
	hai	99.00	99.00	73.32	86.75	15.00	12.00	98.00	99.00
	hun	88.50	91.60	67.12	79.29	17.00	9.00	86.20	91.30
	rom	90.70	93.00	73.49	84.06	8.50	4.50	88.20	91.30
	Avg.	96.01	96.63	74.46	86.20	13.12	7.94	94.39	95.61
G2P	arm_e	95.40	95.90	76.60	84.57	7.00	10.00	92.90	93.80
	dut	90.50	90.50	71.83	77.91	7.00	11.50	83.70	84.80
	geo	100.00	100.00	83.30	91.33	8.00	8.00	99.60	99.60
	hun	98.60	99.00	76.02	94.66	13.50	2.00	99.40	98.90
	Avg.	96.12	96.35	76.94	87.12	8.88	7.88	93.90	94.28

Table 6: Results on the exact match accuracy of each sweep for every language and task. Test accuracies are for the best performing model according to development accuracy.

Performance Correlation The ρ values are overall quite low: there does not seems to be a meaningful linear relationship between hyperparameter similarity and accuracy overall. However, the per-language breakdown in Table 12 in the appendix reveals some positive correlations for LSTMs.

5.3 Sensitivity to Batch Size

On the right side of Table 5, we additionally present similarity-based sensitivity metrics when batch size is the only hyperparameter considered. As mentioned in §2, batch size has frequently been discussed as a hyperparameter transformers are particularly sensitive to. Here we measure similarity-based sensitivity in the same way as before, but the ranking here only uses batch size. We find that transformers are more sensitive to batch size according to the number of maxima and average change in performance. However, the scores for batch size scale very closely to the scores for all hyperparameters, though they are consistently slightly lower. Indeed the only case of higher sensitivity to batch size than to all hyperparameters together is for LSTMs on the G2P task. We interpret this as showing that, while transformers are more sensitive to batch size than LSTMs, they are no more sensitive to it than they are to the full set of hyperparameters. Whereas previous work fixed all other hyperparameters and found batch size to have a high impact on optimization (Popel and Bojar, 2018; Wu et al., 2021), our work varies many hyperparameters together and does not support the claim that transformers are particularly sensitive to batch size, at least for character transduction.

5.4 Overall Performance

Table 6 summarizes the performance of the sweeps for all tasks, languages, and architectures. We present the mean and max accuracy on the development set, as well as the accuracy of the best model according to development accuracy on the test set. For both architectures, some hyperparameter configurations result in an accuracy of zero, which is likely due to certain unreasonable combinations of hyperparameters. We additionally present the percentage of runs for which this is the case. Transformers result in more zero-accuracy runs in every language for inflection, which signals another aspect of hyperparameter sensitivity. For G2P, LSTMs result in more zero-accuracy runs in Armenian and Dutch, however.

Contrary to most work on character transduction tasks, an LSTM run attains the best performance on almost every language for both tasks. When a transformer performs best, it is always within 0.1 percentage point of accuracy, with the single exception of Hungarian G2P, where the transformer performs better by an absolute 0.5 percent. **This contradicts prior work showing that transformers outperform LSTMs on character-level sequence-**

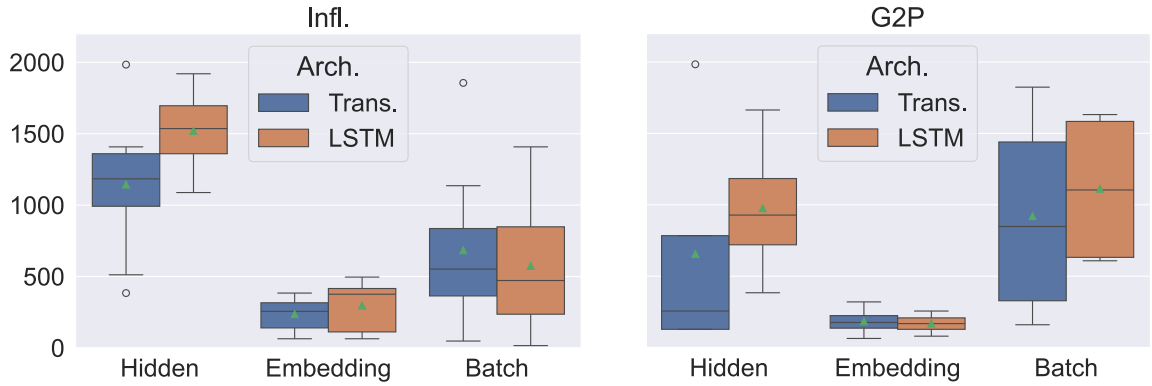

Figure 2: Boxplots of the hidden, embedding, and batch sizes of the best performing systems in each language and task. Mean values are marked with a small green triangle.

to-sequence tasks (Wu et al., 2021). One explanation for this in our experiments is that, being less sensitive to hyperparameters, in a large enough sweep, it is possible to train an LSTM which outperforms the transformers. Another explanation is that most work has simply not considered suitable hyperparameters for LSTMs. Indeed, one widely held belief about the superiority of transformers is due to the direct access of self-attention from any token in a sequence to any other token. LSTMs, on the other hand, potentially lose information through the recurrence over long enough sequences. As we consider tasks with shorter sequences, this apparent advantage of the transformer may be irrelevant.

5.5 Which Hyperparameters Work Best?

A side effect of our experiments is that we have found the best performing hyperparameters for each architecture from very large sweeps, for several languages. We present the best configurations in Table 8 and Table 9 in the appendix. In Figure 2, we present box plots for some hyperparameters of particular interest. Both architectures tend to perform best with large hidden sizes. Most works in character transduction tasks consider LSTM layers with a hidden sizes of less than 500—often as low as 100 (e.g., Kann and Schütze, 2016)—and assume that transformers require a larger hidden size. In our inflection experiments, the best LSTMs have hidden sizes about three times larger than what we would have expected—typically much larger than transformers on average. However, most of the optimal transformers have many layers—an option that is not available to LSTMs in our sweeps—which may make up for the smaller hidden sizes. Both architectures, and especially LSTMs, perform best with a large range of embedding sizes spanning almost our entire sample space. For G2P, hidden sizes are smaller but the trend is the same, and small embedding sizes seem to work best for both architectures. Finally, the optimal transformer batch sizes are not consistently higher than for LSTMs: in G2P the average LSTM batch size is higher. Typically LSTMs for character transduction are trained with very small batch sizes, but this does not seem necessary in our experiments. The best batch size is, however, highly varied in both architectures. One possible explanation is that the impact of batch size depends heavily on the learning rate and scheduler it is coupled with.

6 Conclusion

We have presented metrics for quantifying hyperparameter sensitivity according to two different definitions. In applying these metrics, we confirm that the LSTM is less sensitive to hyperparameters than the transformer for character transduction. Like in previous work, transformers also appear to be more sensitive to batch size than LSTMs. However, we find that they are no more sensitive to batch size than to all hyperparameters together. Lastly, we find that LSTMs outperform transformers for these tasks and languages with few exceptions, which contradicts previous findings. This is likely because we have sampled hyperparameters outside of the typical range for LSTMs in character transduction. We believe that a careful measurement of the relationship between architectures and hyperparameter search is important for testing claims about hyperparameter sensitivity. We hope that our metrics will be used to explore hyperparameter sensitivity in other tasks.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable feedback.

Limitations

The scope of this study is limited due to the large compute budget needed to cover more languages and tasks in a large hyperparameter search. For the similarity-based metrics in particular, we do see some variation in results, and a larger study on more data sets could potentially change the results. Additionally, in the large random search space, the number of runs is a limiting factor and more runs will always lead to more stable results. Furthermore, we have not yet used structured search methods in our evaluations.

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*, pages 115–125, Online. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. 2017. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *Journal of Machine Learning Research*, 18(77):1–36.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2).

Marcel Bollmann. 2019. A large-scale comparison of historical text normalization systems. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3885–3898, Minneapolis, Minnesota. Association for Computational Linguistics.

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1442–1451, Copenhagen, Denmark. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, Géraldine Walther, Ekaterina Vylomova, Patrick Xia, Manaal Faruqui, Sandra Kübler, David Yarowsky, Jason Eisner, et al. 2017. Conll-sigmorphon 2017 shared task: Universal morphological reinflection in 52 languages. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 1–30.

Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith. 2019. Show your work: Improved reporting of experimental results. 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)*, pages 2185–2194, Hong Kong, China. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hirofumi Inaguma, Shun Kiyono, Kevin Duh, Shigeki Karita, Nelson Yalta, Tomoki Hayashi, and Shinji Watanabe. 2020. ESPnet-ST: All-in-one speech translation toolkit. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 302–311, Online. Association for Computational Linguistics.

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, Berlin, Germany. Association for Computational Linguistics.

Haokun Liu, William Huang, Dhara Mungra, and Samuel R. Bowman. 2020. Precise task formalization matters in Winograd schema evaluations. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8275–8280, Online. Association for Computational Linguistics.

Kenton Murray, Jeffery Kinnison, Toan Q. Nguyen, Walter Scheirer, and David Chiang. 2019. Auto-sizing the transformer network: Improving speed, efficiency, and performance for low-resource machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 231–240, Hong Kong. Association for Computational Linguistics.

Benjamin Olsen and Barbara Plank. 2021. Finding the needle in a haystack: Extraction of informative COVID-19 Danish tweets. In *Proceedings of the Seventh Workshop on Noisy User-generated Text (W-NUT 2021)*, pages 11–19, Online. Association for Computational Linguistics.

Martin Popel and Ondřej Bojar. 2018. Training tips for the transformer model. *arXiv preprint arXiv:1804.00247*.

Brian Roark, Lawrence Wolf-Sonkin, Christo Kirov, Sabrina J. Mielke, Cibu Johny, Isin Demirsahin, and Keith Hall. 2020. Processing South Asian languages written in the Latin script: the Dakshina dataset. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 2413–2423, Marseille, France. European Language Resources Association.

Elias Stengel-Eskin, Kenton Murray, Sheng Zhang, Aaron Steven White, and Benjamin Van Durme. 2021. Joint universal syntactic and semantic parsing. *Transactions of the Association for Computational Linguistics*, 9:756–773.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Rik van Noord, Cristian García-Romero, Miquel Esplà-Gomis, Leopoldo Pla Sempere, and Antonio Toral. 2022. Building domain-specific corpora from the web: the case of European digital service infrastructures. In *Proceedings of the BUCC Workshop within LREC 2022*, pages 23–32, Marseille, France. European Language Resources Association.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

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*, pages 1901–1907, Online. Association for Computational Linguistics.

Ye Zhang and Byron Wallace. 2017. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 253–263, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Wangchunshu Zhou, Canwen Xu, and Julian McAuley. 2022. BERT learns to teach: Knowledge distillation with meta learning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7037–7049, Dublin, Ireland. Association for Computational Linguistics.

A Special Hyperparameters

In order to to handle certain hyperparameters in our experiments, we need to take care to handle some special edge cases, which we describe here. To ensure we can run every experiment on a single GPU, we set a maximum batch size of 256, and accumulate gradients on equal sized batches up to the requested batch size. This is to avoid out-of-memory errors on extremely large batch sizes, while still simulating the same gradient accumulation of the requested batch size in practice. We set a number of warmup *samples* when using a warmup scheduler, rather than number of warmup steps, as is typical. This is because the number of warmup steps is not comparable across different batch sizes. In practice, we compute a number of warmup steps at runtime as the number warmup samples divided by batch size. We also search over conditional hyperparameters, depending on the scheduler. These are (i) number of warmup samples (ii) factor (iii) LR patience (iv) minimum LR. We force each of these to 0 no matter what value is sampled in the hyperparameter search when the dependent scheduler is not sampled. Finally, when computing rank_i, we fix the order for our categorical variable: learning rate scheduler as follows: (i) None (ii) Reduce On Plateau (iii) Warmup. Our justification for this ranking reflects the observation that the reduce-on-plateau scheduler may not hit a plateau during all runs and thus is conceptually similar to no scheduling. Additionally, it can be thought of as simply tuning the learning rate in later stages of training. In contrast, the warmup scheduler will always have some impact on optimization.

Task	Source	Lang	Train	Dev	Test
Inflection	Cotterell et al. (2017)	ara	10k	1k	1k
		alb	10k	1k	1k
		cat	10k	1k	1k
		dut	10k	1k	1k
		gle	10k	1k	1k
		hai	6.84k	100	100
		hun	10k	1k	1k
		rom	10k	1k	1k
G2P	Ashby et al. (2021)	arm	10k	1k	1k
		dut	10k	1k	1k
		hun	10k	1k	1k
		geo	10k	1k	1k

Table 7: All data we use for experiments.

Task	Arch	batch size	LR	beta1	beta2	LS	sched.	wrmp steps	factor	LR patience	min LR
alb Infl.	LSTM	576	1.1e-3	0.92	1.0	0.04	wrmp.	1570	0	0	0
	Trans.	1136	2.1e-4	0.91	0.99	0.09	wrmp.	1968	0	0	0
ara Infl.	LSTM	368	8.26e-4	0.85	0.99	0.09	wrmp.	3483	0	0	0
	Trans.	1856	7.01e-4	0.88	1.0	0.13	None	0	0	0	0
cat Infl.	LSTM	304	3.56e-3	0.95	0.98	0.04	wrmp.	1847	0	0	0
	Trans.	432	8.66e-4	0.92	0.99	0.04	wrmp.	3488	0	0	0
dut Infl.	LSTM	752	8.21e-4	0.87	0.99	0.04	reduce	0	0.45	1	4.9e-4
	Trans.	576	4.22e-4	0.85	0.98	0.03	None	0	0	0	0
gle Infl.	LSTM	1408	1.02e-3	0.94	0.99	0.18	None	0	0	0	0
	Trans.	160	1.48e-3	0.81	0.99	0.14	reduce	0	0.77	3	9.63e-4
hai Infl.	LSTM	1136	1.64e-4	0.91	1.0	0.04	None	0	0	0	0
	Trans.	736	6.08e-4	0.92	1.0	0.08	reduce	0	0.34	2	6.13e-4
hun Infl.	LSTM	32	3.25e-5	0.84	0.98	0.07	None	0	0	0	0
	Trans.	528	1.99e-3	0.87	0.99	0.05	wrmp.	2246	0	0	0
rom Infl.	LSTM	16	7.12e-4	0.82	0.98	0.19	None	0	0	0	0
	Trans.	48	1.03e-3	0.87	0.99	0.15	wrmp.	47847	0	0	0
arm-e G2P	LSTM	640	6.49e-5	0.89	0.99	0.11	wrmp.	2026	0	0	0
	Trans.	1312	1.12e-4	0.86	0.98	0.1	None	0	0	0	0
dut G2P	LSTM	608	1.21e-4	0.93	0.99	0.15	None	0	0	0	0
	Trans.	384	2.2e-4	0.97	0.99	0.06	reduce	0	0.18	4	4.83e-4
geo G2P	LSTM	1568	1.12e-4	1.0	0.99	0.12	None	0	0	0	0
	Trans.	160	2.46e-4	0.97	0.99	0.12	wrmp.	5656	0	0	0
hun G2P	LSTM	1632	7.34e-4	0.8	1.0	0.005	wrmp.	165	0	0	0
	Trans.	1824	1.09e-3	0.81	0.99	0.06	wrmp.	558	0	0	0

Table 8: Optimization hyperparameters for the best performing system in each task and language.

Task	Arch	enc. layers	hidden size	dropout	emb. size	attn heads	dec. layers
alb Infl.	LSTM	1	1408	0.08	416	1	1
	Trans.	4	1152	0.16	352	8	4
ara Infl.	LSTM	2	1920	0.05	384	1	1
	Trans.	8	1216	0.23	64	4	6
cat Infl.	LSTM	2	1792	0.11	64	1	1
	Trans.	6	1408	0.29	304	4	6
dut Infl.	LSTM	2	1216	0.39	368	1	1
	Trans.	6	1152	0.3	384	2	8
gle Infl.	LSTM	2	1600	0.19	112	1	1
	Trans.	6	384	0.19	208	2	8
hai Infl.	LSTM	1	1472	0.26	496	1	1
	Trans.	8	1344	0.19	128	8	6
hun Infl.	LSTM	2	1664	0.43	416	1	1
	Trans.	4	1984	0.42	304	8	4
rom Infl.	LSTM	2	1088	0.28	112	1	1
	Trans.	8	512	0.08	144	4	2
arm-e G2P	LSTM	2	1664	0.17	144	1	1
	Trans.	6	384	0.28	320	2	6
dut G2P	LSTM	2	832	0.01	192	1	1
	Trans.	4	128	0.21	192	2	2
geo G2P	LSTM	1	384	0.06	80	1	1
	Trans.	4	1984	0.16	160	8	4
hun G2P	LSTM	1	1024	0.16	256	1	1
	Trans.	2	128	0.07	64	2	8

Table 9: Architectural hyperparameters for the best performing system in each task and language.

Task	Arch	Rel@25 ↑	Rel@50 ↑	Rel@100 ↑	Rel@150 ↑
alb Infl.	LSTM	100.00	99.90	99.80	99.50
	Trans.	99.80	99.70	98.30	84.78
ara Infl.	LSTM	99.49	99.18	98.36	96.72
	Trans.	99.28	98.87	95.70	84.03
cat Infl.	LSTM	99.70	99.70	99.60	99.29
	Trans.	99.60	99.29	98.29	87.50
dut Infl.	LSTM	99.49	99.29	98.98	97.77
	Trans.	99.29	98.89	97.37	84.60
gle Infl.	LSTM	98.95	98.42	97.36	93.04
	Trans.	99.26	98.52	93.64	48.52
hai Infl.	LSTM	100.00	100.00	100.00	100.00
	Trans.	100.00	100.00	98.99	53.54
hun Infl.	LSTM	97.71	96.83	95.74	93.56
	Trans.	99.10	98.53	96.72	72.32
rom Infl.	LSTM	99.14	97.53	96.45	94.09
	Trans.	99.01	98.46	96.25	81.04
arm-e G2P	LSTM	99.58	99.37	98.96	97.60
	Trans.	99.27	98.74	96.12	84.49
dut G2P	LSTM	99.34	99.12	98.56	96.13
	Trans.	98.67	97.57	95.14	78.90
geo G2P	LSTM	100.00	100.00	100.00	100.00
	Trans.	99.90	99.90	99.70	97.20
hun G2P	LSTM	99.90	99.80	99.49	98.79
	Trans.	99.70	99.39	97.46	72.92

Table 10: Performance-based sensitivity metrics: Rel @k for each language.

Task	Arch	μ@25 ↑	μ@50 ↑	μ@100 ↑	μ@150 ↑	% Best ↑	$\mathbb{E}$[%Equiv.] ↑
alb Infl.	LSTM	100.00	99.96	99.92	99.86	83.42	71.32
	Trans.	99.85	99.81	99.54	98.08	43.72	27.49
ara Infl.	LSTM	99.63	99.45	99.12	98.65	29.15	38.01
	Trans.	99.56	99.32	98.46	96.25	24.12	18.20
cat Infl.	LSTM	99.80	99.75	99.69	99.60	79.90	66.45
	Trans.	99.68	99.57	99.27	98.15	39.70	29.49
dut Infl.	LSTM	99.68	99.55	99.34	99.10	52.26	52.98
	Trans.	99.48	99.32	98.78	96.97	25.13	22.05
gle Infl.	LSTM	99.51	99.08	98.56	97.69	14.07	25.72
	Trans.	99.65	99.27	97.97	92.48	18.59	13.98
hai Infl.	LSTM	100.00	100.00	100.00	100.00	83.92	71.36
	Trans.	100.00	100.00	99.82	94.65	52.76	30.99
hun Infl.	LSTM	98.59	97.95	97.13	96.46	3.02	27.30
	Trans.	99.42	99.11	98.40	95.79	18.09	21.42
rom Infl.	LSTM	99.49	98.78	97.92	97.11	13.07	23.46
	Trans.	99.31	98.99	98.25	95.61	15.58	17.72
arm-e G2P	LSTM	99.72	99.58	99.37	99.10	54.27	52.08
	Trans.	99.56	99.30	98.55	96.50	22.61	18.07
dut G2P	LSTM	99.60	99.39	99.12	98.67	39.70	41.83
	Trans.	99.26	98.72	97.61	95.53	10.05	13.97
geo G2P	LSTM	100.00	100.00	100.00	100.00	88.94	78.91
	Trans.	99.93	99.92	99.86	99.67	72.36	54.10
hun G2P	LSTM	99.94	99.88	99.79	99.61	73.37	60.33
	Trans.	99.83	99.71	99.18	96.48	33.67	21.85

Table 11: Performance-based sensitivity metrics: Mean @k for each language.

Task	Arch	$\rho \uparrow$	Maxima $\downarrow$	μ length $\downarrow$	BS $\rho \uparrow$	BS Maxima $\downarrow$	BS μ length $\downarrow$
alb Infl.	LSTM	0.14	45.00	11.75	0.07	37.00	10.82
	Trans.	−0.04	52.00	33.91	0.03	53.00	33.79
ara Infl.	LSTM	0.04	61.00	19.53	−0.03	58.00	15.59
	Trans.	0.12	63.00	30.61	−0.12	60.00	29.77
cat Infl.	LSTM	0.16	53.00	15.88	0.06	34.00	15.62
	Trans.	0.09	62.00	27.15	−0.13	63.00	28.51
dut Infl.	LSTM	−0.00	60.00	10.06	−0.02	52.00	11.08
	Trans.	−0.03	64.00	28.88	−0.01	55.00	30.05
gle Infl.	LSTM	0.09	65.00	22.04	0.14	62.00	20.32
	Trans.	0.09	63.00	36.98	0.09	58.00	34.81
hai Infl.	LSTM	0.22	12.00	19.70	−0.04	6.00	18.59
	Trans.	0.04	38.00	38.64	−0.00	38.00	36.71
hun Infl.	LSTM	−0.23	71.00	15.62	0.11	54.00	16.50
	Trans.	0.14	62.00	31.57	−0.12	62.00	25.43
rom Infl.	LSTM	0.27	64.00	12.84	0.06	63.00	12.38
	Trans.	−0.01	66.00	24.73	0.11	60.00	22.18
arm-e G2P	LSTM	−0.06	65.00	17.28	−0.08	57.00	19.03
	Trans.	0.04	63.00	25.47	0.06	63.00	25.83
dut G2P	LSTM	0.01	61.00	19.79	−0.01	62.00	20.37
	Trans.	0.11	65.00	25.44	0.12	59.00	24.34
geo G2P	LSTM	−0.03	6.00	16.31	0.01	5.00	16.33
	Trans.	0.11	49.00	24.91	−0.02	48.00	26.51
hun G2P	LSTM	−0.07	53.00	7.49	0.04	49.00	7.73
	Trans.	0.04	66.00	33.59	0.13	57.00	31.30

Table 12: Similarity-based sensitivity metrics for each language. We additionally present similarity-based sensitivity for just the batch size denoted by BS.