Karakasidis, Georgios; Kurimo, Mikko; Bell, Peter; Grósz, Tamás

Comparison and analysis of new curriculum criteria for end-to-end ASR

Published in:
Speech Communication

DOI:
10.1016/j.specom.2024.103113

Published: 01/09/2024

Document Version
Publisher's PDF, also known as Version of record

Speech Communication 163 (2024) 103113

Contents lists available at ScienceDirect

Speech Communication

journal homepage: www.elsevier.com/locate/specom

Comparison and analysis of new curriculum criteria for end-to-end ASR

Georgios Karakasidis [a,b,*], Mikko Kurimo [a], Peter Bell [b], Tamás Grósz [a]

[a] Aalto University, Otakaari 1, 02150, Espoo, Finland
[b] The University of Edinburgh, 10 Crichton St, EH8 9AB, Edinburgh, United Kingdom

ARTICLE INFO

Keywords:
Curriculum learning
Speech recognition
ASR
End to end
Deep learning

ABSTRACT

Traditionally, teaching a human and a Machine Learning (ML) model is quite different, but organized and structured learning has the ability to enable faster and better understanding of the underlying concepts. For example, when humans learn to speak, they first learn how to utter basic phones and then slowly move towards more complex structures such as words and sentences. Motivated by this observation, researchers have started to adapt this approach for training ML models. Since the main concept, the gradual increase in difficulty, resembles the notion of the curriculum in education, the methodology became known as Curriculum Learning (CL). In this work, we design and test new CL approaches to train Automatic Speech Recognition systems, specifically focusing on the so-called end-to-end models. These models consist of a single, large-scale neural network that performs the recognition task, in contrast to the traditional way of having several specialized components focusing on different subtasks (e.g., acoustic and language modeling). We demonstrate that end-to-end models can achieve better performances if they are provided with an organized training set consisting of examples that exhibit an increasing level of difficulty. To impose structure on the training set and to define the notion of an easy example, we explored multiple solutions that use either external, static scoring methods or incorporate feedback from the model itself. In addition, we examined the effect of pacing functions that control how much data is presented to the network during each training epoch. Our proposed curriculum learning strategies were tested on the task of speech recognition on two data sets, one containing spontaneous Finnish speech where volunteers were asked to speak about a given topic, and one containing planned English speech. Empirical results showed that a good curriculum strategy can yield performance improvements and speed-up convergence. After a given number of epochs, our best strategy achieved a 5.6% and 3.4% decrease in terms of test set word error rate for the Finnish and English data sets, respectively.

1. Introduction

Automatic Speech Recognition (ASR) refers to the task of mapping an audio signal to a sequence of words. This has traditionally been challenging due to the high dimensionality of the input features (audio signal) and the outputs (word sequences). In addition, there is also a time dependency among the inputs, making the problem even more complex. The importance of good ASR systems has become more prominent as people gradually adopt the use of commercial applications such as voice assistants. Furthermore, these systems have been proven beneficial to people with disabilities that need to communicate with a machine (Terbeh et al., 2013; Guo et al., 2020).

The recent advances in Deep Learning (DL) have significantly affected the field of ASR. In the past, the sequential structure of speech and text, and the domain difference between the two, made it challenging to use deep neural networks (DNNs) for this task. As more research was devoted to this topic, and as computational resources became more abundant, both the academic and industry sectors started using end-to-end (E2E) ASR architectures that rely on a single DNN for the whole speech-to-text pipeline (for example, the Recurrent Neural Network (RNN) Transducer (Graves, 2012; Graves et al., 2013) and the attention-based encoder–decoder (Chan et al., 2016) architectures) . Prior to that, Hidden Markov Model (HMM)-based architectures (and most notably DNN-HMMs) were used to train ASR systems. Even though these models are powerful (and can still outperform end-to-end DNNs in several tasks Rouhe et al., 2021), their modeling assumptions, along with the fact that they consist of separate models that need to be separately optimized, make them harder to implement effectively (Wang et al., 2019). On the other hand, end-to-end systems have a more straightforward architecture, but they typically require much more data and computational resources to work effectively, while

* Corresponding author at: The University of Edinburgh, 10 Crichton St, EH8 9AB, Edinburgh, United Kingdom.
E-mail addresses: g.karakasidis@sms.ed.ac.uk (G. Karakasidis), mikko.kurimo@aalto.fi (M. Kurimo), peter.bell@ed.ac.uk (P. Bell), tamas.grosz@aalto.fi (T. Grósz).

https://doi.org/10.1016/j.specom.2024.103113
Received 2 May 2023; Received in revised form 15 March 2024; Accepted 27 July 2024
Available online 31 July 2024

they are also slower to train. To increase the convergence speed of DNNs and improve their performance, researchers have come up with a multitude of different techniques, one of which is *Curriculum Learning* (CL) (Bengio et al., 2009).

Empirical evidence has shown that humans are more likely to learn a new task effectively when provided with a clear curriculum (Newport, 1990; Turkewitz and Kenny, 1982). This has been shown to aid the learning process by providing the learners with a planned sequence of instructions that lead to structured learning and the building upon already acquired knowledge. A good example of that are school curricula where each class has been designed so that the taught material is based on what the students already know. The teachers are assigned the task of arranging the material in a way so that it reflects the increasing level of *difficulty*, while, on top of that, they also decide the *pace* by which the material is presented to the students. Both the *pace* and the notion of *difficulty* are typically adjusted to match the capabilities of the students (e.g., a high-school class consisting of child prodigies would be expected to cover their learning material faster than any other class). Teachers typically learn to create such a curriculum through a combination of experience and observations.[1] The benefits of this approach have also been tested in the field of machine learning (ML) and is the main intuition behind CL.

CL is already a proven technique used in many ML sub-fields (Wang et al., 2021; Kuznetsova et al., 2022; Penha and Hauff, 2019). Even though it shares many similarities with other ML concepts, such as transfer learning (Bengio et al., 2009) and active learning (Wang et al., 2021), it differs in that it aims to accelerate convergence and improve the performance of a model without necessarily trying to utilize external knowledge or learn from fewer labeled data (Wang et al., 2021). In simple terms, given a fixed set of training data, the goal of CL is to find an optimal ordering of it, which will either remain static during training or will dynamically adapt to the strength of the model. In this work, we will propose multiple ways to calculate the difficulty of a training sample and analyze how a good ordering and pacing can influence the performance and convergence speed of E2E systems. We will primarily focus on the applications of CL for ASR, but our core methodology can be easily extended to other tasks, too. Our experiments are performed on two corpora: one of spontaneous Finnish speech and one of planned English speech. The results render CL as an extremely beneficial technique for ASR that, when applied correctly, can increase a network's accuracy and speed-up convergence.

A subset of these experiments was performed in our previous work (Karakasidis et al., 2022), which empirically proved the effectiveness of certain CL strategies in training Finnish ASR systems. In this work, we follow the same principles and introduce new training strategies that modify the pace by which models are trained. Additionally, we show how textual information can be included in the calculation of an utterance's difficulty. These additions aim to prove the flexibility of curriculum learning. The motivation of this work is to establish CL as an efficient way to improve the performance of ASR systems and (potentially) speed up their convergence. Lastly, this work contains a more thorough analysis of how CL managed to improve the underlying ASR models.

The remainder of this paper is organized as follows. First, Section 2 provides a literature review and analyzes related works. Next, Section 3 formally defines CL as a combination of scoring and pacing functions, and proposes a variety of different training strategies. After that, Section 4 contains the experiments conducted in this work, starting with an overview of the data sets used, followed by the neural network architecture and the final results. Next, Section 6 discusses the obtained results while Section 7 concludes the main points of this work.

[1] This teacher-student analogy should not be confused with the teacher-student transfer learning technique that is commonly used in deep learning.

2. Related works

The fact that organized and structured learning leads to better and faster accumulation of knowledge was observed a long time ago. The famous *Pavlov's dogs* experiment, conducted during the early 20th century, can be considered one of the oldest scientific applications of curriculum learning. In this paradigm, the Russian physiologist Ivan Pavlov studied the learning process of dogs by measuring their salivation levels given different stimuli (Pavlov, 2010). Among other findings, he concluded that the dogs learned to associate the ringing of a bell with the serving of their meal after several repetitions of the same feeding process. This implies that the dogs were able to gradually identify a *pattern* (which is similar to what ML systems are expected to achieve) by following a curriculum.

The idea of CL gained popularity after the publication of Elman (1993), which introduced the notion of *starting small* through a study that laid on the intersection of cognitive sciences and machine learning. The main idea is to first start teaching easier aspects of the task in question (or easier sub-tasks) and then gradually increase the difficulty level. Early approaches focused on static solutions that used prior knowledge to define a fixed ordering of the data before training (Bengio et al., 2009). Such strategies depended heavily on the quality of prior knowledge, which made them prone to generalization issues (Jiang et al., 2015). More recent works have pointed out the importance of adaptive curriculum learning where the ranking of the training samples depends on the competence of the student at a given time (Jiang et al., 2015; Zhou and Bilmes, 2018; Hacohen and Weinshall, 2019; Zhou et al., 2020; Kuznetsova et al., 2022).

Regardless of the underlying methodology, the positive effects of CL have been emphasized in a range of works. In particular, Bengio et al. (2009) concluded that CL can speed up convergence and increase the quality of the local minima obtained. The same paper also points out that pre-training with a curriculum strategy shares many similarities with unsupervised pre-training regarding the positive effects of these methods on initializing and updating the trainable parameters. In particular, the results of this work indicated that the right CL strategy can:

1. speed up convergence, since the learner spends less time being trained on difficult or noisy inputs (by being gradually introduced to them), and
2. help find a better optimum in cases where the learner is stuck in a low-performing local optimum.

In Graves et al. (2017) and Platanios et al. (2019), the authors observed significant training time improvements when a curriculum was followed, while Zhou et al. (2020) and Zhu et al. (2022) noticed that training with certain CL strategies was less computationally expensive compared to traditional optimization. Regarding the second point, it is important to mention that CL does not change a model's global optimum (Hacohen and Weinshall, 2019), but rather assists optimization into converging to better solutions. This finding was also observed in Zaremba and Sutskever (2015), where CL was applied in the task of program evaluation. The results demonstrated that the right curriculum strategy can improve performance in tasks that cannot be optimized with standard Stochastic Gradient Descent (SGD).

In the task of information retrieval (IR), the authors of Penha and Hauff (2019) and Zhu et al. (2022) noticed that curriculum learning helped guide their models towards better local optima, leading to an increase in retrieval effectiveness. More specifically, the former work noticed significant improvements in the performance of neural ranking models by simply reordering the instances in the training set. In a similar manner, the latter work employed a dual CL approach that first ranked positive examples from easy to hard, and then oppositely ranked the negative examples. By doing this, their IR models managed to more effectively separate the positive examples from the negative ones.

In Shi et al. (2015), the authors used CL to get a better-performing RNN language model (RNN-LM). In particular, they experimented with CL strategies that first focus on generic textual data and then gradually move towards task-specific data. Each strategy is progressively increasing the influence of more complex examples during training. The results of this publication pointed out that besides improving performance, CL can be used to tweak an RNN-LM to emphasize certain domain-specific patterns (e.g., words typically present in a particular domain/topic). In addition, it was observed that curriculum learning consistently improved the baseline performance, even when the architecture of the RNN-LM was changed. Since language modeling is a task related to ASR (primarily due to its sequential nature), these results give promising prospects for the application of CL in ASR. Other works in natural language processing (Platanios et al., 2019; Tay et al., 2019) have pointed out similar benefits as a result of applying CL.

These findings are also validated in a number of computer vision experiments, such as Jiang et al. (2014), Guo et al. (2018), Hacohen and Weinshall (2019) and Zhang et al. (2019). In particular, Hacohen and Weinshall (2019) concluded that models trained with a CL strategy are more robust, and the majority of them tended to improve the learning accuracy when compared to a baseline. The same paper also contains an extensive analysis of how curriculum learning can be formally defined, regardless of a specific task. They do that by decomposing CL into scoring and pacing functions which they define as two different mechanisms for assigning difficulty scores to utterances and controlling the pace by which new data are presented to the model under training. Due to the straightforwardness of this definition, the approach of Hacohen and Weinshall (2019) was chosen as the basis of our work.

Even though the above works showcased the importance of CL in many DL tasks, they still do not cover how it has so far been applied in speech processing. One could say that CL has a strong representation in this field since most speech-processing frameworks tend to sort audio utterances based on their duration before starting the training process (Povey et al., 2011; Ravanelli et al., 2021; Watanabe et al., 2018). Although, this is mainly treated as a pre-processing step and is not commonly referred to as a CL technique. The main reasons behind choosing this duration-based sorting are:

1. Padding is minimized: By using duration-level sorting, each batch will contain utterances with similar lengths, meaning that they will not need to be excessively padded to match the length of the longest utterance. This results in a decrease in the amount of required memory while it also speeds up matrix computations.
2. Faster Gaussian Mixture Model (GMM) training: In the traditional HMM-based systems, the initial (monophone) GMMs need to learn very simple initial distributions for the phones (Gales and Young, 2008; Novoa et al., 2018). In addition, the speech-to-text alignment procedure is typically quite expensive for longer utterances. For these reasons, it is customary to use a subset of the initial training data that contains only the shortest utterances (Povey et al., 2011). As the models become more and more complex, the amount of training data increases.

Even though the benefits of this duration-based curriculum strategy are important in ASR (Ravanelli et al., 2021), this method has not been shown to bring any performance improvements (see Karakasidis et al. (2022)). That is mainly because the duration of an audio file is not necessarily a good indicator of its difficulty. To address that, the authors of Braun et al. (2017) came up with a novel CL technique named accordion annealing (ACCAN). It uses the Signal-to-Noise Ratio (SNR) values of each training example to measure the difficulty level of each utterance. The corresponding ASR system is first trained on clean data (e.g., SNRs above 30db) and is gradually introduced to noisier data with lower SNRs. The authors based their experiments on the expectation that neural networks tend to perform better on the SNR they are trained on Yin et al. (2015). Using the proposed SNR-based CL strategy, the ASR system can first learn to optimize the clean audio and then use that knowledge to improve its performance on the noisy data. The experiments showed that the noise robustness of the ASR system is increased without having to rely on complex pre-processing frameworks to remove noise.

A similar SNR-based approach was followed in Ng et al. (2022) and Higuchi et al. (2021), which used CL for small footprint and noise-robust keyword spotting (KWS). The results pointed out that CL can help improve the performance of KWS models. Ng et al. (2022) also trained a more complex and resource-heavy transformer model without utilizing CL and found out that its accuracy was on par with the lightweight model even though the latter was much smaller in size.

It is clear that CL has the potential to bring benefits to a variety of ML tasks. Another advantage is that it is very flexible in how it can be applied. This means it can easily be combined with other techniques, such as reinforcement learning (RL). For example, in the works of Kuznetsova et al. (2021) and Kala and Shinozaki (2018), the authors utilized the one-armed bandit and policy-based algorithms (both are RL methods) to maximize the reward that certain candidate curricula had to offer. In both experiments, the combination of CL and RL brought improvements in terms of Word Error Rate (WER), while Kuznetsova et al. (2021) also noticed a decrease in training times.

In conclusion, CL is a robust method that has been shown to bring improvements in many ML tasks (Bengio et al., 2009; Wang et al., 2019; Hacohen and Weinshall, 2019) while also being easy to integrate into existing training pipelines. In this work, we will analyze different strategies to create efficient curricula for E2E ASR and discuss their advantages and disadvantages.

Compared to the previously discussed research, the main contributions of this work are in that we formalize CL as a combination of scoring and pacing functions, and we define new CL strategies for ASR. We show that both audio and textual information can be used in the calculation of each utterance's difficulty score, and we analyze the efficiency and performance of each strategy under different training conditions. Our experiments prove that CL has the potential to reduce error rates for ASR models, while certain strategies have the ability to increase convergence speed.

3. Methods

This section will present and formally define all relevant CL approaches used in this work. As already mentioned, CL lacks a widely accepted definition and its applications are typically limited to only certain aspects of it. Here, we follow the proposition of Hacohen and Weinshall (2019), where CL is characterized by the combination of two functions: a *scoring* and a *pacing* function. The role of these two is to depict what constitutes a difficult training example and to define the rate by which the examples will be presented to the model under training. The final strategy that a CL algorithm uses depends on what combination of those two functions was utilized.

Revisiting the teacher-student analogy presented in Section 2, one could connect scoring functions (SFs) with the experience of teachers that allows them to create a curriculum of increasing difficulty. For example, experienced teachers have a better understanding of what differentiates a difficult subject from an easy one. This way they are more aware of the parts that they should emphasize and focus on whilst teaching. Transferring this to the ML world, a good CL strategy should have an efficient methodology to assign scores to the training examples, which is the purpose of scoring functions. Creating such functions is not a straightforward task, especially for speech recognition, where both inputs and outputs consist of high-dimensional, complex features.

Besides the order of the presented material, teachers are also responsible for deciding the pace by which the students will be introduced to new material. This should be done in a way so that they first learn the easy subjects and only after feeling comfortable, move towards more complex ones. Our implementation of pacing functions (PFs) slightly

differs from the one described in Hacohen and Weinshall (2019), but the main idea behind both approaches is that the training set is split into several subsets through the use of some heuristics, and a different subset is presented to the network at each epoch.

Before going into further details about how these two types of functions operate, it would be helpful to formally define CL as follows:

Definition 1 (*Curriculum Learning (CL)*)**.** Curriculum learning aims to improve the performance and convergence speed of ML models, by using a combination of scoring and pacing functions. These functions control the order and pace, respectively, by which the training data are provided to a model.

So far, we have primarily discussed CL methods that sort the training data sets in ascending order (i.e., from the easiest to the hardest example). Even though this is the main focus of this work, we will also present the effect of reversing the order and starting training a model with difficult examples before moving to easier ones. This method was referred to as *Anti CL* in Ranjan and Hansen (2018) where the performance of the corresponding models were found to be very similar to the ones of the models trained with ascending order. Through empirical results (see Section 5), this finding does not seem to apply to E2E ASR models since, when trained with a reverse ordering, they consistently underperform compared to the baseline and their ascending counterparts. From now on we will refer to this method as **reverse curriculum**.

3.1. Curriculum creation strategies

We can think of CL as a way to provide guidance to a ML model. To create an efficient CL algorithm, it is required to efficiently estimate the difficulty of each training example. The most common approaches for doing so can be grouped into the following three classes:

1. The **metadata-based approach** estimates the difficulty scores based on some meta-information (e.g., the duration of an audio sample or its SNR) at the beginning of the training process. This is a *static* solution, meaning that the curriculum is established before training, and the order is kept fixed throughout training.
2. The **transfer learning approach**[2] relies on an external, already trained teacher model that is used to infer the training data before the first epoch. The assumption is that the external model should recognize the easy training examples with fewer errors than the difficult ones. This method utilizes the outputs of the teacher to sort the utterances at the beginning of the training process. This makes the transfer learning approach a static approach, as well.
3. The **adaptive approach** proposes to sort the examples adaptively using feedback from the student neural network under training. This approach addresses the fact that the difficulty of the examples (as perceived by the model) could change as the training progresses and was first proposed in Kumar et al. (2010). One can view this approach as dynamically adjusting the curriculum to the current capabilities of the model.

To continue the analogy of CL with human learning, the *transfer learning* approach is equivalent to a teacher network trying to help a student, while in the *adaptive* approach the student is an autodidact trying to adapt to the learning difficulty. Finally, the *metadata* approach consists of a non-adaptive teacher that simply follows guidelines and there is no knowledge transmission.

[2] Even though the transfer learning CL approaches are not the same as the most commonly known transfer learning technique in DL, these two methods still share some similarities. In particular, in both cases, the initial tasks are used to guide the learner (Bengio et al., 2009).

The aforementioned categorization of CL strategies and the distinction between scoring and pacing functions can be used regardless of the underlying ML task. This terminology provides a good basis to further develop CL for the task of ASR. Next, we will provide the mathematical formulation behind scoring functions and we will proceed by defining certain ASR-specific SFs and PFs, that can be easily adjusted to work with other ML tasks, too.

3.2. Scoring functions

Let $\mathcal{D}$ be the set of all training examples of the form $\mathcal{D} = (x^i, y^i)_{i=1}^{N}$ where x^i, y^i are the features and labels, respectively, and N is the total number of training examples. In addition, the target classifier is assumed to be trained on mini-batches of size b. The set of all mini-batches that comprise $\mathcal{D}$ can then be denoted as $[\mathbb{B}_1, \dots, \mathbb{B}_{\mathbb{D}}] = \mathcal{D}$ where each mini-batch $\mathbb{B}_i$ is a subset of $\mathcal{D}$ of length b, and $\mathbb{D}$ denotes the total number of mini-batches.

Commonly, neural networks that use a variant of SGD to update their weights, sample data points $(x^j, y^j)_{j=1}^{b}$ uniformly from $\mathcal{D}$ in order to form mini-batches. When applying CL, though, the data points are not randomly sampled from $\mathcal{D}$, but rather ordered. At each training epoch, the model is expected to receive an ordered set of mini-batches whose size will be $\leq \mathbb{D}$ (the exact number depends on whether a pacing function was used or not (see Section 3.3). If we denote our *scoring function* with $f : \mathcal{D} \to \mathbb{R}$, then the ordering of the examples will happen in such a way so that for a pair of utterances u^i, u^j, the ith utterance has order less than the jth one ($u^i < u^j$) if $f(u^i) < f(u^j)$. Choosing a good scoring function is the primary task that CL tries to solve since f has to meaningfully encode information about the input data (Hacohen and Weinshall, 2019).

There is no limitation on when a scoring function may be applied. As mentioned in Section 2 most of the early works focused on static approaches where the function is applied before the first epoch and the training proceeds on the arranged data set. More recent research overcomes the limitations of this approach by re-ordering the training set at frequent epoch intervals, based on, for example, self-feedback (i.e. taken from the same student model). By doing this, it is clear that the training times could be affected unfavorably since such scores need to be calculated and stored in memory at frequent time intervals (e.g., at each epoch). Although, it shall be highlighted that this work does not focus on the time improvements that other CL approaches seem to offer, but its goal is to examine whether the final performance and/or *convergence speed* is improved. We refer to convergence speed as the number of hours of audio that a network needs to train on, before reaching a certain level of performance.

3.2.1. Types of scoring functions for ASR

Duration-based SFs are the most commonly used scoring functions since they are easy to implement and they minimize padding, resulting in fast training times. In particular, this is a metadata curriculum learning method where the training examples are sorted with respect to their durations. The ordering occurs before the first epoch and remains the same till the end.

A disadvantage of this approach is that an audio's duration is not a good indicator of its difficulty. A short but noisy speech segment can be much harder to decode than a longer, clean one. For this reason, in cases where the training examples resemble real-life environments (which are typically noisy), no performance improvements should be expected from this method. Overall, this is a good curriculum creation approach from a computational point of view, but it cannot guarantee any performance improvement.

Frequency-based SFs compose yet another metadata curriculum learning approach which, in this case, focuses on the text domain. In particular, given an utterance u^i with transcript y_i, a score $f(u^i)$ is assigned as the negative of the mean of the frequencies of each token in y_i. The assumption is that examples with frequent tokens will be easier

to decode since the model is more likely to learn those frequent tokens. A similar approach was followed on Platanios et al. (2019), where the authors used a set of text-level scoring approaches to measure the difficulty of sentences. On the contrary, rare tokens will make the model less confident in its predictions, and hence the corresponding utterances can be considered difficult. Through empirical results, it was observed that taking the average of these values is a good way to aggregate them.

This approach has three different flavors, based on how the textual frequencies are extracted:

1. *Word-level* frequencies: The tokens are considered to be words. Transcripts that contain many frequent words (e.g., *the*, *in*, e.t.c.) are considered easier.
2. *Character-level* frequencies: The tokens are considered to be characters.
3. *Subword-level* frequencies: The tokens are considered to be subword units (see 4.2). This is also referred to as the *token-level* approach.

An obvious disadvantage of this approach is that it does not take into account the audio signal, which is where most of the complexity lies (e.g., noisy environments, stuttering speakers). Of course, utterances that contain difficult or rare words can be difficult to transcribe, too, and frequency-based SFs can help counter such issues. In addition, these kinds of SFs are easy and fast to implement, and due to their similarity to the duration-based SFs, they also lead to close-to-optimal padding since in many cases there is a one-to-one correspondence between duration and sentence length.

Adaptive SFs correspond to an adaptive curriculum learning approach where the training set is re-organized in frequent time intervals based on feedback from the model under training. The basic idea is to adapt the utterances' scores depending on how the model performs on them at any given moment. The intuition behind this is that as a weak model improves through training, it will learn to distinguish new patterns that will help it decode examples that were previously deemed difficult. Adaptive SFs are divided into the following two categories:

1. **Loss-based**: While training, the negative log likelihood (or sequence) loss values of each utterance are stored in memory. These are then used as an indicator of the difficulty of each training example. In particular, it is assumed that low loss values correspond to easy examples, while higher values imply greater difficulty.
2. **Metric-based**: A metric value (WER or CER) of each utterance is used as its difficulty score. Lower WER or CER values mean that the corresponding examples are easy, while higher values imply otherwise. Even though this is an intuitive method, it requires a decoding step before backpropagation and the updating of the parameters. A direct effect of this is that it increases the total training time of the ASR model.

Before the first epoch, the model's weights are randomly initialized, which means that we cannot use it to calculate the utterances scores. Consequently, in our experiments, we chose to employ each utterance's duration as its score for the first epoch, and switch to adaptive SFs for subsequent epochs.

3.2.2. Uniform mixing

During our experiments, we noticed that the metric-based SFs tended to produce models that overfitted to the data (see Section 5). We hypothesize that this occurred due to *excessive guidance*, which is inspired by the real world where teachers sometimes over-indulge their students by easily providing answers and allowing memorization without learning. In the ML world, a similar issue was approached in Zaremba and Sutskever (2015) by introducing some randomness to the ordering of the training set as a post-processing step. In this work, we refer to this technique as **uniform mixing** (UM), which corresponds to infiltrating some hard examples among the easy ones, with the aim of improving the model's generalization capabilities.

The idea is to ensure that the *student* has to put some effort into understanding each example. To achieve that in our case, we proposed a modification of the method described in Zaremba and Sutskever (2015), where we split the training examples into three *difficulty sets* consisting of *easy*, *medium* and *hard* examples. This is done by first applying a SF on the training set so that each example is assigned a difficulty score. We then proceed by ordering the training set based on these scores and splitting it into three parts of equal size. The set that consists of examples with the lowest scores is the *easy* set. The next set consists of utterances considered as *medium*-level in terms of difficulty, while the last set (the one with the highest difficulty scores) is considered the *hard* set. Note that UM does not filter the training data, but rather performs a controlled shuffling.This means that the output of the UM function has exactly the same size as the input set of utterances.

The idea of UM is that the set of easy examples will be augmented with some hard and medium-level ones. In particular, a hyperparameter is used to tune the percentage of "noise" which will be added to the easy set. For example, if this value is 20%, then a fifth of the easy examples will be replaced with a mixture of hard and medium-level examples. To increase the impact of this noise, 60% of these added examples will be hard, and the remaining 40% will be medium-level. These values were tuned during our preliminary experiments and, as a result, were used in all of our UM-based trials.

By following the above procedure the model is exposed to different kinds of data during each training stage, increasing its generalization capabilities. Uniform mixing can be combined with all kinds of SFs, and in our case, we have tested it with both frequency-based and adaptive SFs. In the majority of cases, the resulting models significantly outperformed the "vanilla" application of SFs (i.e., the generalization capabilities of models trained with UM were better compared to when UM was not used) (see 5). From now on, the SFs that use UM to order a training set will be referred to with the abbreviation **UM-SFs**.

3.2.3. Transfer learning-based SFs

As already mentioned, the transfer learning CL approach creates a constant ordering of the training set. This is done by applying a scoring function $f(\mathcal{D}, \mathcal{M}) \in \mathbb{R}$ that accepts a training set $\mathcal{D}$ and an already-trained model $\mathcal{M}$, and uses $\mathcal{M}$ to calculate its scores on $\mathcal{D}$ based on an adaptive SF. For example, f can use the utterance-level losses that $\mathcal{M}$ outputs for each data point in $\mathcal{D}$. This method can also be combined with uniform mixing to inject some randomness into the ordering of the training set.

The pre-trained model $\mathcal{M}$ is also known as the *teacher*, and the assumption is that it will recognize the easy examples with fewer errors than the difficult ones. An advantage of this approach is that the scoring function needs only to be applied once, resulting in reduced training times compared to models optimized with the adaptive CL approach. A disadvantage is that it heavily relies on an external model, which may not be a good teacher. Weak teachers yield close-to-random orderings, which minimize the effect of curriculum learning. On the other hand, it was observed that using very good teachers can confuse the model and lead to overfitting, especially during the first few epochs.

3.2.4. Score post-processing

As mentioned previously, a duration-based sorting of the training set has the advantage of minimizing padding which results in more efficient computations and faster training times compared to all other methods. Another advantage of this approach is that the duration of an audio file is still a (weak) indicator of what level of difficulty may be expected. For example, if a SF assigns the same score on two utterances, the duration is a good way to settle which utterance should be considered easier.

To incorporate this in our score calculation, all scores (for the non-duration-based SFs) are normalized by taking into account the utterances' durations. For example, if $s(u)$ is the original score of utterance u and d_u is its duration, then $s'(u) = \frac{s(u)}{d_u}$ is the normalized score. The intuition is that if u has the same score as another utterance u', but has a longer duration, then it must be easier since despite the theoretically increased complexity of u' (due to its length), the score was the same as u.

Another useful piece of information that can be integrated into an utterance's score is the confidence that an ASR system has in its prediction (here denoted by c_u). For example, if a model is only 50% confident about a transcription then this is a good indicator that the corresponding utterance was hard to decode. Since acquiring the confidences requires a decoding step, they are only incorporated in post-processing when a metric-based adaptive SF is used. Instead of confidence scores, our normalizers use uncertainty values which are defined as the negative of the confidences and are denoted as $n_u = -c_u$.

In the case of this work, we combine the duration and uncertainty information, by first normalizing the latter based on the corresponding durations, then applying min–max normalization, and, finally, combining the information as follows:

$$n_u \leftarrow \frac{n_u}{d_u} \tag{3.1}$$

$$n_u \leftarrow \frac{n_u - min(n_u)}{max(n_u) - min(n_u)} \tag{3.2}$$

$$s(u) \leftarrow \frac{s(u) - min(s(u))}{max(s(u)) - min(s(u))} \tag{3.3}$$

$$f(u) \leftarrow (s(u) + \epsilon) * n_u \tag{3.4}$$

where $s(u)$ now denotes the WER or CER score of utterance u, while ϵ is a threshold value to prevent zeros from appearing in the scores, and $f(u)$ is the final score assigned to u. This sequence of steps results in a score that is always between 0 and 1 (for sufficiently small ϵ) and also takes into account both the duration and confidence values.

3.3. Pacing functions

Pacing functions control the pace by which data are presented to a ML model. Their goal is to allow incremental training by trying to first teach the easy examples before moving to harder ones. PFs usually work by providing a model with easy examples during the first few epochs, and then gradually introducing more difficult examples. For instance, in Kim et al. (2018) the ASR model initially tries to solve a simplified classification problem where it is tasked to recognize only four symbols. After this task is learned, the complete training set is restored and the training proceeds. The following is a formal definition of how PFs are used in this work:

Definition 2 (*Pacing Function (PF)*)**.** A function g is called a pacing function if it transforms a training set D to a sequence of training subsets $\left[S_1, \ldots, S_E\right]$ where E is the total number of epochs for which the model will be trained. At each epoch, $e = 1, \ldots, E$, the model will use the S_e subset instead of the whole training set.

The most straightforward way to acquire these subsets is to first apply a SF on D and then split the ordered training set into K subsets, in a similar manner as done with uniform mixing. The number K is a hyperparameter that controls the frequency by which each training set is used and it shall always be less than or equal to E. For example, if $K = \frac{E}{2}$, then the first subset (containing the easiest examples) will define the difficulty for the first 2 epochs, the second subset for the next 2 epochs, and so on. Instead of using a single subset at each stage, we concatenate it with the subset from the previous stage (i.e. $S^{(i)} \leftarrow S^{(i-1)} + S^{(i)}$ for $i = 2, \ldots, K$). From now on, these functions will be referred to as *Vanilla* PFs (**VPFs**). Such PFs can be theoretically combined with any kind of scoring function. Here, only the combination of VPFs and transfer learning CL approaches will be tested, since they provide static scores that can be calculated before the initialization of the model. In addition, VPFs can also work with UM-SFs, by assuming that the noise injection occurs on the level of the $\left\{S^{(i)}\right\}_{i=1}^{E}$ subsets and not the whole training set.

Another way to apply pacing functions to a curriculum is by randomly selecting a set of examples from the training set. In contrast to VPFs, this approach does not focus on incrementally increasing the average difficulty of each training subset, but rather on the effect of solely increasing the number of training examples provided at each epoch. This category of pacing functions will be referred to as *Subsampling* PFs (**SPFs**). This strategy works by first randomly sampling a subset of the training data, and then sorting it by applying the scoring function to it. This process is repeated at frequent time intervals (e.g. at each epoch), with the sole modification being a gradual increase in the size of each subset as training progresses.

Note that the models trained with this strategy are guaranteed to be trained on the whole data set for at least one epoch. In addition, since SPFs work with random sampling, a subset $S^{(i)}$ (for $i > 1$) may contain examples that do not exist in $S^{(i-1)}$, even though $|S^{(i)}| > |S^{(i-1)}|$. Such pacing functions share many similarities with uniform mixing since, in both cases, the training does not necessarily start from the easiest possible examples in D. The injection of randomness in both of these approaches has proved to be extremely beneficial and corresponding models were shown to be less prone to overfitting and underfitting (see Section 5). This goes against how VPFs work, where each subsequent subset $S^{(i)}$ is always a superset of $S^{(i-1)}$ (i.e. it contains all elements of $S^{(i)}$ plus some additional ones).

Regardless of the underlying approach, the scope of pacing functions is to accommodate incremental learning by gradually introducing new examples of higher difficulty. One way to do that is by discarding the easy examples and using only the harder ones whenever the PF is applied (i.e. each subset $S^{(i)}$ for $i = 1, \ldots, K$ will have the same length, $|S^{(i)}| = |S^{(i-1)}|$, $i = 2, \ldots, K$ with $S^{(i)}$ containing harder examples than its predecessor). However, by doing so, the model is more prone to "catastrophic forgeting", meaning that it is more likely to lose general knowledge as training proceeds (Parisi et al., 2019). Due to that, the default behavior in this work is to re-use the easy examples, which results to gradually increasing the length of each subsequent subset (i.e., $|S^{(i)}| > |S^{(i-1)}|$, $e = 2, \ldots, K$). Both approaches have been implemented in this work and the choice is controlled by a hyperparameter.

4. Experiments

This section contains a range of different experiments on the application of curriculum learning for ASR. In particular, we will provide results for several CL strategies tested on two data sets: *Lahjoita puhetta* (in Finnish) (Moisio et al., 2022), and *Common Voice* (in English) (Ardila et al., 2019). In both cases, the speech recognition models are trained using the Speechbrain toolkit (Ravanelli et al., 2021), which provides a straightforward way to train DNNs for ASR in Python, while it already includes a duration-based ordering of the training data. All experiments were executed on an Nvidia Tesla A100 GPU card, that allows 6912 CUDA cores and has 80 GB of memory.

For reasons discussed in Section 2, the duration-based ordering will be considered as our baseline strategy along with random sorting, which is commonly used for stochastic gradient descent optimization. This is because duration-based ordering is already widely used in ASR and is the default in popular toolkits such as Speechbrain and Kaldi. For each experimental setup, the main baseline will be the one with the lowest word error rate on the test set.

4.1. Data

4.1.1. Lahjoita Puhetta

The Lahjoita puhetta (Donate speech) corpus is a collection of colloquial Finnish speech, gathered through the *Donate Speech* campaign (Moisio et al., 2022). In total, it contains over 3600 h of speech, more than half of which has been transcribed to date. The corpus (which we will refer to as **LP**) includes over twenty thousand speakers spread across all regions of Finland. This way speakers of many different dialects found within the country are well represented in the corpus.

The data set comes with official splits for training, validation, and test sets. These have been created so that all of them have a similar distribution on the topics covered by the speakers. In addition, the test set has been created so that the gender ratio of the speakers is balanced. Specifically, the training set consists of almost 70% female speakers, with the rest being either male (20%) or unknown (10%). In contrast, the test set is comprised of $\sim$ 42% male speakers and $\sim$ 51% female speakers, with the rest being unknown ($\sim$ 7%). This was done to ensure that the test data are closer to the real-world distribution of speakers. More information on the creation of the corpus and its attributes can be found in the original paper (Moisio et al., 2022).

During the campaign, volunteers were asked to explore a topic (out of a predefined list of topics) for as long as they wanted. This has led to utterances of many different durations with a median duration of $\sim$ 41 s and a mean duration of $\sim$ 54 s. Unfortunately, it is known that E2E models are more prone to bad performance when the input (and output) sequences are very long (Chang et al., 2017). In such cases, we can expect memory errors and long training times due to the small batch size that is required to fit such sequences. To counter this issue and make LP suitable for E2E models, we have segmented all long utterances into chunks of at most 10 s, by using the best Finnish ASR system described in Moisio et al. (2022). This is a hybrid HMM-DNN system trained with the Kaldi toolkit (Povey et al., 2011). The latter provides a set of useful scripts to assist with audio segmentation. In our case, the chunks were gathered by detecting silences that appear close to 10 s after the end of the previous chunk and splitting into the corresponding segments.

By performing the aforementioned segmentation, we ended up with a training set of 1377.2 h with a median of 8 s and a mean of 6.9 s. Similarly, the validation and test sets were 8.3 and 8.9 h, respectively, with median/mean durations of 4.7/5 and 4.6/4.9 s each. This adds up to $1377.2 + 8.3 + 8.9 = 1394.4$ h of speech which is almost 14% less than the original amount of data (1620 h). This decrease is due to the removal of silences that occurred throughout the segmentation process. It is also worth mentioning that the segmentation provided by the ASR system could potentially contain mistakes, but according to our observations the chunking worked as expected and mistakes were rare. This approach is common practice when segmenting big audio datasets and a similar approach was also used for the creation of the TED-LIUM corpus (Rousseau et al., 2012).

In addition, to test the effect of our curriculum learning strategies on different sizes of data, we also experimented with subsets of the LP training set. In particular, we have tested three different subsets of different lengths: (a) $\mathbf{LP}_{138}$ of 138 h (10% of the data), (b) $\mathbf{LP}_{414}$ 414 h (30% of the data), and (c) $\mathbf{LP}_{1377}$ hours (the whole set). Results for these splits are presented in Table 4 and are provided only for a subset of our curriculum strategies.

4.1.2. Common Voice (version 8.0)

Common Voice (CV) is a popular data-gathering project, created by Mozilla (Ardila et al., 2019). Its goal is to crowd-source ready-made audio and transcription pairs, by providing volunteering speakers with an intuitive user interface that first shows them an utterance and then asks them to pronounce it. The project is currently available in more than 90 languages from all over the world, while the collected data are available under a public domain license that allows researchers and developers of all kinds to use it for creating speech-to-text software.

In this work, we will focus on the eighth version of the English common voice corpus (from now on denoted as **CV8**). Earlier versions of the data set were known to contain many near-duplicates among the training, validation, and test sets (mostly because the same transcription could be presented to the volunteers on multiple occasions).[3] This tainted the ability to evaluate the generalization capabilities of a model since the test set did not represent the variety of speech found in the real world and it rewarded memorization.

Overall, the data set consists 1375 h of training speech, while $\sim$ 28 and $\sim$ 27 h are present on the validation and test sets, respectively. The duration of the initial utterances are already small (median duration of 5.62 s) and so there was no need for extra segmentation.

4.2. Architecture

For both data sets, the top-level DNN architecture is the same, consisting of an encoder and an autoregressive decoder similar to Chan et al. (2016). The encoder is a Convolutional Neural Network (CNN), followed by a RNN and a Deep (linear) Neural Network (often abbreviated as CRDNN altogether). The decoder is a RNN with attention that accepts the output of the encoder along with the output of an embedding layer that provides a dense representation of the transcripted text. The chosen hyperparameters for this architecture are available on the Github repository (Georgios, 2022). Audio signals of both sets were sampled at 16 kHz, with 400 samples used in each short-time Fourier transform (STFT) and 40 Mel filters (the dimensionality of each final frame). We also performaed mean and variance normalization to the input signals. Fig. 1 depicts a top-level representation of how the training pipeline looks like.

The models are trained using the *Adam* optimizer (Kingma and Ba, 2014) which attempts to accelerate the gradient descent by taking into consideration the momentum of the gradients. Over the past decade, *Adam* has become one of the most widely used optimization techniques, and its implementation is featured in most DL toolkits. On top of that, we incorporated the use of **gradient accumulation** which aggregates the gradients and proceeds to update the parameters after a certain number of minibatches. This is helpful since audio features tend to be complex and high dimensional, making them hard to fit to memory when using big batch sizes, and the use of gradient accumulation helps overcome this issue. In our case, and after performing hyperparameter tuning, we chose to accumulate 4 gradients before each update.

The loss function that is minimized in this work is the *negative log likelihood*, which is also referred to as *sequence to sequence* loss or just *sequence loss*. In addition, the Connectionist Temporal Classification (CTC) (Graves et al., 2006) loss is minimized on the output of the encoder for the first few epochs, depending on the data set and the number of hours in it. Since the CTC loss has to be aggregated with the sequence loss, a new hyperparameter needs to be introduced which defines the importance of each loss. In our case, when the CTC loss is used, it has a weight of 0.5, i.e., the final loss value is the average of the CTC and sequence losses. Table 1 summarizes the information about the number of epochs and the batch sizes that were used across different data sets. In the case of both datasets, the initial learning rate was $1e-3$ and we used the Newbob scheduler implementation provided by speechbrain (Ravanelli et al., 2021) (i.e. reducing the learning rate once the improvement on a validation set falls below a certain threshold). All such hyperparameters were tuned and chosen before applying our CL methods, so as to guarantee a fair setup. Interested readers are encouraged to consult our Github repository (Georgios, 2022) which includes the whole set of hyperparameters and the model architecture for the **CV8** experiments. The same model architecture and

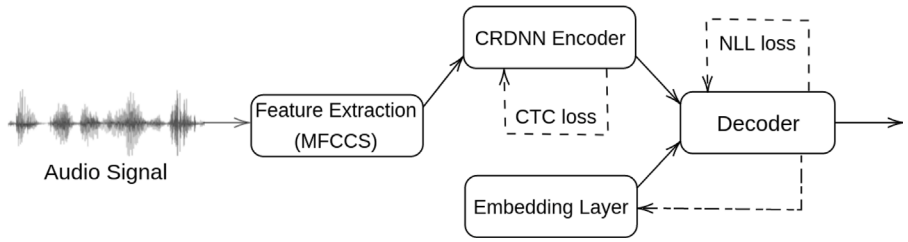


Fig. 1. Top-level architecture of the E2E ASR pipeline. The raw audio waveform is first processed to get the MFCC features, which are then re-represented by the encoder and embedding layer, and the output of these proceeds to the decoder.

Table 1
Top-level information that varies across the different training set configurations.

	LP	LP_{138}	LP_{414}	CV8
Train-set hours	1377	138	414	1375
Epochs[a]	15	75	60	30
CTC epochs	10	15	15	10
Batch size	32	32	32	64
Subwords	1750	1750	1750	500

[a] When using VPFs, the total number of epochs is 27 on **LP**, and 63 for **CV8**.

Table 2
Scoring function abbreviations and categorization.

SF Abbr.	CL family	Description
DUR	Metadata	Duration-based SF
CHR		Character-based frequency SF
TOK		Token-based frequency SF
WRD		Word-based frequency SF
WER	Adaptive	Adaptive SF using the WER
SEQ		Adaptive SF using the NLL loss

hyperparameters were used for the **LP** experiments, but due to license limitations at the time of this work, they were not made public.

The only top-level (non-CL-related) difference between the models trained on the two data sets, is the total number of output tokens. Here, we used Byte Pair Encoding (**BPE**) (Gage, 1994) to tokenize each word to what is known as *subword units*. This algorithm works by iteratively replacing the most frequent pair of characters with a single token (i.e., merging them) until the total number of desired tokens is reached. By applying this transformation to all textual data, the final tokens are thought to contain useful semantic information about the parts of each word (e.g., the word *abnormal* could be segmented as *ab*, *norm*, *al*, where the first token negates what follows). The number of chosen tokens is a hyperparameter that needs to be tuned carefully. Languages with low morphological variety (for example English) tend to require fewer tokens than languages that are morphologically richer, such as Finnish. Here, we will use 1750 tokens for the Lahjoita puhetta data set, since that was the number used in the preliminary experiments of Moisio et al. (2022). For Common Voice, we will follow the official Speechbrain recipe for the English dataset which uses 500 BPE tokens for their basic sequence-loss model.

4.3. Training strategies

In Section 3, we introduced and formalized the notion of curriculum learning along with certain strategies that can be formed by combining different scoring and pacing functions. Now, we will analyze the effect of a total of 23 CL strategies for the **LP** data set and 23 for the **CV8** data set. In the case of LP_{138} and LP_{414}, a total of 6 strategies will be examined.

4.3.1. Strategy abbreviations

To avoid long names, we will refer to each strategy with an abbreviation. These are created by adding a suffix or a prefix to the abbreviation of a scoring function. Table 2 summarizes the main abbreviations that correspond to SFs.

Each strategy abbreviation is created by combining the SF abbreviation with a prefix to denote the use of a pacing function or transfer learning CL and a suffix to denote whether uniform mixing was used or not. In addition, the down-arrow symbol (↓) will be used to mark all strategies that use the reverse ordering of the training data (i.e., starting from the hardest examples and moving towards the easier ones). The final abbreviations are created based on the following rules:

- **Pacing Functions**: The prefix **VPF-** for vanilla PFs and **SPF-** for subsampling PFs is added when PFs are used.
- **Transfer learning CL**: When using transfer learning CL, the prefix **TR-** is added. If a PF is also used then this abbreviation is appended to the PF prefix. One thing to point out regarding such strategies is that they should not be compared to the "*curriculum* strategy" of Hacohen and Weinshall (2019), since the latter utilizes a bigger model trained on more data. In our case, the teacher model has the exact same architecture as the student and in trained on the same set of data.
- **Uniform Mixing**: To mark models that used UM, we add the star symbol (*) as the suffix.

For example, a strategy that combines an adaptive UM-SF with WER as its criterion, and a subsampling pacing function, will be named **SPF-WER***. Similarly, a duration-based SF that uses descending sorting (hardest to easiest example), will be named **DUR** ↓.

At last, to examine how CL compares to simply randomly shuffling the training set, we have included the *random* strategy. In this case, the data set is randomly shuffled before the first epoch and the order is saved in a meta-data file, so that it will be kept even if the training script fails and re-starts. This strategy will be abbreviated as **RND**.

5. Results

5.1. Lahjoita Puhetta

As already mentioned, the Lahjoita Puhetta training set has been divided into three sets of different sizes to imitate the performance of our training strategies in mid and low-resource settings. We will first present the results on the full training set of 1377 h (**LP**) and then move towards the results on the LP_{138} and LP_{414} data sets. For the last two data sets, only the best-performing adaptive SFs and the transfer learning CL approach will be examined.

Additionally, each experiment will be executed **3** times with different random initializations of the weights of the corresponding models. To aggregate the results, we take an average of the scores yielded by the three models. This way, we ensure that the weight initialization does

[3] This issue was brought to the attention of the public after the creation of this GitHub issue https://github.com/kaldi-asr/kaldi/issues/2141.

Table 3
Results of various CL strategies on the **LP** data set. The STD is the average of the test set's WER scores across the three runs. **Bold** results are the best ones in each strategy group, while the best result of each column across the whole table is underlined.

Strategy	Validation		Test		STD
	CER	WER	CER	WER	
Baselines					
DUR	10.21	26.63	10.77	30.69	0.53
DUR↓	17.74	36.62	21.87	40.96	2.03
RND	**7.8**	**24.45**	**10.54**	**29.73**	0.45
Adaptive SFs					
WER	22.01	47.17	23.33	50.95	4.6
WER ↓	11.92	32.68	14.43	36.17	0.28
WER*	**7.47**	**23.27**	**10.14**	**28.98**	0.37
WER* ↓	9.44	27.98	11.13	30.91	0.24
SEQ*	9.03	27.25	11.5	32.74	0.35
Frequency SFs					
CHR	44.67	68.47	48.31	72.13	6.47
CHR*	9.2	27.31	11.07	30.62	0.95
TOK	27.14	52.66	30.63	55.97	22.09
TOK*	9.42	28.08	11.2	31.16	0.26
WRD	21.08	66.12	24.67	69.38	0.99
WRD*	**9.03**	**27.19**	**10.97**	**30.41**	0.37
Transfer CL					
TR-SEQ	9.22	27.61	**11.06**	**30.72**	0.38
TR-SEQ*	8.04	24.85	11.19	30.82	0.16
TR-WER*	**7.98**	**24.7**	11.1	30.88	0.74
Pacing functions					
VPF-TR-SEQ	9.46	28.63	11.23	31.64	0.23
VPF-TR-SEQ*	9.43	28.52	11.26	31.76	0.34
VPF-TR-WER	12.92	34.05	14.94	37.22	8.66
VPF-TR-WER*	13.44	33.76	15.64	37.1	10.23
SPF-WER*	**8.26**	**25.59**	**10.74**	**30.78**	0.28
SPF-TR-WER*	8.43	26.2	11.05	31.79	0.61

not taint the final results. The cohesion of the performances across the three runs is also a good indicator of the stability of the corresponding strategies (i.e., a high standard deviation of a strategy's scores would imply that it is unstable and that the average is not a good indicator of its capabilities).

Table 3 summarizes the effect of different CL strategies on the performance of models trained on the **LP** data set, while Table 4 contains the results for the LP subsets. The *STD* column corresponds to the standard deviation of the test set's WER scores across the three runs. For every CL strategy family, the top-scoring approach always corresponds to a UM-SF. In addition, for all **LP** sets, the best strategy (corresponding to underlined and bold scores) is the **WER***, which consistently outperforms both baseline strategies (DUR and the RND).

Frequency SFs and the transfer learning CL approach also seem to be improving the performance of the corresponding models when UM is used. On the contrary, without UM, all frequency-based strategies result in extremely unstable models (high st. deviation) that vastly overfit the training data, due to the concept of "excessive guidance" which was introduced in Section 3.2.2. In contrast, PFs do not offer any increase in the network's accuracy on the test set, even though SPFs still manage to produce WERs similar to those of the DUR strategy.

5.2. Common Voice

The effectiveness of our CL strategies on **CV8** is summarized in Table 5. These experiments were only run once and the reader should keep in mind that there is no way to distinguish unstable strategies. The error rates are a bit different compared to the Finnish ones and very few strategies managed to outperform the DUR baseline (whose performance was better than RND). In particular, only two had a higher accuracy on the test set and both of these were adaptive UM-SFs. The top scoring strategy is still the **WER*** one, which goes in accordance with the **LP** results, and the second best is SEQ*. Contradicting our previous observations, the transfer CL methods and the frequency-based SFs performed far worse in **CV8** than in **LP**.

Table 4
Results of various CL strategies on the LP_{138} and LP_{414} data sets. The STD is the average of the test set's WER scores across the three runs. **Bold** results are the best ones in each strategy group, while the best result of each column across the whole table is underlined.

Strategy	Validation		Test		STD
	CER	WER	CER	WER	
138 h set					
Baselines					
DUR	13.67	37.94	**16.14**	**43.56**	0.4
RND	**12.98**	**37.17**	16.15	43.69	0.28
WER	13.39	38.34	16.43	43.92	0.32
WER*	**12.74**	**36.74**	**15.86**	**43.17**	0.1
SEQ*	16.36	44.16	19.21	50.13	2.62
TR-WER*	**14.23**	**39.97**	**17.29**	**46.07**	3.56
414 h set					
Baselines					
DUR	13.03	32.41	13.61	35.88	1.26
RND	**9.23**	**28.2**	**12.48**	**34.74**	0.15
WER	11.66	32.34	16.36	39.44	0.89
WER*	**9.2**	**27.89**	**12.18**	**34.19**	0.38
SEQ*	10.39	30.97	13.17	37.03	0.08
TR-WER*	**9.62**	**29.19**	**12.86**	**35.65**	0.62

Table 5
Results of various CL strategies on the **CV8** data set. **Bold** results are the best ones in each strategy group, while the best result of each column across the whole table is underlined.

Strategy	Validation		Test	
	CER	WER	CER	WER
Baselines				
DUR	**11.62**	25.46	**15.63**	**31.23**
DUR↓	12.83	28.26	17.23	34.37
RND	11.82	**25.44**	16.28	31.7
Adaptive SFs				
WER	12.36	27.91	16.46	33.24
WER ↓	11.59	25.58	15.75	31.29
WER*	**11.09**	**24.34**	**15.27**	**30.19**
WER* ↓	12.33	25.96	18.11	33.48
SEQ	17.38	33.44	22.88	40.03
SEQ*	11.33	24.95	15.49	30.57
Frequency SFs				
CHR	15.01	30.99	19.93	37.28
CHR*	12.4	26.15	**17.85**	**33.35**
TOK	18.15	39.49	23.45	45.8
TOK*	**12.2**	**26.08**	18.13	33.79
WRD	20.89	47.08	27.35	54.19
WRD*	13.02	26.95	19.09	34.92
Transfer CL				
TR-SEQ*	**14.49**	**30.58**	**19.0**	**36.9**
TR-WER*	16.06	32.33	22.01	40.14
Pacing functions				
VPF-TR-SEQ	12.08	26.52	16.31	32.46
VPF-TR-SEQ*	12.23	26.62	16.6	32.63
VPF-TR-WER	**11.69**	**25.55**	**15.84**	**31.3**
VPF-TR-WER*	12.91	27.89	17.25	33.99
SPF-TR-WER*	13.51	27.58	19.93	35.44
SPF-WER*	12.73	26.96	17.89	33.92

6. Further analysis

6.1. Performance

The results presented in Section 5 support our initial hypothesis that a suitable curriculum learning strategy can decrease ASR errors in the E2E system. This finding was observed on both examined data sets (**LP** and **CV8**) and on the smaller subsets of **LP** that reflected a low-resource ASR setting. One of the novelties of this work in the CL realm was that of adaptive scoring functions combined with randomness injection techniques (uniform mixing and subsampling pacing functions). Our best CL

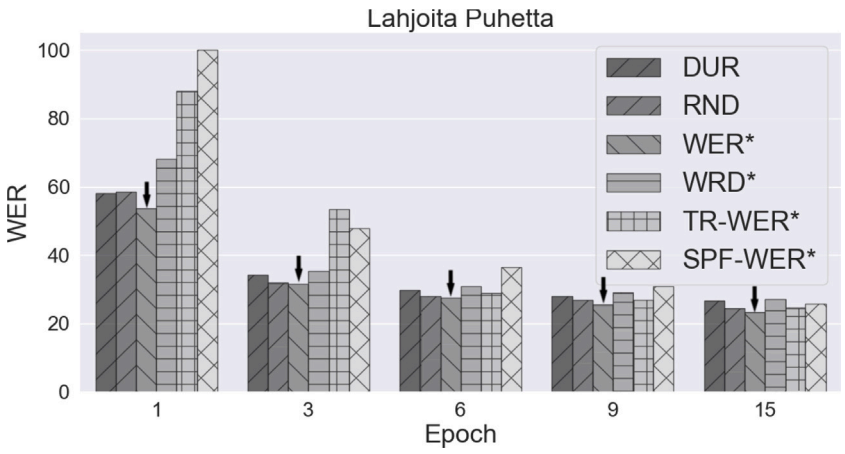


Fig. 2. Validation set WER scores for the top scoring strategies of each CL family on the **LP** data set.

strategy (WER*) consistently outperformed all of its competitors on all data sets while also providing stable results.

Another important observation is that UM had a 100% success rate in strengthening a model's generalization capabilities. As discussed in Section 3.2.2, providing *perfect guidance* can lead to severe overfitting (i.e., memorization) or even underfitting (e.g., when excessive guidance from a teacher leads to students that do not know how to approach solving a problem). The addition of noise in terms of mixing hard and easy examples was shown to help overcome this issue. A similar finding was observed in Zaremba and Sutskever (2015) where the authors employed a resembling randomness-injection technique. It is also important to note that, when using UM, the standard deviation of the test set WERs was significantly lower than the strategies' non-UM counterparts, with the exception of PF-based strategies. This shows the stabilizing effect that uniform mixing can bring, along with significant performance improvements. The only case were UM brings only small improvements, is with the LP_{138} and LP_{414} datasets that are smaller in size and for which the model is trained for more epochs. In addition, the complexity of colloquial Finnish, makes it hard for an E2E ASR system to learn from 138 h of data, and that is why all CL strategies result in weak models. It is hence natural that we get diminishing returns from the application of UM. Regarding PF-based strategies, and VPFs in particular, the inclusion of UM in a CL strategy did not prove beneficial. This is because the initial models are only trained on a small subset of the data, and allowing hard examples at the start of training

Figs. 2 and 3 illustrate the evolution of the validation WER scores for some of the best-scoring CL strategies (excepting VPF strategies). The downward arrows mark the strategies with the lowest error rate at each stage. The differences between those values and the WERs of the DUR, RND baselines are also statistically significant (see Section 6.2). The WER* strategy is consistently the one with the lowest error rate. It is also worth noting that the transfer CL strategies start from high validation errors, and then drastically improve around the middle of their training. This behavior is observable for all transfer CL strategies (mainly on **LP**) and the reason behind this is that the transferred difficulty scores from the teacher network do not match the capabilities of the initially bad networks. In such cases, we hypothesize that it would be helpful to initialize the models with the DUR strategy for the first few epochs and then move to the more complex transfer-CL strategies, but testing this idea is out of the scope of this paper.

In addition, the success of the **RND** strategy in the **LP** results also supports the argument that randomness can be beneficial when training neural networks. In the case of **RND** though, this typically comes at the expense of resources and higher training times, since the examples in each batch are likely to be of different lengths, hence increasing the amount of padding that needs to be done. In most cases, this is also an issue with our CL strategies. Although we empirically noticed that adaptive CL strategies start with slow training times (per epoch), as the utterances' scores change (adapt), the batches become more consistent in terms of length, and, hence, each epoch is completed faster (see 6.2 for more information). The non-adaptive CL approaches typically resulted in much more uniform batches, and the training times were closer to the DUR baseline (which minimizes padding). A more detailed training time analysis follows in 6.4.

In the case of the **LP** data set (and its subsets), two out of the three strategies that outperformed DUR were created using frequency-based SFs. In particular, by combining uniform mixing with the word and character-level frequency SFs, we get strong and stable models that showed little variation in their test set performances. Unfortunately, this trend is not followed in the case of **CV8**, where all frequency-based SFs failed to get close to the either baseline's performance. A possible reason for this difference is that the grammatical and syntactical structure of Finnish is vastly different from that of English. In addition, in contrast to English, Finnish words are (almost) always pronounced the same way as it is written. Due to that, the characters, words, and tokens that are present in a sentence have a 1-1 mapping to the phonemes pronounced in the corresponding utterance, hence making the textual frequencies more informative about the underlying difficulty.

6.1.1. *Pacing functions*

In Hacohen and Weinshall (2019), the authors' implementation of pacing functions was shown to bring significant improvements over their baselines. In this work, we chose a slightly different approach by combining them with different SFs and UM, and treating them as a tool to improve the convergence speed of end-to-end models. All PF-based strategies fail to surpass the DUR strategy while the combination of WER-based SFs and VPFs, resulted in unstable models (high standard deviation on the **LP** data). In addition, contrary to the rest of UM-based strategies, combining VPFs with UM does not result in stronger models (e.g. the addition of UM in the VPF-TR-WER strategy, leads to an $\sim$ 9% relative increase in WER). There is no clear indication of what causes this, although it could be argued that this is due to the sudden introduction of difficult examples from the early stages of training when using just a subset of the data. On the other hand, given the fact that PFs (and especially SPFs) parse the training set fewer times than normal training, we could argue that using them is beneficial. For instance, in the case of **LP**, the SPF-TR-WER* strategy achieved a test WER very close to the DUR baseline while being much faster to train (see Sections 6.1.3 and 6.4). This means that the use of PFs has the ability to increase convergence speed and reduce computation costs.

The relative success of SPFs on both data sets can also validate the general effectiveness of UM, since the SPF strategies typically introduce hard examples at much earlier stages and there is no abrupt addition of information. It is worth pointing out again that SPFs are trained

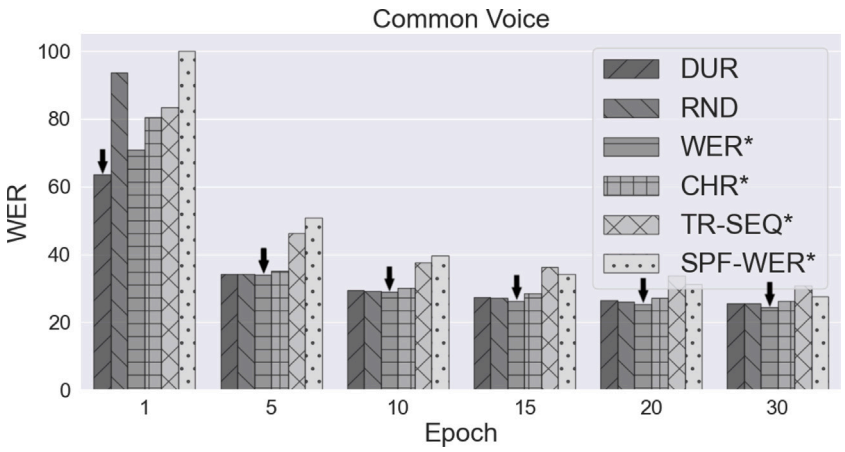


Fig. 3. Validation set WER scores for the top scoring strategies (no VPF strategies included due to the different number of total epochs) of each CL family on the **CV8** data set.

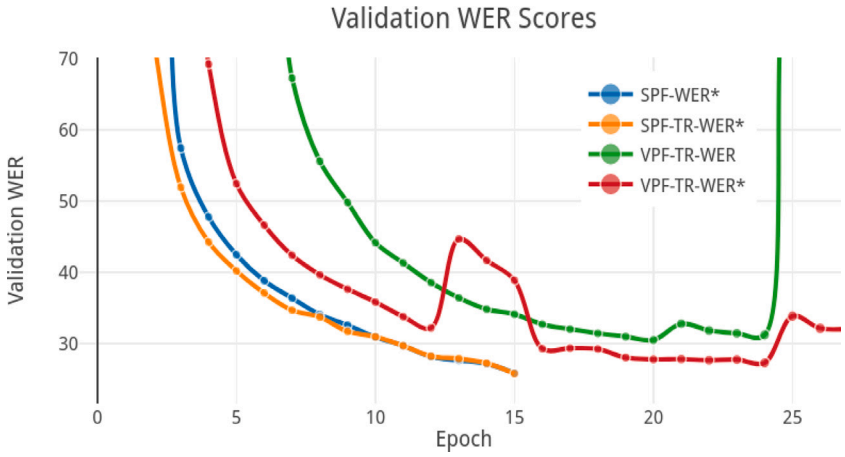


Fig. 4. Validation set WER scores for some PF strategies in **LP**. All VPF strategies were trained for 27 epochs, while SPF ones for 15 epochs.

for much fewer hours than VPFs (see 6.1.3 for a comparison) without any significant decrease in quality. Fig. 4 shows the stability of SPFs since the validation error keeps on improving at the end of training without any significant oscillations. This implies that the corresponding models have not yet converged and could potentially perform even better. Unfortunately, due to time limitations, this assumption was not tested.

6.1.2. Reverse curriculum

As mentioned in Section 3, the reverse curriculum is not an effective CL method, at least for E2E ASR. In both of our data sets, the reverse duration-based CL approach resulted in considerably worse models compared to both the DUR and RND baseline, while also being more unstable (high standard deviation in **LP**). Fig. 5 portrays the effect of reverse CL strategies on the training and validation losses on the **LP** data set. On the left, it is clear that the DUR↓ model shows signs of overfitting for the bigger part of training since the validation loss is noticeably higher than the training loss.

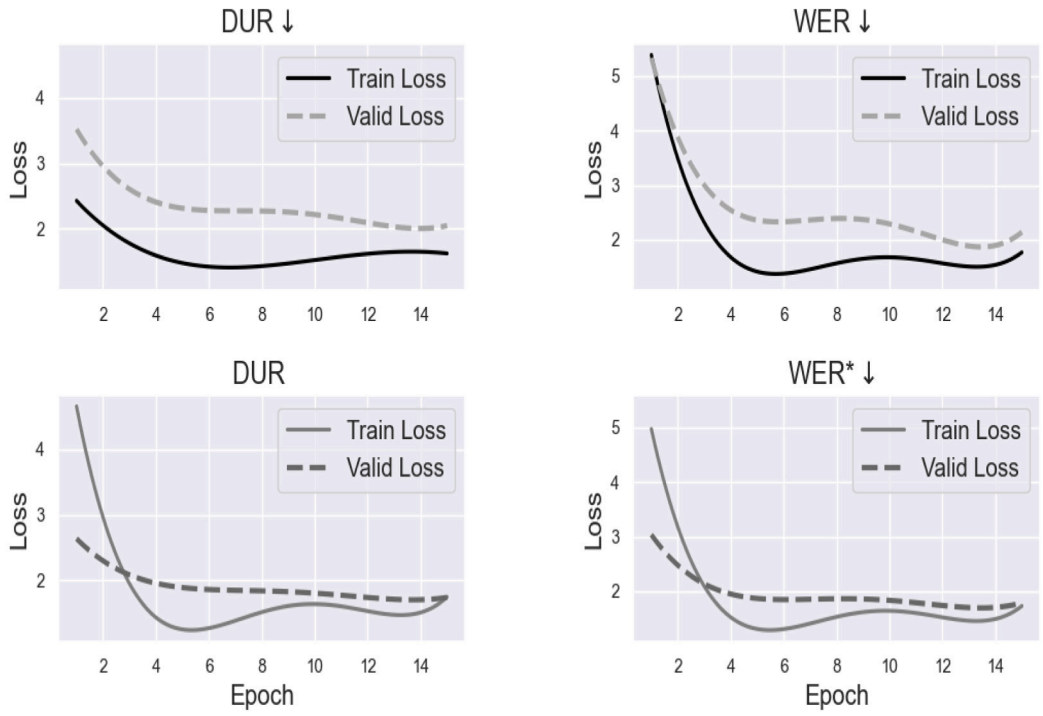
On the right-hand side, it seems that reverse CL does not have such a negative impact on the adaptive CL strategies. This is also clear from the results in Table 3, where reverse CL contributes as a regularizer to the otherwise weak WER strategy, leading to an improvement of almost 29%. Of course, even this kind of improvement is not enough to make this model competitive. In contrast, the UM-based WER strategy (WER*) has managed to remain relatively stable, even under the application of a reverse curriculum. Its test set performance is close to the DUR baseline, being only 0.7% worse in terms of word error rate.

Similar findings are also observed in the case of the Common Voice data set, except for the WER*↓ strategy, which suffered a setback of 7% compared to the DUR baseline. On the contrary, reverse curriculum was again beneficial for the WER strategy, leading to a moderate improvement of 6%.

Overall, our empirical results point us towards the conclusion that reverse curriculum as a non-beneficial method, especially when combined with a duration-based scoring function. Our experiments show that initializing a model with hard examples leads to larger updates of a model's parameters at the start of training, and, consequently, slower convergence. On the other hand, normal curriculum learning – with ascending sorting – tends to overfit on the easy data that it learns during the early stages of training and fails to generalize. The solution is to balance the two approaches and utilize easy examples for faster learning, while also introducing harder examples as a form of regularization.

6.1.3. Hours seen

Neural networks are typically trained for many epochs to better approximate the optimal parameters. To fairly compare two models trained on the same data set, one would expect them both to be trained for the same number of epochs (assuming that the neural network architecture stays the same). In the case of models trained with PF strategies, though, the size of the training set increases at frequent time intervals, which means that, if the corresponding models are trained for the same epochs as the either the DUR or the RND baseline, then they would still have encountered fewer data. For this reason, we are going



(a) Metadata CL (b) Adaptive CL

Fig. 5. Comparison of training and validation loss values across the epochs with reverse curriculum strategies on the **LP** dataset. Note the severe overfitting in the case of DUR↓ and WER↓.

to use the number of hours that each model has seen instead of the number of epochs they have been trained on.

For all non-PF strategies, calculating this number can be done by multiplying the number of epochs, with the total amount of hours in the training set (e.g., $30 \cdot 1375$ for **CV8**). With VPF strategies, it is possible to approximate the total number of times the full training set has been parsed, by using the following formula:

$$k\frac{1}{g} + k\frac{2}{g} + \cdots + k\frac{g}{g} = k\sum_{i=1}^{g}\frac{i}{g} = \frac{k}{g}\frac{g(g-1)}{2} = \frac{k(g-1)}{2}$$

where k is the number of epochs after which we increase the training set size, and g is the number of subset groups defined as $g = \frac{E}{k}$, with E being the total number of epochs. The above sequence can be read as *"use $\frac{1}{g}$ of the training set for the first k epochs, then $\frac{2}{g}$ for the next k epochs, and repeat until the maximum number of epochs is reached"*. In the case of **CV8**, k was equal to 3 and $g = \frac{63}{3} = 21$, which means that for a total of 1375 h and after 63 epochs, the model had seen $1375 \cdot \frac{3 \cdot 20}{2} = 41250$ h of training data, which is exactly the same as the amount of hours seen by a model trained for 30 epochs (i.e., $30 \cdot 1375 = 41250$).

Calculating the total number of hours seen by the models trained with the SPF strategy is a bit more complicated since each subsequent subset does not necessarily contain all utterances present in the previous subset. This means that the total amount of hours seen has to be calculated by summing up the individual hours of each subset. An implementation of that is provided in the Github repository (Georgios, 2022).

To sum up, Table 6 contains the total amount of hours seen by the DUR baseline and the PF-based strategies. As already mentioned, VPFs are designed so that they *see* almost as many hours as the baselines (in the case of **LP**, VPFs processed fewer hours of data because there was no performance improvement after increasing the hours seen by the models and we chose to save on resources by reducing the total number of epochs). On the other hand, SPFs typically process much less data, but even though that is the case, their performance is on par with the rest of the models. In particular, on the **LP** data set, the SPF-WER* strategy had seen around 44% fewer hours of data than the baselines, while its test set performance is only 3% worse than that of the DUR strategy. To bring this difference into context, the models trained on the LP_{138} data set have seen $10.35K$ h of data, but the best model's performance is $\sim$ 40% worse than that of SPF-WER* which has seen a similar amount of hours. This is mostly because SPF-WER* models have seen many different kinds of training examples, which allows them to generalize better. This finding is also an indicator of the importance of CL and pacing functions, in particular, in training deep neural networks.

Table 6
Number of hours that PF strategies have seen throughout their training cycle (shown under the *Hours* column). The validation and test set WER scores are also shown for comparison.

Strategy	Lahjoita Puhetta			Common Voice		
	Hours	Valid	Test	Hours	Valid	Test
DUR	20.73K	26.63	**30.69**	41.25K	**25.46**	**31.23**
VPF-TR-SEQ	16.54K	28.63	31.64	41.25K	26.52	32.46
VPF-TR-WER	16.54K	34.05	37.22	41.25K	25.55	31.30
SPF-WER*	**11.54K**	**25.59**	30.78	**23.06K**	26.96	33.92
SPF-TR-WER*	11.82K	26.2	31.79	28.81K	27.58	35.44

6.2. Variability of the curricula

Each CL strategy leads to the creation of different orderings of the corresponding training set. To validate the results presented in 5, we have performed two kinds of statistical significance tests, using the Matched Pairs Sentence-Segment Word Error test and a p-value threshold of 0.001.

1. Variability of the orderings: This refers to testing the similarities of the orderings produced by a certain strategy, across the different weight initializations (only applicable to **LP**).
2. Variability of the results: This refers to testing how significant the difference is in our CL strategies' test set scores compared to the DUR baseline. The reason for choosing the DUR strategy instead of RND is because DUR outperformed the latter in the **CV8** dataset which is the one used in our tests.

Concerning the former, no significant difference was observed between the three TR-WER* models, while the difference in their orderings compared to DUR's was significant at a lower confidence level ($0.001 < p < 0.05$). This supports the argument that our transfer CL scoring functions result in orderings that also minimize padding to a certain degree. Our best CL strategy WER* arranged the training set in a significantly different way than all other approaches. Even though that was the case, for one weight initialization, the difference of the orderings compared to the ones of the RND strategy was shown to be significant at a lower confidence level ($0.001 < p < 0.05$). The performance of this model, though, was the worst one out of the three runs. This could mean that a bad weight initialization can negatively affect the functioning of the WER* strategy, and bring it closer to random shuffling.

Regarding the second type of statistical significance test that focused on the difference of the test WER of our models with respect to DUR's performance, we ran the same kind of test on the results of the **CV8** data set. All of our strategies had significantly different performances compared to DUR ($p < 0.001$), which is a good indicator that our top-scoring strategies are indeed better. Although, when comparing the performance among the strategies we proposed, the results of the best-scoring WER* strategy were shown to be different at a lower confidence from that of VPF-TR-WER ($0.001 < p < 0.05$).

By further examining Table 5, it can be noticed that the validation and test CER scores of the VPF-TR-WER and WER* strategies are relatively close, while the difference between the WER scores of the two strategies is bigger. This is partly explained by the large number of character substitutions needed in the case of the VPF-TR-WER strategy, due to a large number of orthographic errors. For example, if the ground truth is *"where do you live"*, and the prediction is *"where do you leave"*, the word error rate would be $1/4 = 25\%$ while the character error rate would be $2/17 = 11.76\%$. Even though the prediction shows a lack of grammatical understanding of English, it is still acoustically correct since *"leave"* and *"live"* are pronounced very similarly. Such errors were very prominent in the case of VPF-TR-WER and are the main reason behind its absolute difference in terms of test word error rate when compared to the WER* strategy. Besides that, the two models have similar predictions, hence, the lack of the aforementioned significant difference in terms of performance is partly explained.

6.3. Error analysis

Empirical analysis showed that non-UM-based strategies tended to have a higher number of substitution and deletion edits required to transform the predicted transcriptions to the ground truth. Fig. 6 represents the distribution of insertions, deletions, and substitutions for the WRD and WER strategies, with and without UM, on the LP test set. Along with the results of Table 7, it is clear that these two types of edits separate the well-performing models from the bad ones. As already mentioned, a high number of substitutions (when calculating the word error rate) implies a misunderstanding of orthography and syntax. On the other hand, many deletions mean that the model failed to transcribe some part of an utterance.

The top-scoring WER* strategy has many outliers with a high number of insertion edits. This is due to extensive word repetitions that occurred in cases of noisy/unintelligible audio. For instance, a noisy utterance was transcribed as the word *"hm"* repeated 126 times, which, in turn, led to 119 insertion edits. Such errors are a well-known issue with E2E models and Speechbrain. A straightforward way to approach them would be to ignore all words repeated more than twice in a row. But, this solution is not able to handle repeated sequences of words, which are also apparent in many of the predicted transcripts. The aforementioned outliers are the main reason why the *Total Edits* violin plot also shows this kind of sign. Even though this problem was more apparent in **LP**, it was also noticed in **CV8**, which is generally considered cleaner.

Table 7
Total number of word-level edits (Insertions, Deletions, Substitutions) needed to be done to "fix" a predicted transcription among UM and non-UM-based strategies. Higher values are marked with bold.

Strategy	Ins	Dels	Subs	Total
LP				
WER	**1887**	4127	12 199	18 213
WER*	1585	1053	4637	7275
WRD	147	**22 252**	**15 620**	**38 019**
WRD*	1841	1004	5196	8041
CV8				
WER	1700	**6001**	18 545	26 246
WER*	2838	3345	16 546	22 729
SEQ	**8509**	5752	**23 546**	**37 807**
SEQ*	3282	3259	16 864	23 405

These effects are also visible in Table 7, which shows that non-UM-based strategies result in a higher number of deletions and substitutions when compared to their counterparts on both **LP** and **CV8**. The presence of outliers is not as visible here as it is in Fig. 6 and that is why both forms of representations are needed to find the relationship between edits and final performance. Even though the best UM-based strategies have some outlier utterances with the highest number of edits, their summed number of edits is much less than that of non-UM-based strategies, which explains the significant difference in their performance and the superiority of UM-based strategies.

6.4. Training time analysis

Another factor to take into consideration is the total training time overhead that each CL strategy had when compared to DUR. The reason we compare with DUR is because this is the strategy that is the fastest for a given number of steps, since it minimizes padding. In our case, this is possible to compute since all **LP** and **CV8** models were trained on the same GPU architecture, using the same resources. Table 8 contains the observed training time overhead of our tested CL strategies for both **LP** and **CV8** data sets (note that this also includes the time required to compute and update the curriculum scores). The overhead is calculated as the normalized difference between the training time required by a CL strategy and the time required by DUR. For **LP**, the times are first averaged across the three initializations and then the overhead is computed.[4] Note that all results below should be taken with a grain of salt, since measuring wall-clock times is not always completely precise. Other (system-related) background jobs may have affected the processing of each batch and the transitions from CPU to GPU.

It is clear that metric-based adaptive SFs are much slower than the other alternatives. This is because, in such strategies, a decoding step needs to be performed for the whole training set, which is typically slow. Another interesting observation is that UM-based strategies are slower than non-UM-based ones (except for WRD on **LP**). In general, there is a clear difference between the training times on the two data sets. For example, TR-WER* had only an 8.3% overhead on **LP**, while it had 44.9% overhead on **CV8**. Another noteworthy comparison is

[4] The LP_{138} and LP_{414} recipes used an older codebase that did not consider training times, and so they were exempt from this analysis.

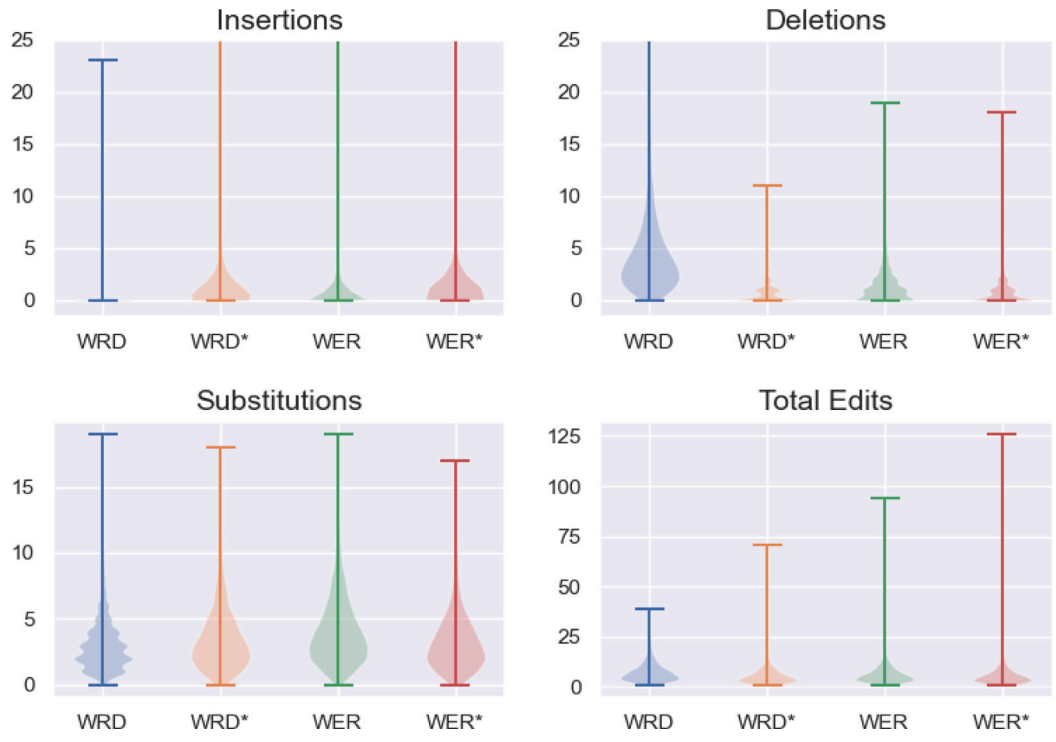


Fig. 6. Distribution of insertions/deletions/substitutions for two pairs of UM and non UM-based strategies on **LP**.

Table 8
Training time overhead of our CL strategies, compared to the DUR strategy (also includes the time required to compute the curriculum scores). **Bold** numbers denote the faster models of each CL family, while underlined ones denote the slowest strategies.

Strategy	Overhead (%)	
	Lahjoita Puhetta	Common Voice
RND	12.7%	27%
Adaptive SFs		
SEQ	–	**14%**
SEQ*	**−0.3%**	17%
WER	2%	17.7%
WER ↓	11.1%	15.6%
WER*	15%	43%
WER* ↓	15.5%	39%
Frequency SFs		
CHR	8.7%	18.8%
CHR*	12.1%	22.2%
TOK	13%	21.7%
TOK*	14.8%	21.9%
WRD	11.7%	**18.4%**
WRD*	**1%**	21.2%
Transfer CL		
TR-SEQ	4.6%	–
TR-SEQ*	**−2%**	**0.5%**
TR-WER*	8.3%	44.9%
Pacing functions		
VPF-TR-SEQ	15.2%	14.7%
VPF-TR-SEQ*	15.1%	15.4%
VPF-TR-WER	23.3%	40.4%
VPF-TR-WER*	27.5%	15%
SPF-TR-WER*	**−32.1%**	**−8.1%**
SPF-WER*	−8.1%	12.2

between RND and the WER strategy on **CV8**, where RND was much slower, even though WER requires the extra decoding step mentioned earlier. It is not clear what caused these discrepancies, but the fact that **CV8** experiments were run only once could mean that this is the result of randomness.

Regarding the pacing functions, normally, they would be expected to be much faster than DUR since they perform fewer training steps, but that is not always the case. For the VPFs, this is easily explained by the fact that the corresponding models were trained so that they see the same amount of hours of data as an unpaced model would. This means that, overall, models trained with VPFs, had to do more iterations (which can be costly since, at the end of each epoch, the validation error is also computed) while having the same overhead as inflicted by our CL strategies (mainly adaptive SFs). On the other hand, SPFs were trained for the exact number of epochs as the rest of the models, while having seen much fewer data. This means that they are much faster in terms of total training time. Unfortunately, this is not clearly seen in the case of **CV8** and SPF-WER*, where the decoding step was probably too costly. Although, it is clear that when combining SPFs with transfer CL, training is always faster, without a huge decrease in terms of quality.

7. Conclusion

In this work, we formally defined curriculum learning as a combination of scoring and pacing functions and evaluated multiple novel, ASR-specific CL strategies. Our experiments demonstrate the effectiveness of such strategies, both in terms of improving a network's accuracy and enabling faster convergence, and, at the same time, leading to lower training times. In particular, the WER* strategy, which combines uniform mixing and a word error rate-based ordering of the training data, led to consistent improvements in both Finnish and English ASR models. However, a drawback of this technique is the extra decoding step that needs to be performed at the end of each epoch, which makes training slower. To counter this, we showed that the use of pacing functions can lead to faster convergence without suffering big drawbacks in performance when compared to either the commonly used DUR baseline or the RND baseline. In addition to the audio-based strategies, the paper also explored text-based ones, which take token frequencies as an indication of an utterance's difficulty. The combination of this approach with uniform-mixing led to the TOK*, CHR*, and WRD* strategies that provided good results on the Finnish

data set. However, these strategies were not proved as effective in English, since words are usually not pronounced the same way they are written, hence making the token frequency metrics irrelevant. Overall, it is clear that curriculum learning can be beneficial for ASR, and we highlight some trade-offs between performance and training time that can be managed by picking the right CL strategy.

CRediT authorship contribution statement

Georgios Karakasidis: Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Mikko Kurimo:** Conceptualization, Funding acquisition, Investigation, Resources, Supervision, Writing – review & editing. **Peter Bell:** Formal analysis, Supervision, Writing – review & editing. **Tamás Grósz:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Both datasets used in our experiments are publicly available. The lahjoita puhetta subsets (138 and 414 h) are currently not online, but could be made available upon request.

Acknowledgments

The computational resources were provided by Aalto Science IT, Finland. We are grateful for the Academy of Finland project funding number 345790 in ICT 2023 programme's project "Understanding speech and scene with ears and eyes".

References

Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F.M., Weber, G., 2019. Common voice: A massively-multilingual speech corpus. arXiv preprint arXiv:1912.06670.

Bengio, Y., Louradour, J., Collobert, R., Weston, J., 2009. Curriculum learning. In: Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09. ACM Press, Montreal, Quebec, Canada, pp. 1–8. http://dx.doi.org/10.1145/1553374.1553380, URL http://portal.acm.org/citation.cfm?doid=1553374.1553380.

Braun, S., Neil, D., Liu, S.-C., 2017. A curriculum learning method for improved noise robustness in automatic speech recognition. In: 2017 25th European Signal Processing Conference (EUSIPCO). pp. 548–552. http://dx.doi.org/10.23919/EUSIPCO.2017.8081267, ISSN: 2076-1465.

Chan, W., Jaitly, N., Le, Q., Vinyals, O., 2016. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing. ICASSP, IEEE, pp. 4960–4964.

Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., Cui, X., Witbrock, M., Hasegawa-Johnson, M.A., Huang, T.S., 2017. Dilated recurrent neural networks. Adv. Neural Inf. Process. Syst. 30.

Elman, J.L., 1993. Learning and development in neural networks: The importance of starting small. Cognition 48 (1), 71–99, Publisher: Elsevier.

Gage, P., 1994. A new algorithm for data compression. C Users J. 12 (2), 23–38.

Gales, M., Young, S., 2008. The Application of Hidden Markov Models in Speech Recognition. Now Publishers Inc.

Georgios, K., 2022. Speechbrain-CL. URL https://github.com/aalto-speech/speechbrain-cl.

Graves, A., 2012. Sequence transduction with recurrent neural networks. arXiv preprint arXiv:1211.3711.

Graves, A., Bellemare, M.G., Menick, J., Munos, R., Kavukcuoglu, K., 2017. Automated curriculum learning for neural networks. In: International Conference on Machine Learning. PMLR, pp. 1311–1320.

Graves, A., Fernández, S., Gomez, F., Schmidhuber, J., 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: Proceedings of the 23rd International Conference on Machine Learning. pp. 369–376.

Graves, A., Mohamed, A.-r., Hinton, G., 2013. Speech recognition with deep recurrent neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. Ieee, pp. 6645–6649.

Guo, S., Huang, W., Zhang, H., Zhuang, C., Dong, D., Scott, M.R., Huang, D., 2018. Curriculumnet: Weakly supervised learning from large-scale web images. In: Proceedings of the European Conference on Computer Vision. ECCV, pp. 135–150.

Guo, A., Kamar, E., Vaughan, J.W., Wallach, H., Morris, M.R., 2020. Toward fairness in AI for people with disabilities SBG@ a research roadmap. ACM SIGACCESS Access. Comput. (125), 1.

Hacohen, G., Weinshall, D., 2019. On the power of curriculum learning in training deep networks. In: International Conference on Machine Learning. PMLR, pp. 2535–2544.

Higuchi, T., Saxena, S., Souden, M., Tran, T.D., Delfarah, M., Dhir, C., 2021. Dynamic curriculum learning via data parameters for noise robust keyword spotting. In: ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing. ICASSP, IEEE, pp. 6848–6852.

Jiang, L., Meng, D., Mitamura, T., Hauptmann, A.G., 2014. Easy samples first: Self-paced reranking for zero-example multimedia search. In: Proceedings of the 22nd ACM International Conference on Multimedia. pp. 547–556.

Jiang, L., Meng, D., Zhao, Q., Shan, S., Hauptmann, A.G., 2015. Self-paced curriculum learning. In: Twenty-Ninth AAAI Conference on Artificial Intelligence.

Kala, T., Shinozaki, T., 2018. Reinforcement learning of speech recognition system based on policy gradient and hypothesis selection. In: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp. 5759–5763.

Karakasidis, G., Grósz, T., Kurimo, M., 2022. Comparison and analysis of new curriculum criteria for end-to-end ASR. In: Proc. Interspeech 2022. pp. 66–70. http://dx.doi.org/10.21437/Interspeech.2022-10046.

Kim, S., Seltzer, M., Li, J., Zhao, R., 2018. Improved training for online end-to-end speech recognition systems. In: Proc. Interspeech 2018. pp. 2913–2917. http://dx.doi.org/10.21437/Interspeech.2018-2517.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kumar, M., Packer, B., Koller, D., 2010. Self-paced learning for latent variable models. Adv. Neural Inf. Process. Syst. 23.

Kuznetsova, A., Kumar, A., Fox, J.D., Tyers, F.M., 2022. Curriculum optimization for low-resource speech recognition. In: ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing. ICASSP, IEEE, pp. 8187–8191.

Kuznetsova, A., Kumar, A., Tyers, F.M., 2021. A bandit approach to curriculum generation for automatic speech recognition. arXiv:2102.03662 [cs, eess] URL http://arxiv.org/abs/2102.03662.

Moisio, A., Porjazovski, D., Rouhe, A., Getman, Y., Virkkunen, A., AlGhezi, R., Lennes, M., Grósz, T., Lindén, K., Kurimo, M., 2022. Lahjoita puhetta: a large-scale corpus of spoken finnish with some benchmarks. Lang. Resour. Eval. 33. http://dx.doi.org/10.1007/s10579-022-09606-3.

Newport, E.L., 1990. Maturational constraints on language learning. Cogn. Sci. 14 (1), 11–28.

Ng, D., Chen, Y., Tian, B., Fu, Q., Chng, E.S., 2022. ConvMixer: Feature interactive convolution with curriculum learning for small footprint and noisy far-field keyword spotting. In: ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing. ICASSP, IEEE, pp. 3603–3607.

Novoa, J., Fredes, J., Poblete, V., Yoma, N.B., 2018. Uncertainty weighting and propagation in DNN–HMM-based speech recognition. Comput. Speech Lang. 47, 30–46, Publisher: Elsevier.

Parisi, G.I., Kemker, R., Part, J.L., Kanan, C., Wermter, S., 2019. Continual lifelong learning with neural networks: A review. 113, pp. 54–71,

Pavlov, P.I., 2010. Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex. Ann. Neurosci. 17 (3), 136, Publisher: SAGE Publications.

Penha, G., Hauff, C., 2019. Curriculum learning strategies for IR: An empirical study on conversation response ranking. arXiv preprint arXiv:1912.08555.

Platanios, E.A., Stretcu, O., Neubig, G., Póczos, B., Mitchell, T., 2019. Competence-based curriculum learning for neural machine translation. 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). pp. 1162–1172.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., et al., 2011. The kaldi speech recognition toolkit. In: IEEE 2011 Workshop on Automatic Speech Recognition and Understanding. Vol. CONF, IEEE Signal Processing Society.

Ranjan, S., Hansen, J.H.L., 2018. Curriculum learning based approaches for noise robust speaker recognition. IEEE/ACM Trans. Audio Speech Lang. Process. 26 (1), 197–210. http://dx.doi.org/10.1109/TASLP.2017.2765832, Conference Name: IEEE/ACM Transactions on Audio, Speech, and Language Processing.

Ravanelli, M., Parcollet, T., Plantinga, P., Rouhe, A., Cornell, S., Lugosch, L., Subakan, C., Dawalatabad, N., Heba, A., Zhong, J., et al., 2021. SpeechBrain: A general-purpose speech toolkit. arXiv preprint arXiv:2106.04624.

Rouhe, A., Van Camp, A., Singh, M., Van Hamme, H., Kurimo, M., 2021. An equal data setting for attention-based encoder-decoder and HMM/DNN models: A case study in finnish ASR. In: Speech and Computer: 23rd International Conference, SPECOM 2021, St. Petersburg, Russia, September 27–30, 2021, Proceedings 23. Springer, pp. 602–613.

Rousseau, A., Deléglise, P., Esteve, Y., 2012. TED-LIUM: an automatic speech recognition dedicated corpus. In: LREC. pp. 125–129.

Shi, Y., Larson, M., Jonker, C.M., 2015. Recurrent neural network language model adaptation with curriculum learning. Comput. Speech Lang. 33 (1), 136–154. http://dx.doi.org/10.1016/j.csl.2014.11.004, URL https://linkinghub.elsevier.com/retrieve/pii/S0885230814001211, Number: 1.

Tay, Y., Wang, S., Luu, A.T., Fu, J., Phan, M.C., Yuan, X., Rao, J., Hui, S.C., Zhang, A., 2019. Simple and effective curriculum pointer-generator networks for reading comprehension over long narratives. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. pp. 4922–4931.

Terbeh, N., Labidi, M., Zrigui, M., 2013. Automatic speech correction: A step to speech recognition for people with disabilities. In: Fourth International Conference on Information and Communication Technology and Accessibility. ICTA, IEEE, pp. 1–6.

Turkewitz, G., Kenny, P.A., 1982. Limitations on input as a basis for neural organization and perceptual development: A preliminary theoretical statement. Dev. Psychobiol.: J. Int. Soc. Dev. Psychobiol. 15 (4), 357–368.

Wang, X., Chen, Y., Zhu, W., 2021. A survey on curriculum learning. IEEE Trans. Pattern Anal. Mach. Intell..

Wang, D., Wang, X., Lv, S., 2019. An overview of end-to-end automatic speech recognition. Symmetry 11 (8), 1018, Publisher: Multidisciplinary Digital Publishing Institute.

Watanabe, S., Hori, T., Karita, S., Hayashi, T., Nishitoba, J., Unno, Y., Soplin, N.E.Y., Heymann, J., Wiesner, M., Chen, N., Renduchintala, A., Ochiai, T., 2018. ESPnet: End-to-end speech processing toolkit. http://dx.doi.org/10.48550/ARXIV.1804.00015, URL https://arxiv.org/abs/1804.00015.

Yin, S., Liu, C., Zhang, Z., Lin, Y., Wang, D., Tejedor, J., Zheng, T.F., Li, Y., 2015. Noisy training for deep neural networks in speech recognition. EURASIP J. Audio Speech Music Process. 2015 (1), 1–14, Publisher: Springer.

Zaremba, W., Sutskever, I., 2015. Learning to execute. arXiv:1410.4615 [cs] URL http://arxiv.org/abs/1410.4615.

Zhang, D., Han, J., Zhang, Y., Xu, D., 2019. Synthesizing supervision for learning deep saliency network without human annotation. IEEE Trans. Pattern Anal. Mach. Intell. 42 (7), 1755–1769.

Zhou, T., Bilmes, J., 2018. Minimax curriculum learning: Machine teaching with desirable difficulties and scheduled diversity. In: International Conference on Learning Representations.

Zhou, T., Wang, S., Bilmes, J., 2020. Curriculum learning by dynamic instance hardness. Adv. Neural Inf. Process. Syst. 33, 8602–8613.

Zhu, Y., Nie, J.-Y., Su, Y., Chen, H., Zhang, X., Dou, Z., 2022. From easy to hard: A dual curriculum learning framework for context-aware document ranking. In: Proceedings of the 31st ACM International Conference on Information & Knowledge Management. pp. 2784–2794.