An Empirical Exploration of Curriculum Learning for Neural Machine Translation

Xuan Zhang*[1], Gaurav Kumar[1], Huda Khayrallah[1], Kenton Murray[2], Jeremy Gwinnup[3], Marianna J Martindale[4], Paul McNamee[1], Kevin Duh[1], and Marine Carpuat[4]

[1]Johns Hopkins University
[2]University of Notre Dame
[3]Air Force Research Laboratory
[4]University of Maryland

Abstract

Machine translation systems based on deep neural networks are expensive to train. Curriculum learning aims to address this issue by choosing the order in which samples are presented during training to help train better models faster. We adopt a probabilistic view of curriculum learning, which lets us flexibly evaluate the impact of curricula design, and perform an extensive exploration on a German-English translation task. Results show that it is possible to improve convergence time at no loss in translation quality. However, results are highly sensitive to the choice of sample difficulty criteria, curriculum schedule and other hyperparameters.

1 Introduction

Curriculum learning (Bengio et al., 2009) hypothesizes that choosing the order in which training samples are presented to a learning system can help train better models faster. In particular, presenting samples that are easier to learn from before presenting difficult samples is an intuitively attractive idea, which has been applied in various ways in Machine Learning and Natural Language Processing tasks (Bengio et al., 2009; Tsvetkov et al., 2016; Cirik et al., 2016; Graves et al., 2017, inter alia).

In this paper, we conduct an empirical exploration of curriculum learning for Neural Machine Translation (NMT). NMT is a good test case for curriculum learning as training is prohibitively slow in the large data conditions required to reach good performance (Koehn and Knowles, 2017). However, designing a curriculum for NMT training is a complex problem. First, it is not clear how to quantify sample difficulty for this task. Second, NMT systems already rely on established data organization methods to deal with the scale and varying length of training samples (Khomenko et al., 2016; Doetsch et al., 2017; Sennrich et al., 2017; Hieber et al., 2017), and it is not clear how a curriculum should interact with these existing design decisions. Kocmi and Bojar (2017) showed that constructing and ordering mini-batches based on sample length or word frequency helps when training for one epoch. It remains to be seen how curricula impact training until convergence.

To address these issues, we adopt a probabilistic view of curriculum learning that lets us explore a wide range of curricula flexibly. Our approach does not order samples in a deterministic fashion. Instead, each sample has a probability of being selected for training, and this probability changes depending on the difficulty of the sample and on the curriculum's schedule. We explore difficulty criteria based on NMT model scores as well as linguistic properties. We consider a wide range of schedules, based not only on the easy-to-difficult ordering, but also on strategies developed independently from curriculum learning, such as dynamic sampling and boosting (Zhang et al., 2017; van der Wees et al., 2017; Wang et al., 2018).

We conduct an extensive empirical exploration of curriculum learning on a German-English translation task, implementing all training strategies in the Sockeye NMT toolkit.[1]. Our experiments confirm that curriculum learning can improve convergence speed without loss of translation quality, and show that viewing curriculum learning more flexibly than strictly training on easy samples first has some benefits. We also demonstrate that curriculum learning is highly sensitive to hyperpa-

*Corresponding author. xuanzhang@jhu.edu

[1]Sockeye is a state-of-the-art open-source NMT framework at https://github.com/awslabs/sockeye Our modification is publicly available at https://github.com/kevinduh/sockeye-recipes/tree/master/egs/curriculum

rameters, and no clear single best strategy emerges from the experiments.

In this sense, our conclusions are both positive and negative: We have confirmed that curriculum learning can be an effective method for training expensive models like those in NMT, but careful design of the specific curriculum hyperparameters is important in practice.

2 Related Work

Bengio et al. (2009) coined the term of curriculum learning to refer to techniques that guide the training of learning systems "by choosing which examples to present and in which order to present them in the learning system", and hypothesize that training on easier samples first is beneficial. While organizing training samples based on difficulty has been demonstrated in NLP outside of neural models – e.g., Spitkovsky et al. (2010) bootstrap unsupervised dependency parsers by learning from incrementally longer sentences – curriculum learning has gained popularity to address the difficult optimization problem of training deep neural models (Bengio, 2012). Bengio et al. (2009) improve neural language model training using a curriculum based on increasing vocabulary size. More recently, Tsvetkov et al. (2016) improve word embedding training using Bayesian optimization to order paragraphs in the training corpus based on a range of distributional and linguistic features (diversity, simplicity, prototypicality).

While curriculum learning often refers to organizing examples from simple to difficult, other data ordering strategies have also shown to be beneficial: Amiri et al. (2017) improve the convergence speed of neural models using spaced repetition, a technique inspired by psychology findings that human learners can learn efficiently and effectively by increasing intervals of time between reviews of previously seen materials.

Curriculum design is also a concern when deciding how to schedule learning from samples of different tasks either in a sequence from simpler to more difficult tasks (Collobert and Weston, 2008) or in a multi-task learning framework (Graves et al., 2017; Kiperwasser and Ballesteros, 2018). In this work, we focus on the question of organizing training samples for a single task.

In NMT, curriculum learning has not yet been explored systematically. In practice, training protocols randomize the order of sentence pairs in the training corpus (Sennrich et al., 2017; Hieber et al., 2017). There are works that speed training up by batching the samples of similar lengths (Khomenko et al., 2016; Doetsch et al., 2017). Such works attempt to improve the *computational efficiency*, while curriculum learning is supposed to improve the *statistical efficiency* — fewer batches of training examples are needed to achieve a given performance.

Kocmi and Bojar (2017) conducted the first study of curriculum learning for NMT by exploring the impact of several criteria for curriculum design on the training of a Czech-English NMT system for one epoch. They ensure samples within each mini-batch have similar linguistic properties, and order mini-batches based on complexity. They show translation quality can be improved by presenting samples from easy to hard based on sentence length and vocabulary frequency. However, it remains to be seen whether these findings hold when training until convergence.

Previous work has focused on dynamic sampling strategies, emphasizing training on samples that are expected to be most useful based on model scores or domain relevance. Inspired by boosting (Schapire, 2002), Zhang et al. (2017), at each epoch, assign higher weights to training examples that have lower perplexities under the model of previous epoch. Similarly, van der Wees et al. (2017) and Wang et al. (2018) improve the training efficiency of NMT by dynamically select different subsets of training data between different epochs. The former performs this dynamic data selection according to domain relevance (Axelrod et al., 2011) while the latter uses the difference between the training costs of two iterations.

Taken together, these prior works show that sample difficulty can impact NMT, but it remains unclear how to balance the benefits of existing sample randomization and bucketing strategies with intuitions about sample ordering, as well as which ranking criteria and strategies should be used. We revisit these ideas in a unified framework, via experiments on a German-English task, training until convergence.

3 A Probabilistic View of Curriculum Learning

Let (x, y) be a bitext example, where x is the source sentence and y is the target reference translation. We use subscripts i to denote the

sample index and assume a training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1,2,...S}$ of size S. Curriculum learning can be formulated in a probabilistic view, where each sentence pair (x_i, y_i) has a probability of being selected for training, and this sampling probability changes depending on the difficulty of the example and the curriculum schedule (Bengio et al., 2009).

Specifically, we segment the curriculum schedule into distinct phases t which correspond to different time points during training. For instance, $t = 1$ could be the first N checkpoints, $t = 2$ is the next N checkpoints, etc. The definition of phases is flexible: alternatively $t = 1$ may correspond to the first epoch, and $t = 2$ may correspond to the second epoch (or more). At each phase t, we maintain a multinomial distribution q_i^t over the examples in $\mathcal{D}$, i.e. $\sum_{i=1}^{S} q_i^t = 1$ and $q_i^t \geq 0 \; \forall i$. To implement the curriculum schedule that begins with easy examples, we would start at $t = 1$ by setting q_i^t to be high for easy examples and q_i^t to be low (or zero) for difficult examples. Gradually, for large t, we increase q_i^t for the more difficult examples. At some point, all examples have equal probability of being selected; this corresponds to the standard training procedure. An illustration of this probabilistic view of curriculum learning is shown in Figure 1.

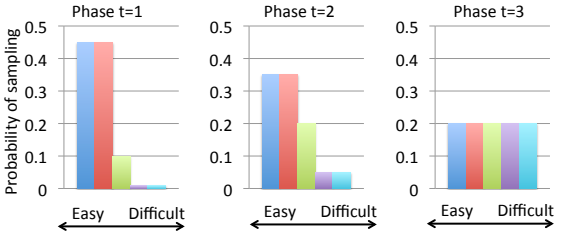

Figure 1: Probabilistic view of curriculum learning: On the x-axis, the examples are arranged from easy to difficult. y-axis is the probability of sampling the example for training. By specifying different kinds of sampling distributions at different phases, we can design different curriculums. In this example, $t = 1$ samples from the first three examples, $t = 2$ includes the remaining two examples but at lower probability, and $t = 3$ defaults to uniform sampling (regardless of difficulty).

There are two advantages to this probabilistic sampling view of curriculum learning:

1. It is a flexible framework that enables the design of various kinds of curriculum schedules. By specifying different kinds of distributions, one can perform easy-to-difficult training or the reverse difficult-to-easy training. One can default to uniform sampling, which corresponds to standard training with random mini-batches. Many of these variants are described in Section 5.2.

2. It is simple to implement in existing deep learning frameworks, requiring only a modification of the data sampling procedure. In particular, it is modular with respect to the optimizer's learning rate schedule and mini-batch shuffling mechanism; these represent best practice in deep learning, and may be suboptimal if modified. Further, the optimizer only needs access to sampling probability q_i^t, which abstracts away from the various difficulty criteria such as sentence length and vocabulary frequency (to be described in Section 4). This enables us to plug-in and experiment with many criteria.

Without loss of generality, in practice we recommend grouping examples into shards (Figure 2) such that those in the same shard have similar difficulty criteria values.[2] Then we define the sampling distributions over shards rather than examples. Since there are fewer shards than examples (e.g., 5 shards vs. 1 million examples for a typical-sized dataset), the distributions are simple to design and visualize. Sharding is described in more detail in Section 5.1.

Figure 2: Training data organized by level of difficulty. Each block is a shard (i.e., a subset of the dataset) and darker shades indicate increasing difficulty. Note that the width of each patch does *not* indicate the number of samples in that shard, as it may vary for different difficulty criteria.

4 Sample Difficulty Criteria

In this work, we quantify the translation difficulty of a sentence pair by two kinds of criteria (or score[3]): 1) how well an auxiliary translation

[2] Shards are not to be confused with buckets (grouping of similar-length samples). Shards are simply subsets of the training data and may allow for bucketing by length within themselves.

[3] Criteria and score are interchangeable in this paper.

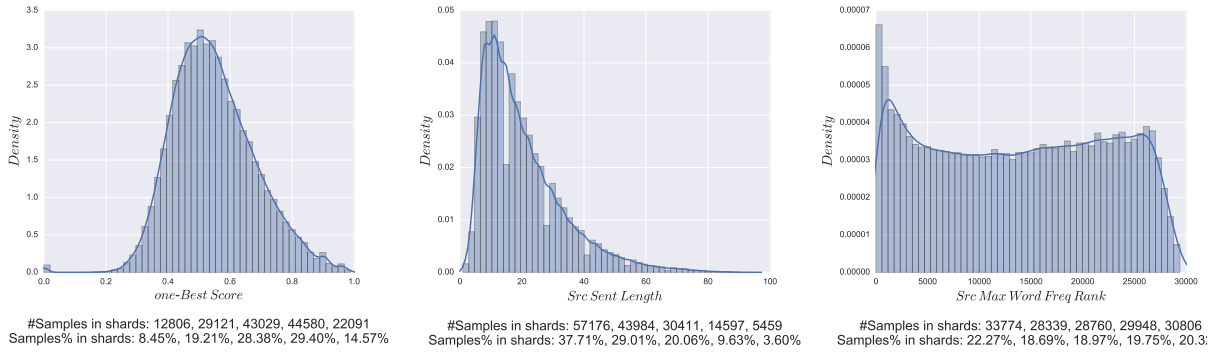

Figure 3: Difficulty score distribution on DE-EN TED Talks training set (151,627 sentence pairs in total) scored by selected difficulty criteria. Sharding results generated from Jenks Natural Breaks classification algorithm are shown below each subplot, in the ascending order of difficulty levels.

model captures the pair and 2) linguistic features which are orthogonal to any translation model.

Model-based Difficulty Criteria We use the *one-best score*, which is the probability of the one-best translation (the product of its word prediction probabilities) from an auxiliary (possibly simpler) translation model, given a source sentence. This represents $p(\hat{y} \mid x)$, where x is the source sentence and $\hat{y}$ is the one-best translation. A high *one-best score* for a translation suggests the auxiliary model is very certain of its prediction with small chance of choosing other candidates. Although the prediction might not be the "correct answer", $p(\hat{y} \mid x)$ shows the confidence of the model for that prediction, and indicates how easy the prediction is according to the model.

Linguistic Difficulty Criteria Linguistic features, including sentence length and vocabulary frequency, can also be used to measure the difficulty of translating a sample (Kocmi and Bojar, 2017). Short sentences usually do not have difficult syntactic structures, while lengthier sentences with long-distance dependencies are difficult to handle for NMT models (Hasler et al., 2017). To capture this phenomenon, we rank samples by the length of source and target sentence and by the sum of the length of each sentence in the pair.

Sutskever et al. (2014) shows that a NMT model's performance decreases on sentences with more rare words. Similar to Kocmi and Bojar (2017), we first sort words by their frequency to get the word frequency rank, then order sentences based on the rank of the least frequent word in the sentence (*max word frequency rank*). Organizing sentences by this criterion is equivalent to gradually increasing the vocabulary size and training on sentences that only contain words in the current partial vocabulary (Bengio et al., 2009). In addition to maximizing, we also experimented with the *average word frequency rank*. Again, we collect word frequency rank scores for source sentences, target sentences and concatenations of both[4].

5 Methods

Having defined criteria for measuring sample difficulty and illustrated how they can be used in a probabilistic curriculum learning framework, we now describe in more detail how this framework was instantiated for our study. We present our approach for organizing data into shards given sample difficulty scores (Section 5.1), how the shards are used by the curriculum schedule (Section 5.2), and how this fits in the overall training strategy (Section 5.3).

5.1 Data Sharding

As described in section 3, samples are grouped into shards of similar difficulty (Figure 2). This can be done by various methods. One approach is to set thresholds on the difficulty score (Kocmi and Bojar, 2017). An alternative is to distribute the data evenly such that each shard will have same number of samples. The first approach makes it difficult to choose reasonable breaks while trying to ensure that each shard has roughly the same number of samples (Figure 3). In contrast, the latter may result in unwanted fluctuations in difficulty within the same shard, and not enough difference between different shards.

[4]In the concatenation, the word rank is obtained based on whether the word belongs in the source or the target; i.e., we maintain separate word frequency lists for each language.

We instead use the Jenks Natural Breaks classification algorithm (Jenks, 1997), an algorithm commonly used in Geographic Information Systems (GIS) applications (Brewer, 2006; Chrysochoou et al., 2012). This method seeks to minimize the variance within classes and maximize the variance between classes. Figure 3 shows examples of the univariate classification results using Jenks algorithm on our training corpus (TED Talks, Duh (2018)) where training samples are reorganized by various criteria representing difficulty (Section 4). Distributions obtained for other complexity criteria are available in the supplementary material.

5.2 Curriculum Schedule

The curriculum's *schedule* defines the order in which samples of different difficulty classes are presented to the learning system. A curriculum's *phase* is the period between two curriculum updates.[5] For NMT models, it is natural to come up with the idea of first presenting easy samples to the models, as suggested by Bengio et al. (2009). In the following sections, we refer to this as the *default* schedule. We also introduce four variants of the *default* schedule (Figure 4) which lets us explore different trade-offs.

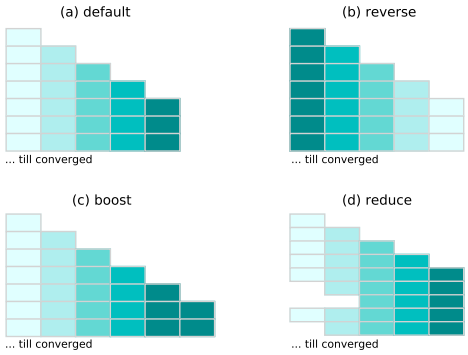

Figure 4: Training with different curriculum schedules. The colored blocks are shards of different difficulty levels (see figure 2). Within a sub-figure, each row represents a phase, and shards in that row are accessible shards based on the curriculum. Training starts from the first row and goes through the following rows in succession. Hence, at each phase only subsets of the training data and certain difficulty classes are available. Note that shards (and the samples within them) are shuffled as described in Section 5.3.

[5]This is similar to the concept of an epoch except that only a subset of the training data may be available based on the curriculum's schedule.

- **default** Shards are sorted by increasing level of difficulty. Training begins with the easiest shard and harder shards will be included in subsequent phases.
- **reverse** Shards are sorted in descending order of difficulty. Training begins with the hardest shard and easier shards will be included in subsequence phases.
- **boost** A copy of the hardest shard is added to the training set, after the model has processed shards of all difficulty classes.
- **reduce** Once all shards have been visited, we start removing shards from training one at the end of each phase, starting with the easiest. Once a fixed number of shards have been removed (2 in our case), we add them back. This *reduce and add-back* procedure will be iteratively continued until the training converges. The effect is that the model gets to look at harder shards more often.
- **noshuffle** Same as *default* except that shards are never shuffled; that is, they are always presented to the model in ascending order of difficulty (Samples within shards are shuffled as usual).

The *reverse* schedule tests the assumption that presenting easy examples first helps learning. It remains unclear if we should start with the easier sentences and move to more difficult ones, or if perhaps some of the difficult sentences are too hard for the model to learn and we should focus on straightforward sentences at the end. In addition, we are unsure of what the model will find more easy or difficult.

Another open question is whether presenting shards randomly during each curriculum phase (as done in the *default* schedule) weakens the curriculum. We explore an alternative by forcing the shard visiting order to be deterministic — always starting from the easiest shard, ending at the hardest shard for this phase. We label this schedule as *noshuffle*, since shuffling does not occur. *Noshuffle* may be helpful in the sense that every time the model is assigned with a new harder shard, it will review old shards in a more organized way. This method can be viewed as restarting the curriculum at each phase.

The last two schedules are adapted from Zhang et al. (2017), who improve NMT convergence

speed by duplicating samples considered difficult based on model scores. The *boost* schedule combines the idea of training on easy samples first (from *default*), while putting more emphasis on difficult samples (as in *reverse*). The *reduce* schedule additionally makes sure that the model gets to look at difficult shards more often. This is accomplished by removing easy shards from epochs and then adding them back again later.

5.3 Training Strategy

Finally, we address the question of how to draw mini-batches from the training data which has been sharded based on difficulty. Current state-of-the-art NMT model implementations bucket the training samples based on source and target length. Mini-batches are then drawn from these buckets, which are shuffled at each epoch. One way of drawing mini-batches while conditioning on difficulty is to sort the training samples by difficulty and to then draw these deterministically starting from the easiest to the most difficult sample. However, this loses the benefits gained by shuffling the data at each epoch.

Instead, our work uses a strategy similar to the work of Bengio et al. (2009). We organize samples into shards[6] according to the univariate classification results (Section 5.1) and allow further bucketing by sentence length within each shard. Samples within each shard are shuffled at each epoch, ensuring that we draw random mini-batches of the same difficulty.

Given shards of different difficulty levels, we follow these steps for training:

- The curriculum's schedule defines which shards are available for training. We call these the *visible* shards for this phase of curriculum training.

- These shards are then shuffled (except when we use the *noshuffle* schedule)[7] so that the model is trained using random levels of difficulty (in contrast to always using easy to hard).

- The samples within each shard are shuffled and bucketed by length. Mini-batches are drawn from these buckets.

- When the *curriculum update frequency* is reached (defined in terms of number of batches), the curriculum's schedule is updated. For example, this may imply that we include more difficult shards in training in the next phase. In cases where the total number of examples in these shards is smaller than the curriculum update frequency, we repeat the previous step until the update frequency has been achieved.

- After all available shards are visible to the model, training continues until validation perplexity does not improve for 32 checkpoints. The NMT model has then *converged*.

6 Experiment Setup

Data All experiments were conducted on the German-English parallel dataset from the Multi-target TED Talks Task (MTTT) corpus (Duh, 2018). The *train* portion consists of about 150k parallel sentences while the *dev* and *test* subsets have about 2k sentences each. All subsets were tokenized and split into subwords using byte pair encoding (BPE) (Sennrich et al., 2016). The BPE models were trained on the source and target language separately and the number of BPE symbols was set to 30k.

NMT Setup Our neural machine translation models were trained using Sockeye[8] (Hieber et al., 2017). We used 512-dimensional word embeddings and one LSTM layer in both encoder and decoder. We used word-count based batching (4096). Our systems employed the Adam optimizer (Kingma and Ba, 2014) with an initial learning rate of either 0.0002 or 0.0008 (see Section 7). The *dev* set from the corpus was used as a validation set for early stopping.

The baseline is an NMT model with the structure and hyperparameters described above without a curriculum; that is, it has access to the entire training set which is bucketed by length to then create mini-batches. Training data are split randomly into the same number of shards as the curriculum models (5 here).

We build the auxiliary model for the use of generating *one-best score* for each training sample, with similar but simpler configurations compared to the baseline model, in terms of number of RNN hidden units (200 vs. 512). While the training time

[6] 5 shards in our experiments.

[7] In shuffling, we ensure that the first shard for this phase is not the same as the last shard from the last phase.

[8] github.com/awslabs/sockeye

for this specific model may cancel out the time saved by curriculum learning in practice, having a high-quality *one-best score* provides a useful reference point for our understanding of curriculum learning.

Curriculum Learning Setup The curriculum learning framework as described in Section 5 was implemented within Sockeye. Curriculum learning can be enabled as an alternative to default training within Sockeye by specifying a file which contains sentence level scores (difficulty ranking per sentence with respect to any criterion). This implementation leverages the Sockeye sharding feature, which was originally meant for data parallelism. The codebase is publicly available with our experimental settings and tutorials[9].

We set the curriculum's update frequency to 1000 batches, which is the same as our checkpoint frequency.

7 Results

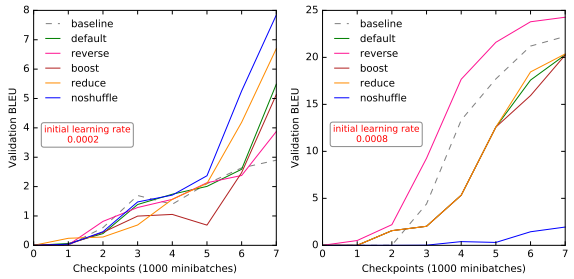

Figure 5: Learning curves for the first 7 curriculum updates. The NMT model is trained on data organized by the *avg word freq rank (de)* difficulty criterion with different curriculum learning schedules.

We start by examining training behavior during early training stages. Figure 5 shows the learning curves (validation BLEU[10] vs checkpoints) for the first 7 checkpoints[11] of curriculum training. The curriculum is updated at each checkpoint using one of the schedules listed in section 5.2. With the smaller learning rate, all curricula improve over baseline validation BLEU at the 7th checkpoint. However, with the higher learning rate, only the *reverse* schedule outperforms the baseline. Similar trends are observed with other difficulty criteria:[12] a few curriculum schedules beat the baseline but this outcome is sensitive to the initial learning rate.

Curr Update Freq	Time (thousand batches)	BLEU (7)	BLEU (best)
1000	108	8.8	28.2
2000	100	1.8	28.0
3000	71	9.2	28.2
4000	56	9.0	27.9
5000	108	14.9	28.0
6000	67	14.9	28.0

Table 1: Impact of curriculum update frequency on the model trained on *default* schedule with data organized by *avg word freq rank (de)*. Training time is quantified as total number of mini-batches the NMT model has processed before convergence. The initial learning rate is set to 0.0002. The last two columns show the decoding performance of the model at 7th and the *best checkpoint* — the checkpoint at which the model got highest BLEU score on val set.

When training until convergence (Tables 2-3), 20 of 100 curriculum strategies successfully converge earlier than the baseline without loss in BLEU. The model trained with the average source word frequency as a difficulty criterion and the *reverse* schedule improves training time by 19% to 30%.[13] However, the optimal curriculum schedule for other complexity criteria change with the initial learning rate. The model trained with the *one-best score* and the *boost* schedule converges after processing 19% fewer mini-batches than the baseline (59,000 vs. 73,000) and yields a comparable BLEU score (28.4 vs. 28.1) with an initial learning rate of 0.002. With a higher initial learning rate, this configuration also speeds up training by 38% (48,000 vs. 79,000) but at the cost of a 1.65 point degradation in BLEU. The default schedule yields better results with the learning rate of 0.0008 but not 0.0002.

Comparing trends across complexity criteria shows there is no clear benefit to the expensive one-best model score compared to the simpler word frequency criteria. Sentence length is not a useful criterion: it helps convergence time only slightly (74,000 vs. 79,000) and in only one of the ten configurations we run.This is a surprising result at first, given that both sentence length and

[9] https://github.com/kevinduh/sockeye-recipes/tree/master/egs/curriculum

[10] BLEU is the standard evaluation for machine translation based on n-gram precision; higher is better (Papineni et al., 2002).

[11] 7 is the lowest number of checkpoints required to discriminate between the different schedules.

[12] All learning curves available in Supplemental Material

[13] These are substantial time savings given that training the baseline took up to 1 day.

	Training Time (thousand batches)					**Test BLEU** (best)				
baseline	73					28.1				
	default	*reverse*	*boost*	*reduce*	*noshuffle*	*default*	*reverse*	*boost*	*reduce*	*noshuffle*
one-best score	56	80	**59**	64	92	27.0	27.9	**28.4**	27.3	27.4
max wd freq(de)	57	88	89	82	77	25.2	26.1	27.4	27.2	28.1
max wd freq(en)	**63**	77	75	64	98	**27.6**	25.3	27.5	26.9	27.6
max wd freq(deen)	**56**	61	**62**	**59**	**62**	**28.1**	27.5	**27.8**	**27.7**	**28.5**
ave wd freq(de)	**72**	**69**	57	73	108	**28.2**	**28.5**	27.3	26.5	28.2
ave wd freq(en)	84	66	61	61	**64**	27.8	25.4	27.4	25.8	**27.9**
ave wd freq(deen)	62	57	84	85	**67**	27.3	27.4	28.3	26.9	**28.2**
sent len(de)	78	118	67	56	83	26.6	28.1	27.2	26.4	27.6
sent len(en)	151	59	67	125	196	27.6	25.1	25.6	27.1	27.7
sent len(deen)	113	189	79	68	195	27.0	26.3	26.3	23.9	27.7

Table 2: Performance of curriculum learning strategies with initial learning rate 0.0002. Training time is defined as in Table 1. Bold numbers indicate models that win on training time with comparable (difference is less or equal to 0.5) or better BLEU compared to the baseline.

	Training Time (thousand batches)					**Test BLEU** (best)				
baseline	79					29.95				
	default	*reverse*	*boost*	*reduce*	*noshuffle*	*default*	*reverse*	*boost*	*reduce*	*noshuffle*
one-best score	**59**	**69**	48	92	112	**30.1**	**29.9**	28.3	28.9	30.4
max wd freq (de)	85	103	**69**	118	**43**	25.9	29.6	**30.7**	25.8	**29.6**
max wd freq (en)	148	80	166	49	158	27.0	29.6	28.4	**29.5**	29.9
max wd freq (deen)	84	**61**	**75**	67	93	29.5	**31.5**	**31.1**	27.9	27.2
ave wd freq (de)	79	**51**	73	88	58	27.3	**30**	27.6	27.1	21.3
ave wd freq (en)	**72**	71	146	61	74	**29.9**	28.4	23.3	25.2	29.4
ave wd freq (deen)	81	47	54	58	71	29.9	28.4	28.5	28.3	29.3
sent length (de)	49	126	88	85	**74**	27.0	30.3	29.3	27.8	**31.0**
sent length (en)	101	52	70	49	114	29.0	27.6	24.2	26.9	30.2
sent length (deen)	155	148	170	95	86	29.4	30.7	30.5	29.6	29.5

Table 3: Performance of curriculum learning strategies with initial learning rate 0.0008.

word frequencies were found to be useful ordering criteria by Zhang et al. (2017). However, their experiments are not directly comparable. They were limited to a single training epoch and use a different training strategy, which is closest to our *noshuffle* schedule. With that schedule, our deen sentence length curricula also outperform the baseline in early training stages, but the baseline catches up and outperforms by convergence time. We also note that the conclusions about the *reduce* stated by Zhang et al. (2017) do not hold true for our dataset and curriculum schedules. Specifically, this schedule provides no improvement in training time. (Table 2 and 3).

These results highlight the benefits of viewing curriculum learning broadly, and of curriculum strategies beyond the initial "easy samples first" hypothesis. Interestingly, the *default* and *reverse* schedules can yield close performance, and forcing data shards to be explored in order (*noshuffle*) does not improve over the *default* sampling schedule.

Table 1 further illustrates how curriculum training in NMT is sensitive to hyperparameters. We change the curriculum update frequency (mini-batches) and notice that while the validation set BLEU ramps up quickly as the number of mini-batches is increased between curriculum updates, the convergence time shows no clear trend and the validation BLEU at convergence is the same.

To sum up, our extensive experiments show that curriculum learning can improve convergence speed, but the choice of difficulty criteria is key: vocabulary frequency performs as well as the more expensive one-best score, and sentence length does not help beyond early training stages. No single curriculum schedule consistently outperforms the others, and results are sensitive to other hyperparameters such as initial learning rate and curriculum update frequency.

8 Conclusion

We investigated whether curriculum learning is effective in speeding up the training of complex neural network models such as those used in neural machine translation (NMT) on a German-English TED translation task. NMT is a good test case for curriculum learning as training is prohibitively slow and much patience is required to reach good performance. While the impact on other language

pairs and datasets remains to be studied, we contribute an extensive exploration of curriculum design in controlled settings. We adopt a probabilistic view of curriculum learning, implemented on top of a state-of-the-art NMT toolkit, in order to enable a flexible evaluation of the impact of various curricula design. Our contribution is an extensive exploration of various ways to design the curriculum, both in terms of the difficulty criteria and the curriculum schedule. Our conclusions can be interpreted both positively and negatively: Our results demonstrate curriculum learning can be an effective method for training expensive models like those in NMT, as 20 of the 100 curricula tried improved convergence speed at no loss in BLEU, and that "easy to hard" is not the only useful sample ordering strategy. However, careful design of the specific curriculum hyperparameters is important in practice.

References

Hadi Amiri, Timothy Miller, and Guergana Savova. 2017. Repeat before forgetting: Spaced repetition for efficient and effective training of neural networks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2401–2410.

Amittai Axelrod, Xiaodong He, and Jianfeng Gao. 2011. Domain adaptation via pseudo in-domain data selection. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 355–362. Association for Computational Linguistics.

Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. *arXiv:1206.5533 [cs]*.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, Montreal, Quebec, Canada. ACM.

Cynthia A Brewer. 2006. Basic mapping principles for visualizing cancer data using geographic information systems (gis). *American journal of preventive medicine*, 30(2):S25–S36.

Maria Chrysochoou, Kweku Brown, Geeta Dahal, Catalina Granda-Carvajal, Kathleen Segerson, Norman Garrick, and Amvrossios Bagtzoglou. 2012. A gis and indexing scheme to screen brownfields for area-wide redevelopment planning. *Landscape and Urban Planning*, 105(3):187–198.

Volkan Cirik, Eduard Hovy, and Louis-Philippe Morency. 2016. Visualizing and Understanding Curriculum Learning for Long Short-Term Memory Networks. *arXiv:1611.06204 [cs]*.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA. ACM.

Patrick Doetsch, Pavel Golik, and Hermann Ney. 2017. A comprehensive study of batch construction strategies for recurrent neural networks in mxnet. *arXiv preprint arXiv:1705.02414*.

Kevin Duh. 2018. The multitarget ted talks task. http://www.cs.jhu.edu/~kevinduh/a/multitarget-tedtalks/.

Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. 2017. Automated Curriculum Learning for Neural Networks. In *International Conference on Machine Learning*, pages 1311–1320.

Eva Hasler, Adrià de Gispert, Felix Stahlberg, Aurelien Waite, and Bill Byrne. 2017. Source sentence simplification for statistical machine translation. *Computer Speech & Language*, 45:221–235.

Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. 2017. Sockeye: A toolkit for neural machine translation. *arXiv preprint arXiv:1712.05690*.

George F. Jenks. 1997. *Optimal Data Classification for Choropleth Maps*. Dept. Geography, Univ. Kansas.

Viacheslav Khomenko, Oleg Shyshkov, Olga Radyvonenko, and Kostiantyn Bokhan. 2016. Accelerating recurrent neural network training using sequence bucketing and multi-GPU data parallelization. In *IEEE First International Conference on Data Stream Mining & Processing (DSMP)*, pages 100–103. IEEE.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Eliyahu Kiperwasser and Miguel Ballesteros. 2018. Scheduled Multi-Task Learning: From Syntax to Translation. *Transactions of the Association for Computational Linguistics*, 6(0):225–240.

Tom Kocmi and Ondrej Bojar. 2017. Curriculum Learning and Minibatch Bucketing in Neural Machine Translation. In *Recent Advances in Natural Language Processing (RANLP)*.

Philipp Koehn and Rebecca Knowles. 2017. Six Challenges for Neural Machine Translation. In *Workshop on Neural Machine Translation*, Vancouver, BC.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Robert E. Schapire. 2002. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*.

Rico Sennrich, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nadejde. 2017. Nematus: A Toolkit for Neural Machine Translation. In *Proceedings of the Demonstrations at the 15th Conference of the European Chapter of the Association for Computational Linguistics*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725. Association for Computational Linguistics.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2010. From Baby Steps to Leapfrog: How "Less is More" in Unsupervised Dependency Parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 751–759.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Brian MacWhinney, and Chris Dyer. 2016. Learning the curriculum with bayesian optimization for task-specific word representation learning.

Rui Wang, Masao Utiyama, and Eiichiro Sumita. 2018. Dynamic Sentence Sampling for Efficient Training of Neural Machine Translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 298–304.

Marlies van der Wees, Arianna Bisazza, and Christof Monz. 2017. Dynamic Data Selection for Neural Machine Translation. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1400–1410.

Dakun Zhang, Jungi Kim, Josep Crego, and Jean Senellart. 2017. Boosting neural machine translation. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 271–276, Taipei, Taiwan. Asian Federation of Natural Language Processing.

A Supplementary Material

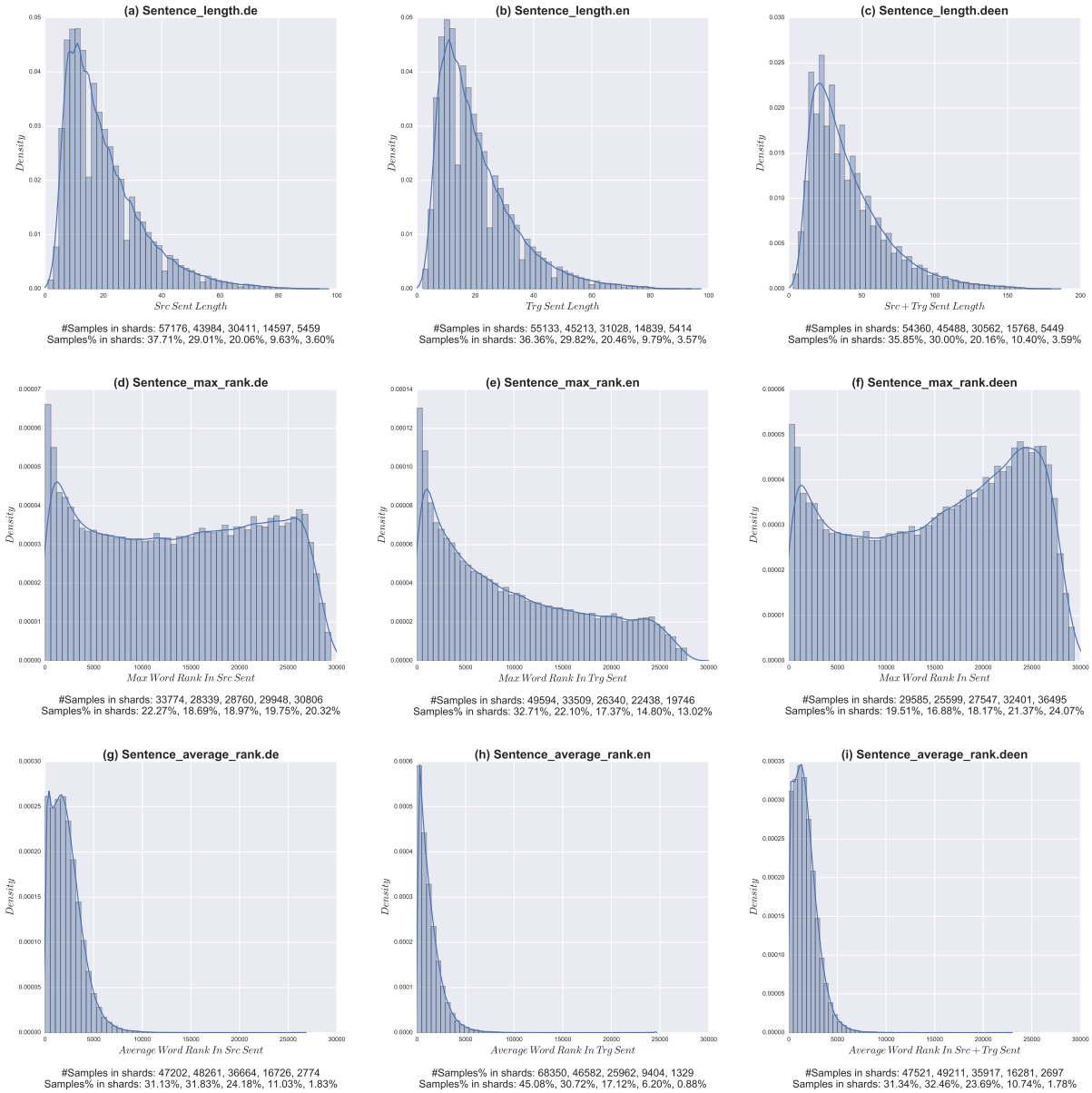

Figure 6: Statistics on GE-EN TED Talks training set (151,627 samples in total) scored by different difficulty criteria. We split the training data into 5 shards. Bucketing results using Jenks Natural Breaks classification algorithm are shown below each subplot, starting from easiest shard to harder shards.

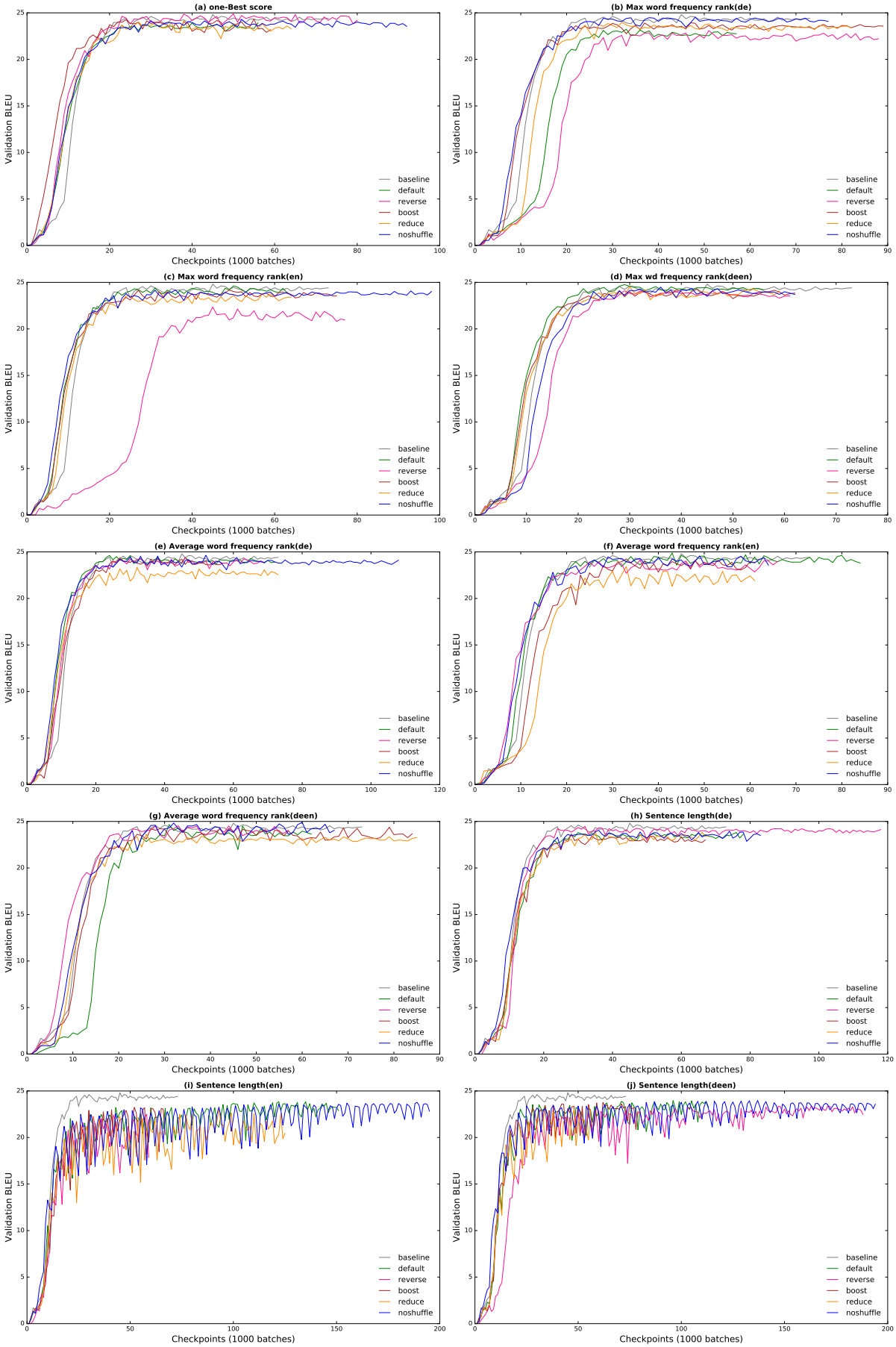

Figure 7: Validation BLEU curves with initial learning rate 0.0002 for different sample ranking criteria.

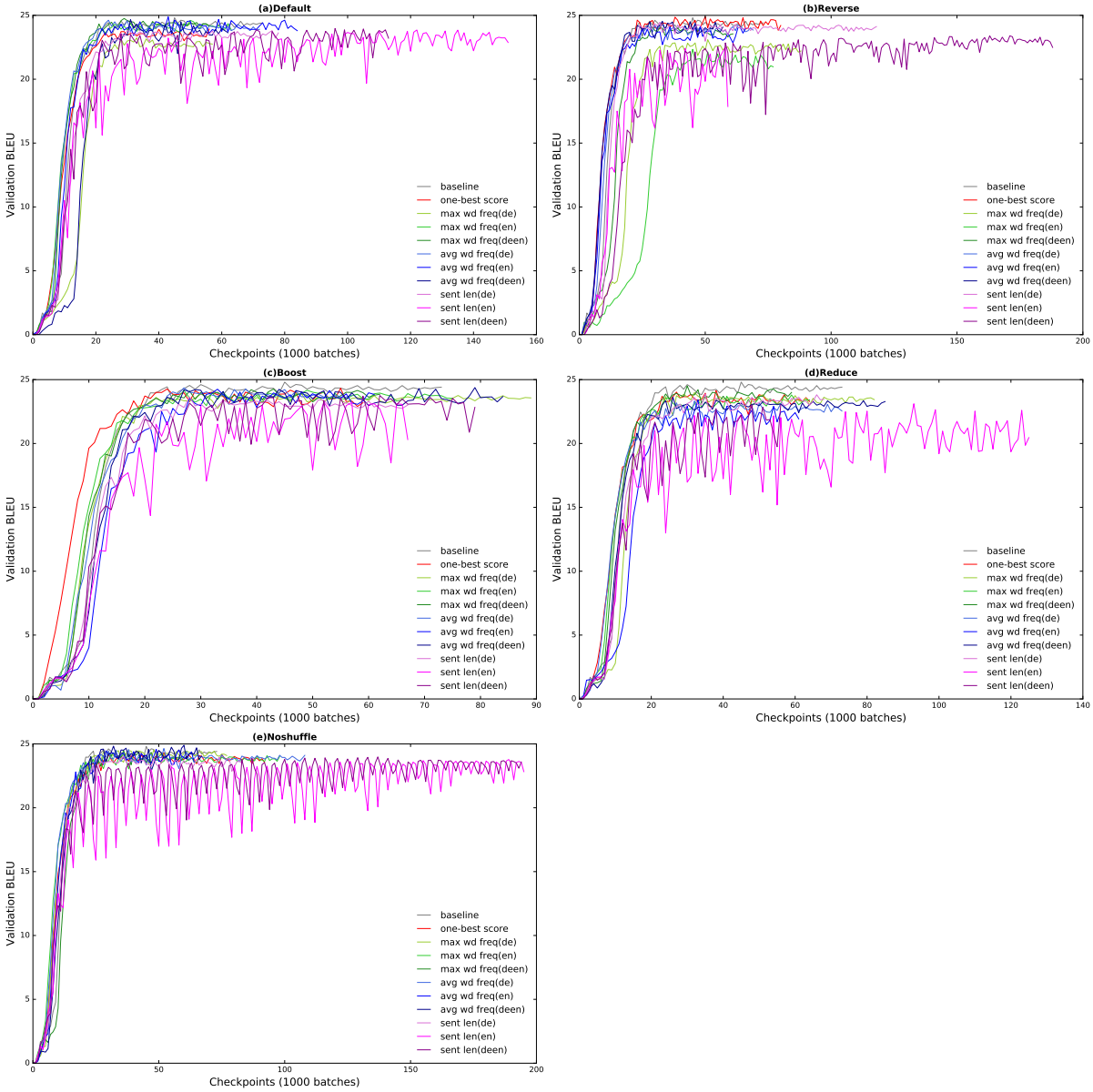

Figure 8: Validation BLEU curves with initial learning rate 0.0002 for different curriculum schedules.

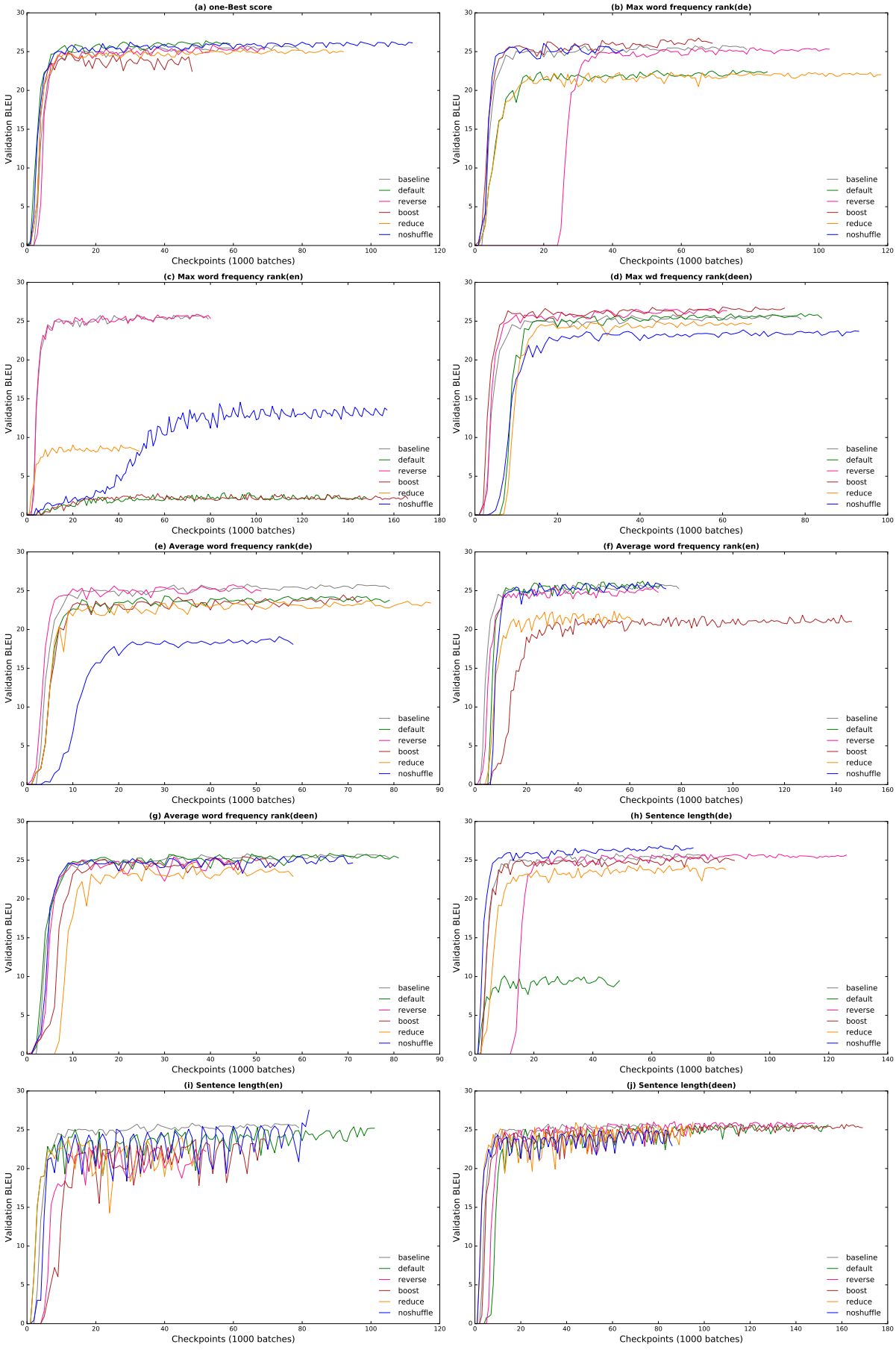

Figure 9: Validation BLEU curves with initial learning rate 0.0008 for different sample ranking criteria.

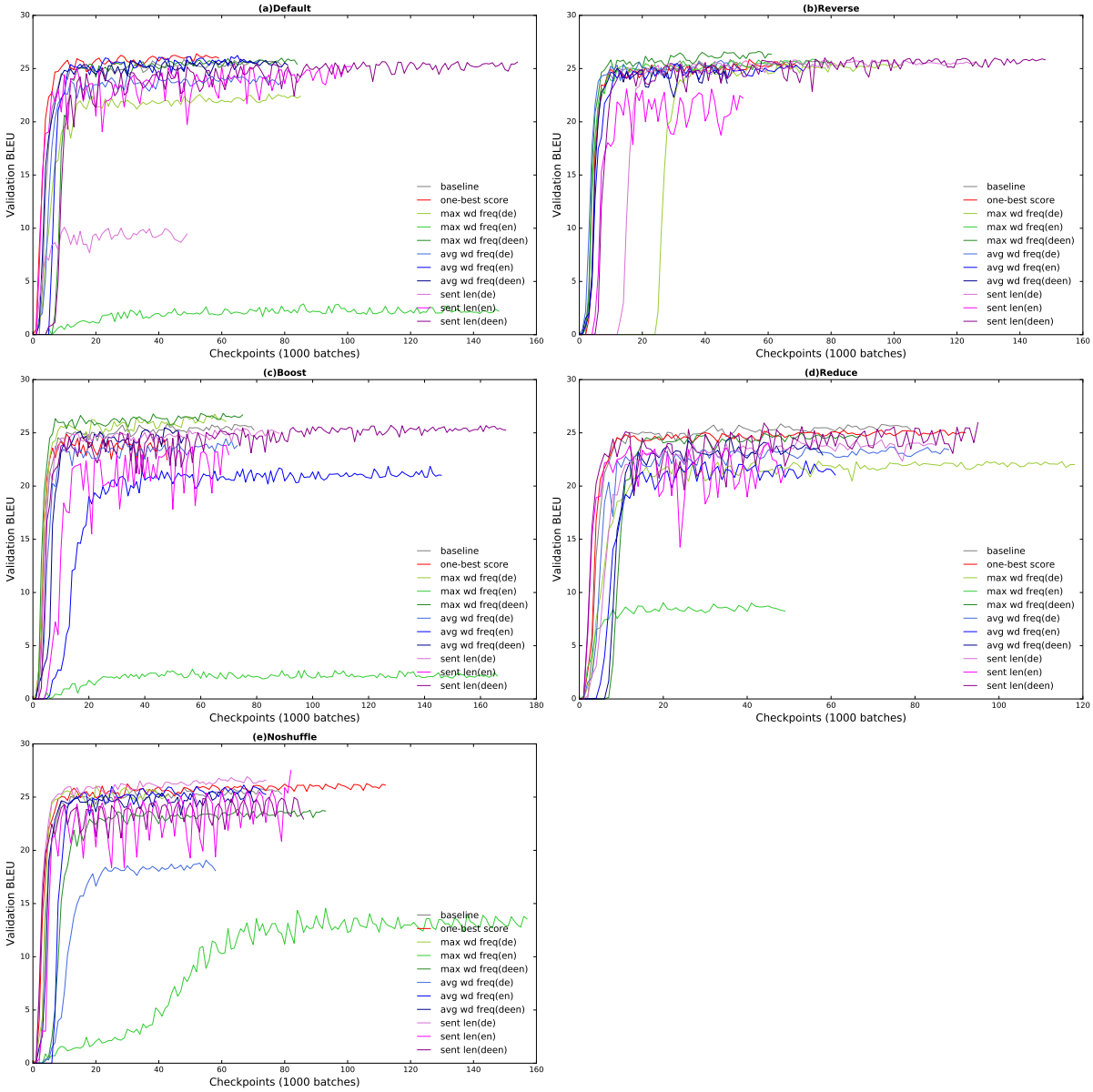

Figure 10: Validation BLEU curves with initial learning rate 0.0008 for different curriculum schedules.

baseline	2.84				
	default	*reverse*	*boost*	*reduce*	*noshuffle*
one-best score	**7.1**	**9.2**	**14.7**	**7.8**	**7.8**
max wd freq(de)	2.0	1.7	**4.6**	1.6	**7.8**
max wd freq(en)	**7.9**	0.8	**8.1**	**5**	**10.8**
max wd freq(deen)	**4.3**	2.5	**4.2**	**2.9**	2.2
avg wd freq(de)	**6.5**	**4.1**	**5.5**	**7.5**	**8.8**
avg wd freq(en)	2.7	**6.8**	2.2	2.6	**5.8**
avg wd freq(deen)	1.6	**8.7**	**2.9**	1.7	**3.7**
sent len(de)	2.4	**3.0**	**2.9**	1.6	**4.6**
sent len(en)	**3.3**	**3.3**	2.5	2.1	**5.1**
sent len(deen)	2.0	2.2	2.0	2.0	**4.3**

Table 4: Decoding performance of different curriculum learning models at the 7th checkpoint with initial learning rate 0.0002.

baseline	25.1				
	default	*reverse*	*boost*	*reduce*	*noshuffle*
one-best score	**28.6**	3.7	**26.5**	**26.3**	**27.8**
max wd freq(de)	18.8	0.0	**27.9**	18.6	**29.1**
max wd freq(en)	0.4	**26.7**	0.0	8.5	0.7
max wd freq(deen))	2.0	**28.3**	**28.3**	0.2	5.2
avg wd freq(de)	24.1	**28.8**	23.3	23.1	2.3
avg wd freq(en)	18.1	21.5	1.9	9.7	4.9
avg wd freq(deen)	**25.2**	**26.4**	18.5	2.0	**26.4**
sent len(de)	9.9	0.0	24.3	17.6	**30.0**
sen len(en)	**26.6**	18.6	5.3	**26.1**	24.2
sent len(deen)	1.1	14.4	24.1	**26.0**	24.2

Table 5: Decoding performance of different curriculum learning models at the 7th checkpoint with initial learning rate 0.0008.