Mitigating the Burden of Redundant Datasets via Batch-Wise Unique Samples and Frequency-Aware Losses

Donato Crisostomi[*]
Amazon Alexa AI
Sapienza, University of Rome
crisostomi@di.uniroma1.it

Andrea Caciolai[*]
Amazon Alexa AI
andccl@amazon.it

Alessandro Pedrani
Amazon Alexa AI
pedrana@amazon.it

Alessandro Manzotti
Amazon Alexa AI
manzotti@amazon.it

Enrico Palumbo
Amazon Alexa AI
palumboe@amazon.it

Kay Rottmann
Amazon Alexa AI
krrottm@amazon.de

Davide Bernardi
Amazon Alexa AI
dvdbe@amazon.it

Abstract

Datasets used to train deep learning models in industrial settings often exhibit skewed distributions with some samples repeated a large number of times. This paper presents a simple yet effective solution to reduce the increased burden of repeated computation on redundant datasets. Our approach eliminates duplicates at the batch level, without altering the data distribution observed by the model, making it model-agnostic and easy to implement as a plug-and-play module. We also provide a mathematical expression to estimate the reduction in training time that our approach provides. Through empirical evidence, we show that our approach significantly reduces training times on various models across datasets with varying redundancy factors, without impacting their performance on the Named Entity Recognition task, both on publicly available datasets and in real industrial settings. In the latter, the approach speeds training by up to 87%, and by 46% on average, with a drop in model performance of 0.2% relative at worst. We finally release a modular and reusable codebase to further advance research in this area.

1 Introduction

Deep neural networks have recently enabled impressive results across many fundamental tasks (Brown et al., 2020; Dosovitskiy et al., 2021). However, training state-of-the-art models is now a very demanding process, in terms of both time and resources (Strubell et al., 2019). The issue is exacerbated when models are required to train on datasets that naturally exhibit a redundant distribution. User queries are one such example: AOL (Pass et al., 2006) and MSN query logs (Zhang and Moffat, 2006) are composed for the 51.6% and 52.4% of duplicates, respectively.

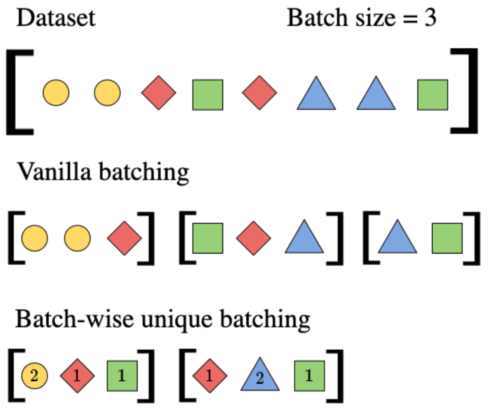

Figure 1: Effect of employing batch-wise unique samples. Consider a dataset with 8 samples of 4 distinct types (e.g. utterance text). Instead of inserting samples sequentially until the batch is full, by inserting only the first encountered sample per type (keeping track of the occurrences) the number of batches is decreased.

A similar phenomenon can be observed in recommender systems data, in which most of the entries involve a relatively small set of popular items (Cremonesi et al., 2010). Finally, in large scale conversational assistant data, the vast majority of user interactions is composed of commands and queries that are frequently expressed with minimal or no variation. When training on this redundant data, there will be instances in which: (i) the training input is the same, and (ii) the model has the same weights; in these instances repeated computations will occur, increasing training times.

In this paper, we propose a simple yet effective approach to reduce repeated computations during training by removing duplicates at the batch level, while accounting for frequency of the samples in the batch during the loss computation. This leaves the data distribution perceived by the model untouched and we empirically show that it leads to a model that has similar weights and has followed a similar trajectory in the parameter space during

[*] Equal contribution.

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 5: Industry Track, pages 235–247
July 10-12, 2023 ©2023 Association for Computational Linguistics

training, compared to a model trained without any data deduplication.

Our contribution is therefore four-fold: (i) we propose a novel online deduplication technique to mitigate the training burden over redundant datasets and provide rigorous mathematical reasoning backing its benefits; (ii) we show its effectiveness on both a real industrial datasets as well as artificially upsampled public datasets; (iii) we further provide a set of empirical analyses, including a study of the effect on the parameters' evolution and the difference in benefit when varying batch size; (iv) finally, we release a modular and extensible codebase[1], implementing deduplicator classes, easily reusable by the community. Even though the methodology is agnostic to both task and model, we experimentally validate its strengths on NER as it constitutes a critical task for large-scale conversational assistants, on which we show the duplication problem to be relevant. We believe this work fills a gap in the current research landscape, since deduplication techniques for training data appear to be an under-explored research territory, but also an increasingly pressing need in industry.

2 Related work

The success of language models pre-trained on large corpora, such as BERT (Devlin et al., 2018), raised awareness on optimization of training times and costs within the NLP community. Strubell et al. (2019) showed that carbon footprint of NLP research is following a concerning trend and spurred researchers to prioritize the development of computationally efficient algorithms. Countless directions have been explored by the community, from distillation techniques (Hinton et al. (2015)) used to produce smaller models (Sanh et al. (2019)) up to efficient computation frameworks to improve distributed training (Song et al. (2023)). To the best of our knowledge, only a few works in the literature address the problem of optimising training in presence of duplicates in the data. Lee et al. (2021) show the benefits of completely removing duplicates when pre-training large language models. Ya-Guan et al. (2020) propose a way to improve accuracy focusing on mini-batches and performing undersampling and oversampling in order to balance classes. Faghri et al. (2020) mention the possibility of reducing computation time by removing all but one of the duplicates in a mini-batch, although their work focuses on the optimal way to sample data to minimize gradient variance, and not on training time reduction. Similarly, Wang et al. (2016) propose a way to balance the training effort among batches, to improve Stochastic Gradient Descent (SGD) by minimizing gradient variance. Compared to these previous works, our approach focuses on reducing the repeated computation that occurs when training on redundant datasets, retaining model performance while being minimally invasive to the pre-existing setup.

3 Approach

A deep learning model $\mathcal{M}(\theta)$ is typically trained over a given dataset $\mathcal{D}$ with SGD (or one of its variants), estimating the gradient of a loss function with respect to the model weights θ, by iterating over the dataset in batches of fixed size. If $\mathcal{D}$ contains duplicates, then some of them might fall within the same batch, resulting in the same computations occurring multiple times. We propose to remove duplicates while building the batch, also accounting for the number of occurrences of the samples when computing the loss. In practice, first creating the batches and then removing the duplicates would result in smaller batch sizes and therefore in under-exploiting the parallel computation enabled by GPUs. Therefore, we keep the actual batch size the same, filling a batch with unique samples (ignoring repetitions when encountered, see fig. 1), but accounting for repetitions by multiplying the loss of a sample by its frequency in the batch. See appendix A.1 for the details of the procedure. By considering the number of repetitions of the samples, the size of the batches remains the same while virtually containing more samples, therefore reducing the required number of batches to iterate over the dataset and leading to training time reduction. We remark that the proposed technique does not make any assumption on the underlying task and model, but only affects the way data is loaded during training. Therefore, it is well suited for a production setting in which one wishes to reduce the burden of frequent re-training, while at the same time changing the setup as little as possible.

Motivation Our methodology is grounded on the following intuitions: (i) frequent samples should be given a larger weight than rarer ones, as they are most frequently encountered in the production traffic, and (ii) the model should be able to see the duplicated samples multiple times per epoch,

[1] Available at github.com/amazon-science/unique-batches.

i.e. at different stages of the model parameters evolution. While (i) explains why we should deduplicate and take sample frequency into account in the loss computation, (ii) explains the need to do it at the batch level. In section 6 we provide empirical support to these observations.

Training time reduction The reduction in training time comes from a direct reduction in the number of batches, as each batch virtually contains more samples than its actual batch size. We can define the virtual batch size B^{virtual} of batch b containing B unique objects o_i as

$$B^{\text{virtual}} = \sum_{i=1}^{B} \text{occurrences}(o_i, b). \tag{1}$$

Let N^{dup} be the number of duplicates in batch b, once it has been filled with B unique samples, then $B^{\text{virtual}} = B + N^{\text{dup}}$. The number of duplicates in a batch is a random quantity that depends on the dataset distribution and the batch size. As the randomness only regards N^{dup}, we have

$$\mathbb{E}\left\{B^{\text{virtual}}\right\} = B + \mathbb{E}\{N^{\text{dup}}\}. \tag{2}$$

The expected relative increase in batch size is then $B^{\text{inc}} = \mathbb{E}\left\{B^{\text{virtual}}\right\}/B$. Let N be the number of training samples. Let $M = \lceil \frac{N}{B} \rceil$ be the number of batches resulting from iterating over the samples with batch size B. Finally, let M' be the number of batches when using batch size $\mathbb{E}\left\{B^{\text{virtual}}\right\}$. By definition, we have

$$\begin{aligned}\mathbb{E}\left\{M'\right\} &= \left\lceil \frac{N}{\mathbb{E}\left\{B^{\text{virtual}}\right\}} \right\rceil \\ &= \left\lceil \frac{N}{BB^{\text{inc}}} \right\rceil = \left\lceil \frac{M}{B^{\text{inc}}} \right\rceil.\end{aligned} \tag{3}$$

If we assume that a mini-batch of size B can be processed in parallel, the time complexity of a pass over N samples to optimize d parameters is (Bottou and Bousquet, 2007) $\mathcal{O}(\frac{Nd}{B}) = \mathcal{O}(Md)$. Our approach introduces an $\mathcal{O}(N)$ step, while at the same time reducing the time complexity to $\mathcal{O}(M'd) = \mathcal{O}(\frac{Nd}{BB^{\text{inc}}})$. This expected reduction in training time complexity simply reflects the expected reduction in number of batches computed above. This leads the overall complexity to

$$\mathcal{O}\left(N + \frac{Nd}{BB^{\text{inc}}}\right) = \mathcal{O}\left(\frac{Nd}{B}\left(\frac{B}{d} + \frac{1}{B^{\text{inc}}}\right)\right) \tag{4}$$

Since usually the number of parameters to optimize is much larger than the batch size, we can assume $B \ll d$ from which it follows $\frac{B}{d} \approx 0$, hence the overall complexity is

$$= \mathcal{O}\left(\frac{1}{B^{\text{inc}}}\frac{Nd}{B}\right) \tag{5}$$

meaning the expected reduction in training time is proportional to the expected relative increase in batch size.

Accounting for duplicates Removing duplicates in the batch also removes the influence of repeated samples, hence removing the larger contribution to the loss of more frequent samples. This leads to different gradients and therefore different training evolution. To counter this, the contribution to the loss of each sample o_i is re-weighted by

$$\textbf{frequency}(o_i, b) = \frac{\text{occurrences}(o_i, b)}{B^{\text{virtual}}} \tag{6}$$

thus resulting in the same loss signal we would have when using the virtual batch size instead. We delve into more details on the effect of including the frequency signal in the loss in section 6.

4 Boost estimation

Estimating the expected increase in virtual batch size is important for two reasons. First, the virtual batch size allows us to compute the expected reduction in the number of batches required to iterate over the dataset. This in turn provides an estimate for the reduction in training time (see eq. (3)) without the need to actually run any training. Second, it is common practice to scale the learning rate when increasing the batch size (Krizhevsky, 2014; Goyal et al., 2017), thus having an estimate of the virtual batch size helps correcting the learning rate. This is discussed more in detail in section 7.

To the best of our knowledge there is no closed form solution for $\mathbb{E}\left\{B^{\text{virtual}}\right\}$ in eq. (2). Therefore we derive a solution for the number of duplicates d in a batch b of size n, and then invert the formula to compute the expected number of unique samples in b as $u = n - d$. This way a numerical solution to the original problem can be found by iterating over the possible values of $n = 1, \ldots, N$, up to the one that yields $u = B$ unique samples. In the following we introduce the formula for d and leave the details of its derivation in appendix A.2. Consider a dataset

$$\mathcal{D} = \{x_i \mid x_i \in \mathcal{C}, i = 1, \ldots, N\} \tag{7}$$

Alias	Name	Scope	# Copies	Weighted
Base	Baseline	D	c_i	✗
DU	Dataset-wise Unique	D	1	✗
DWU	Dataset-wise Weighted Unique	D	1	✓
DL	Dataset-wise Logarithmic	D	$\log(c_i)$	✗
BU	Batch-wise Unique	B	1	✗
BWU	Batch-wise Weighted Unique	B	1	✓

Table 1: The deduplication techniques under consideration. *Scope* regards whether samples are deduplicated at the batch (B) or at the dataset (D) level, *weighted* approaches account for the number of frequencies during the loss computation, and *# copies* determines how many repetitions are left after deduplication. The first row corresponds to the non-deduplication baseline.

of N samples, some of which may be duplicates, taken from a collection $\mathcal{C} = \{o_1, \dots, o_C\}$ of C distinct objects. Let $k_1, \dots, k_C$ denote the number of occurrences in $\mathcal{D}$, such that $k_1 + \dots + k_C = N$. Then, we can show that:

$$d = \sum_{i=1}^{C} \left(n\frac{k_i}{N} - 1 + \frac{\binom{N-k_i}{n}}{\binom{N}{n}} \right). \qquad (8)$$

Effect of sampling strategy The estimate in eq. (8) assumes batches are formed by sampling uniformly at random from the dataset. However, NLP practitioners often rely on sampling strategies that optimize memory consumption, such as *Bucketing by Sequence Length* (Khomenko et al., 2016). Samples are distributed in *buckets* based on their length, and then batches are formed sampling uniformly at random from these. In such a scenario, the boost estimation in eq. (8) still holds, but on each bucket individually. The overall expected number of duplicates can be computed as a weighted average of the estimates on each bucket, weighted by the bucket size. We highlight that if samples are grouped in buckets by length, then all of the duplicates of one sample will fall in the same bucket. This has the effect of increasing the number of expected duplicates in each batch (N in eq. (8) is smaller), leading to larger training time reduction in realistic scenarios that use bucketing.

5 Experimental setting

The proposed approach is tested on the task of NER (Tjong Kim Sang and De Meulder, 2003). Two samples (also referred to as utterances) are considered duplicates if they share the same *annotation*, i.e. word-level tokens and corresponding ground truth NER labels in the dataset.

The experiments are performed on datasets used for training a large-scale conversational assistant (referred to as *internal* in the following) and also on publicly available data. The internal datasets comprise live traffic utterances, de-identified for privacy regulations and annotated to enable supervised training of deep learning models for solving the NER task. The datasets exhibit varying degree of skewness in their redundancy, and we refer to them as Internal$_{\text{MS}}$, Internal$_{\text{VS}}$, Internal$_{\text{XS}}$, meaning *mildly skewed*, *very skewed* and *extremely skewed*, respectively. Given the artificial deduplication of manually curated datasets, we upsample a public dataset to mimic the redundancy observed on internal data. The *MITRestaurant* dataset (Liu et al., 2013), consisting of restaurant-related queries, is chosen for the semantic similarity of its queries to the utterances found in the internal datasets. From this dataset, upsampling is performed to arrive at three datasets with redundancy ratios of $r_1 = 0.5, r_2 = 0.7$ and $r_3 = 0.9$. We refer to these artificially upsampled public datasets as MIT$_{\text{MS}}$, MIT$_{\text{VS}}$, MIT$_{\text{XS}}$, with the same meaning of the acronyms. Given the redundancy ratio r of interest for each of the three datasets, these are generated as follows. First, the number of duplicates to draw is computed, according to the above probability distribution (eq. (8)), as $d = \frac{r}{1-r}n$. Then, to generate a realistic distribution, the fact that shorter utterances are more frequent (Borbély and Kornai, 2019) and thus more likely to be duplicated is leveraged. In a conversational assistant, for instance, we expect to observe more frequently short utterances such as "yes" or "stop" than long sentences related to some more specific queries. This heuristic is implemented by sampling the d duplicates according to a power-law over the length of the utterance in characters, for which an utterance u_i with length l_i is sampled with probability $p(u_i) = \frac{1}{(l_i)^\alpha}$. The exponent of the power-law α is set to 3 for MIT$_{\text{MS}}$ and MIT$_{\text{VS}}$, while it is set to 5 for MIT$_{\text{XS}}$. See table 2 for more statistics on these artificially upsampled datasets.

We compare the proposed approach with various baselines (see table 1), which either deduplicate at the batch level or at the dataset level, can either account for the number of repetitions during the loss computation or ignore them, and keep one or more copies per sample. The comparison is drawn in terms of training steps and time, as well as performance of the trained model in terms of micro-

Dataset	Size	Redundancy	# Named Entities
MITRestaurant	9180	–	17
MIT$_{\text{MS}}$	18360	0.5	
MIT$_{\text{VS}}$	30600	0.7	
MIT$_{\text{XS}}$	91801	0.9	

Table 2: Statistics of the public dataset used in the experiments, and the three artificially upsampled versions.

F1 score, considering averages across 5 seeds.

All the deduplication methods are applied to the training of a model with the same architecture, across all experiments. In particular, we employ a simple NER architecture that obtains contextual word embeddings from a pre-trained DistilBERT encoder (Sanh et al., 2019). The embeddings are then fed to a Multi-Layer Perceptron (MLP) to map into the label space. The models are trained to convergence with early stopping.

We remark that what changes across the experiments is the deduplicator and not the model that is employed. The latter is in fact defined by the same architecture and hyperparameters, except for the learning rate that is adjusted as described in section 7 to account for the difference in virtual batch size. Refer to appendix A.4 for further details.

6 Experimental results

Results on internal data Table 3 reports the results on the three internal datasets. We can see how the only approach that reduces training times while also keeping model performance intact is removing duplicates at the batch level: training times are significantly reduced (-46.5% on average, -87% at best) and the model evaluation metric is almost on par (-0.1% on average, -0.2% at worst). On the other hand, naively removing duplicates at the dataset level is suboptimal: the strategy is consistently the best approach in terms of reduction in training time (-91.3% on average, -97% at best), but also the worst in terms of F1 score of the resulting model (-3.1% on average, -5.8% at worst). Finally, only keeping a logarithmic portion of the duplicates at the dataset level allows to reduce the training times less (-65.2% on average, -84% at best), but with a milder worsening of model performance (-0.3% on average, -0.5% at worst).

We observe that introducing the sample weighting term in the loss when deduplicating batch-wise (BWU) does not seem to result in a significant improvement over the un-weighted variant (BU). We hypothesize this to be a consequence of leaving the pre-trained BERT encoder frozen during training. To test this hypothesis, we experiment also on a simpler LSTM-based model (Hochreiter and Schmidhuber, 1997), without pre-training (see table 7) and find that the weighted variant (BWU) is on average 22.8% faster than the un-weighted variant (BU) since it leads to faster convergence. Finally, it is worth mentioning that, if keeping perfectly on-par model performance is not critical, simpler deduplicators (e.g. DL) may achieve better training time reduction.

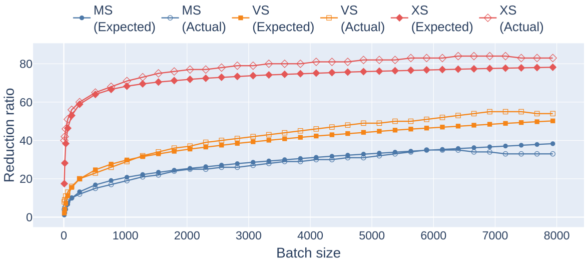

Figure 2: Expected and actual reduction ratio versus batch size for the three upsampled datasets.

Results on public data Table 4 reports the results on public data, where we observe similar patterns to the ones on internal data. Again, the only approach consistently reducing training times while also keeping model performance on par is BWU: it leads to a training time reduction of -23% on average, and -61.1% at best, while exhibiting no model performance drop on average, and -0.1% at worst. We observe again similar results between BWU and BU, and carry out the same test on a simpler LSTM also on public data (see table 8), with similar results. Removing duplicates at the dataset level is still the best approach in terms of time reduction (-32% on average, -68.7% at best), but at the cost of worst model performance decrease (-9.2% on average, -21.2% at worst). The DL deduplicator is less competitive on public data, with an average and best time reduction of -32.2% and -68.7%, respectively, and a model performance decrease of -0.83% on average and -1.4% at worst.

Parameters evolution We hypothesize that including the frequency information in the loss, and thus having an almost identical loss signal to the non-deduplication baseline, leads to models having similar weights. To test this hypothesis, we retain parameter vectors during training with all the dedu-

deduplicator	BS = 512									BS = 1024								
	MIT$_{\text{MS}}$			MIT$_{\text{VS}}$			MIT$_{\text{XS}}$			MIT$_{\text{MS}}$			MIT$_{\text{VS}}$			MIT$_{\text{XS}}$		
	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑
Base	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
DU	-34.0%	-31.0%	-0.7%	-83.0%	-83.0%	-0.6%	-87.0%	-87.0%	-0.2%	-54.0%	-53.0%	-0.9%	-75.0%	-73.0%	-0.3%	-83.0%	-82.0%	-0.2%
DWU	**-86.0%**	**-86.0%**	-5.8%	**-93.0%**	**-93.0%**	-1.9%	**-97.0%**	**-97.0%**	-2.1%	**-88.0%**	**-87.0%**	-5.3%	**-91.0%**	**-91.0%**	-2.2%	**-94.0%**	**-94.0%**	-1.3%
DL	-32.0%	-30.0%	-0.5%	-75.0%	-74.0%	-0.2%	-83.0%	-83.0%	-0.2%	-47.0%	-45.0%	-0.4%	-76.0%	-75.0%	-0.3%	-84.0%	-84.0%	-0.3%
BU	+11.0%	+15.0%	-0.0%	-58.0%	-57.0%	-0.2%	-73.0%	-70.0%	**-0.1%**	-18.0%	-16.0%	-0.1%	-34.0%	-29.0%	-0.2%	-75.0%	-72.0%	**-0.1%**
BWU	-3.0%	+0.0%	**+0.1%**	-52.0%	-50.0%	**-0.1%**	-75.0%	-72.0%	**-0.1%**	-26.0%	-23.0%	**+0.0%**	-51.0%	-47.0%	**+0.1%**	-89.0%	-87.0%	**-0.1%**

Table 3: Comparison of the deduplicators on the internal datasets. Highlighting in bold best results column-wise. While BWU is not the best technique in terms of training steps and time alone, it is the only one that can reduce training time while maintaining model performance.

deduplicator	BS = 512									BS = 1024								
	External$_{\text{MS}}$			External$_{\text{VS}}$			External$_{\text{XS}}$			External$_{\text{MS}}$			External$_{\text{VS}}$			External$_{\text{XS}}$		
	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑
Base	696	99.2s	0.915	1046	136s	0.929	1987	224.6	0.955	450	124.6s	0.914	700	178.4s	0.927	1800	394.8s	0.956
DU	-32.0%	-11.5%	-1.1%	-51.2%	-30.4%	-2.1%	-75.8%	-55.8%	-6.5%	-46.6%	**-30.0%**	-1.4%	-61.4%	-47.8%	-3.0%	-86.7%	-75.2%	-8.8%
DWU	**-39.8%**	**-21.2%**	-3.1%	**-52.9%**	**-32.5%**	-2.9%	**-85.2%**	**-72.3%**	-11.9%	**-48.2%**	**-30.0%**	-4.4%	**-65.6%**	**-51.9%**	-11.9%	**-86.8%**	**-75.6%**	-21.2%
DL	-26.2%	-4.4%	-0.3%	-36.9%	-16.5%	-0.4%	-66.9%	-46.0%	-0.9%	-40.0%	-21.0%	-0.7%	-52.9%	-36.6%	-1.3%	-80.0%	-68.7%	-1.4%
BU	-16.7%	-5.4%	-0.1%	-18.4%	-8.7%	**-0.1%**	-45.6%	-31.4%	**+0.3%**	-20.0%	-8.7%	-0.1%	-31.4%	-20.3%	-0.1%	-70.8%	-61.7%	-0.2%
BWU	-5.6%	+9.1%	**+0.2%**	-27.7%	-17.5%	**-0.1%**	-53.8%	-40.6%	+0.0%	-20.0%	-8.3%	**+0.0%**	-31.4%	-19.3%	**+0.0%**	-70.4%	-61.1%	**-0.1%**

Table 4: Comparison of the deduplicators on the public datasets. Following the same data presentation conventions as in table 3. We observe very similar results to the ones on the internal datasets, with BWU being the only deduplicator able to consistently reduce training time and keep model performance on par with the non-deduplication baseline. Absolute results are available in table 9.

plicators; looking at table 5 we can see how indeed the BWU deduplicator trains a model that is the closest (in parameter space) to the model trained without deduplication, among the tested deduplicators. Furthermore, we find that this behavior is the result of a stronger property of our approach. Let us define the distance between trajectories in parameter space as

$$d_{trajectory}(\mathbf{A}, \mathbf{B}) = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{A}_i - \mathbf{B}_i\|_2 \qquad (9)$$

where $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times d}$ are matrices of n parameter vectors of dimension d, and $\|\cdot\|_2$ is the Euclidean norm. Essentially, we are considering the average point-wise Euclidean distance between corresponding pairs of points along the two trajectories in parameter space. Then, our finding is that employing the BWU deduplicator during training leads to a trajectory in parameter space that is the closest one to that of a model trained without any deduplication (see again table 5). One can qualitatively appreciate this property by resorting to t-SNE (van der Maaten and Hinton, 2008) to project the d-dimensional vectors down to 2D, and visualizing the trajectories in parameter space, as can be seen in fig. 3.

deduplicator	d_{final} ↓	$d_{trajectory}$ ↓
DU	56.8	40.4
DWU	74.1	52.6
DL	49.9	35.15
BU	48.6	32.3
BWU	45.5	29.7

Table 5: Comparison of the deduplicators in terms of Euclidean distance of the final parameter vector, and the overall trajectory in parameter space, from the one of the model trained with no deduplication

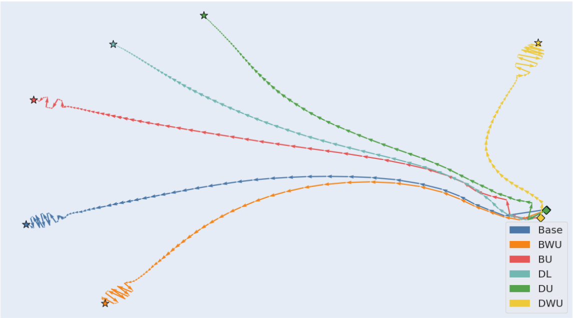

Figure 3: Visualization of the trajectories in parameter space followed by the same model, with same initialization, when trained with different deduplicators. Our proposed deduplicator (BWU) results in the closest trajectory to the non-deduplicated one (Base) across all training. The figure has been obtained following the same procedure used by Huang et al. (2019).

7 Discussion

Influence of batch size As can be seen in Figure 2, the expected number of duplicates increases with the batch size. Intuitively, given a finite set of elements that contains repetitions, the larger a sample, the more likely it is to have repetitions. The plot clearly shows how the greatest increase in the number of duplicates is obtained when passing from small batch sizes to over 1000. The maximum possible reduction in number of batches would trivially be obtained when the batch size is equal to the total number of samples, when the reduction would be exactly the redundancy ratio of the dataset.

Learning rate It is advisable to scale the learning rate when increasing the batch size, as the gradient computed over a larger batch size is a more accurate estimate of the real gradient and should therefore provide a more reliable direction. Small batches naturally provide more chaotic gradients and therefore a large learning rate may result in "overshooting" and missing the minima. A common approach is to increase the learning rate as the square root of the batch size increase factor (Krizhevsky, 2014), or to scale it linearly with the batch size (Goyal et al., 2017). We adopt the latter in this work, multiplying the learning rate by the expected increase in batch size computed as in section 3.

8 Conclusions

In this paper, we address the problem of reducing training time when using redundant datasets, proposing a novel approach agnostic to task and model that results in a significant time reduction without waiving performance. The approach is compared with various deduplication methods on the task of Named Entity Recognition, on both industrial and public datasets. The comparison shows that our approach is the only one to significantly reduce training time while maintaining the same model performance metrics, with observed boosts in training time of $23\%, 47\%$ and 87% on datasets used to train models for a large-scale conversational assistant. Various analysis are conducted, on the tradeoff with batch size and on how learning trajectories are modified. We also provide a theoretically sound procedure to estimate the expected reduction, allowing practitioners to assess the benefits before employing the method. Finally, a modular and reusable codebase is released to foster further research in the area.

We believe that this approach can have a high impact on the industry, where large, expensive models are often trained on datasets containing redundant user queries or items. This reduction in training times may in fact allow for faster experimental iterations while cutting times and costs, also reducing the carbon footprint of deep learning models.

As future work we plan to (i) leverage the generality of the approach on other tasks that exhibit high redundancy in data, like semantic search and (ii) study a more relaxed definition of equality between two samples, to allow considering two samples the same if they only differ up to a tolerated quantity.

Limitations

An important limitation of our contribution lies in breadth of the experimental validation. We decided to focus our experimentation on the setup where we encountered the issue of redundant datasets: NER for a large-scale conversational assistant. While it is true that our approach does not make any assumption neither about the task nor about the model architecture, and while we also provide a rigorous proof to support the estimated gain in terms of training time, the extent to which our approach remains the best time-accuracy trade-off when other tasks are considered is not explored in this work.

An additional limitation stems from the lack of absolute results on the internal datasets, as the latter can only be disclosed as relative improvements over a baseline due to internal policy. We attempt to mitigate this by reporting full results on an open dataset, but as we mention we have to resort to upsampling given the artificial deduplication that manually curated datasets incur before publishing.

Ethics Statement

The carbon footprint generated by the NLP community has shown a concerning trend in recent years. Similarly, the development and maintenance of production models using large neural networks in an industry setting has a non negligible negative impact on our planet. While our technique offers a significant advantage only in presence of duplicates in the training data, which might not always be the case, we see it as a small but tangible contribution towards a more sustainable research. Furthermore, a reduction in training times also directly translates to a reduction in costs, therefore works like ours also contribute to the democratization of language models development and their applications. Finally we note how our approach does not lead to the introduction of any new bias, since it leaves the data distribution observed by the model during training identical to the one of the initial dataset.

Acknowledgements

Donato Crisostomi is partially supported by the PRIN 2020 project n.2020TA3K9N. "LEGO.AI".

References

Gábor Borbély and András Kornai. 2019. Sentence length. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 114–125, Toronto, Canada. Association for Computational Linguistics.

Léon Bottou and Olivier Bousquet. 2007. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, page 39–46, New York, NY, USA. Association for Computing Machinery.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

X. G. Duan. 2021. Better understanding of the multivariate hypergeometric distribution with implications in design-based survey sampling.

Fartash Faghri, David Duvenaud, David J. Fleet, and Jimmy Ba. 2020. A study of gradient variance in deep learning. *CoRR*, abs/2007.04532.

WA Falcon. 2019. Pytorch lightning.

Marco Ferrante and Nadia Frigo. 2012. On the expected number of different records in a random sample. *arXiv: Probability*.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

W. Ronny Huang, Zeyad Emam, Micah Goldblum, Liam Fowl, J. K. Terry, Furong Huang, and Tom Goldstein. 2019. Understanding generalization through visualizations.

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

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Alex Krizhevsky. 2014. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2021. Deduplicating training data makes language models better. *CoRR*, abs/2107.06499.

Jingjing Liu, Panupong Pasupat, Scott Cyphers, and Jim Glass. 2013. Asgard: A portable architecture for multilingual dialogue systems. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8386–8390.

Greg Pass, Abdur Chowdhury, and Cayley Torgeson. 2006. A picture of search. In *Proceedings of the 1st International Conference on Scalable Information Systems*, InfoScale '06, page 1–es, New York, NY, USA. Association for Computing Machinery.

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

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

David A Schum. 2001. *The evidential foundations of probabilistic reasoning*. Northwestern University Press.

Jaeyong Song, Jinkyu Yim, Jaewon Jung, Hongsun Jang, Hyung-Jin Kim, Youngsok Kim, and Jinho Lee. 2023. Optimus-cc: Efficient large nlp model training with 3d parallelism aware communication compression. *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.

Linnan Wang, Yi Yang, Martin Renqiang Min, and Srimat T. Chakradhar. 2016. Accelerating deep neural network training with inconsistent stochastic gradient descent. *CoRR*, abs/1603.05544.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. Huggingface's transformers: State-of-the-art natural language processing.

Qian Ya-Guan, Ma Jun, Zhang Xi-Min, Pan Jun, Zhou Wu-Jie, Wu Shu-Hui, Yun Ben-Sheng, and Lei Jing-Sheng. 2020. Emsgd: An improved learning algorithm of neural networks with imbalanced data. *IEEE Access*, 8:64086–64098.

Omry Yadan. 2019. Hydra - a framework for elegantly configuring complex applications. Github.

Yuye Zhang and Alistair Moffat. 2006. Some observations on user search behaviour. *Aust. J. Intell. Inf. Process. Syst.*, 9:1–8.

A Appendix

A.1 Schedule generation

Algorithm 1 describes in pseudo-code the proposed procedure to generate a batching schedule that removes duplicates at the batch level. Notice that we consider two utterances identical when their *annotation*, i.e. both sequence of word-level tokens and corresponding NER labels, are identical.

Procedure 1 Proposed deduplication approach

```
Input: dataset $data$, batch size $B$
Output: schedule for dataloader
  occurrences ← []
  schedule ← []
  $I_b \leftarrow \{\}$                       ▷ IDs seen in batch
  sample_pos$_b$ ← {}              ▷ sample pos in batch
  occ$_b$ ← []                     ▷ occurrences in batch
  $C \leftarrow B$                      ▷ capacity of current batch
  for all sample $s$ in data do
      $s_{\text{pos}}$ ← position of the sample
      $s_{\text{id}}$ ← unique id of the sample
      if $s_{\text{id}} \notin I_b$ then
          schedule += $s_{\text{pos}}$
          $I_b$ += $s_{\text{id}}$
          sample_pos$_b$[$s_{\text{id}}$] ← len(occ$_b$)
          occ$_b$.append(1)
          $C$ −= 1
      else
          occ$_b$[sample_pos$_b$[$s_{\text{id}}$]] += 1
      end if
      if ($C = 0$) ∨ ($s$ is the last sample) then
          occurrences.append(occ$_b$)
          $C \leftarrow B$
          occ$_b$ ← []
          sample_pos$_b$ ← {}
          $I_b \leftarrow \{\}$
      end if
  end for
  return schedule, occurrences
```

In practice, the schedule is then used to load samples during training, that are then fed to the model to obtain class scores $\mathbf{z} \in \mathbb{R}^{N \times C \times L}$, with N, L, C the number of samples, the sequence length, and the number of classes, respectively. Then, the occurrences (normalized as relative frequencies f_i) are integrated in the loss function as follows

$$\mathcal{L} = -\sum_{i=1}^{N} f_i \cdot \frac{1}{L} \sum_{j=1}^{L} \log \frac{\exp(z_{n,j,y_n})}{\sum_{c=1}^{C} \exp(z_{n,j,c})} \tag{10}$$

where $\mathbf{y} \in \mathbb{R}^n$ is the vector of ground truth labels.

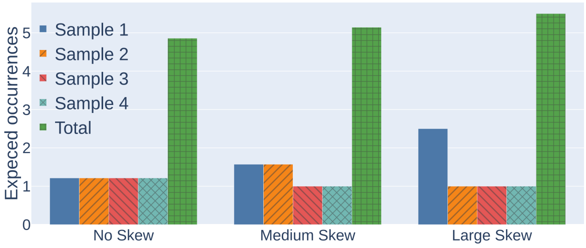

Figure 4: Effect of the skewness of frequency distribution over the number of duplicates in a batch. Consider a dataset with 8 samples of 4 distinct types (redundancy factor $\frac{1}{2}$), with various frequency distribution skewness: no skew ($[2, 2, 2, 2]$), medium skew ($[3, 3, 1, 1]$), large skew ($[5, 1, 1, 1]$). Consider a batch of size 4. More skewed distributions lead to more occurrences in the batch, hence fewer batches to cover the dataset.

A.2 Derivation of boost estimate

Aim of this section is to formally derive the formula for the expected virtual batch size reported in Equation (8) given the distribution of duplicates in the dataset, reported in section 4.

Problem Formalisation Consider a dataset $\mathcal{D} = \{x_1, \dots, x_N\}$ with N objects, some of which might be repeated more than once. As described in section 3, we fill a batch b by sampling B^{virtual} objects $\mathcal{O} = \{x_{p_1}, \dots, x_{p_{B^{\text{virtual}}}}\}$ so that $\mathcal{O}$ contains B distinct elements. We want to compute the expected value $\mathbb{E}\left\{B^{\text{virtual}}\right\}$. This problem has some analogies with a generalized version of the coupon collector problem in which one wants to find how many samples with replacement are required to obtain a certain number of unique objects (Ferrante and Frigo, 2012), but in our case there is no replacement. To the best of our knowledge there is no closed form solution for this version of the problem, and therefore, we derive a solution for the expected number of duplicates $d = N^{\text{dup}}$ in a batch b of size n. This way a numerical solution of the original problem can be found by iterating over the possible values of $n = 1, \dots, N$, up to the one that yields $u = B$ unique samples.

Estimate Derivation Consider now a dataset $\mathcal{D}$ of size N instead as a collection $\mathcal{C} = \{o_1, \dots, o_C\}$ of C distinct objects each associated with its number of occurrences $k_1, \dots, k_C$ such that $k_1 + \cdots + k_C = N$. Consider a random sample without replacement $X_1, \dots, X_n$ over $\mathcal{D}$, with $n < N$ representing the batch size. Now introduce C counting

variables $Z_1, \dots, Z_C$, such that Z_i counts occurrences of o_i in the sample.
Formally:

$$Z_i = \sum_{j=1}^{n} \mathbb{1}(X_j = o_i) \tag{11}$$

The counting variables introduced above follow, by definition, a multivariate hypergeometric distribution (Duan, 2021), characterized by probability mass function

$$p_{Z_1,\dots,Z_C}(z_1, \dots, z_C) = \frac{\prod_{i=1}^{C} \binom{k_i}{z_i}}{\binom{N}{n}} \tag{12}$$

and expected value

$$\mathbb{E}(Z_i) = n\frac{k_i}{N}. \tag{13}$$

We want to compute the expected number of duplicates in the sample $d = \mathbb{E}\left(N^{\text{dup}}\right)$. The number of duplicates of o_i in the sample is $Z_i - 1$, since the first occurrence is not a duplicate; therefore in total we have:

$$N^{\text{dup}} = \sum_{i=1}^{C} \max\left(Z_i - 1, 0\right). \tag{14}$$

By linearity of expectations, it holds that:

$$\mathbb{E}\left(N^{\text{dup}}\right) = d = \sum_{i=1}^{C} \mathbb{E}\left(\max\left(Z_i - 1, 0\right)\right) \tag{15}$$

Let $d_i = \mathbb{E}\left(\max\left(Z_i - 1, 0\right)\right)$, then by the chain rule of expectations (also known as *Law Of The Unconscious Statistician* (Schum, 2001)) it follows:

$$d_i = \sum_{j=0}^{k_i} \max(j-1, 0) p_{Z_i}(j)$$

where $p_{Z_i}(j)$ is the probability mass of Z_i in j. Now, noticing that the first two contributions to the sum are 0, we have

$$\begin{aligned} d_i &= \sum_{j=2}^{k_i} \max\left(j-1, 0\right) p_{Z_i}(j) \\ &= \left[\mathbb{E}\left(Z_i\right) - p_{Z_i}(1)\right] - \left[1 - p_{Z_i}(0) - p_{Z_i}(1)\right] \end{aligned} \tag{16}$$

and substituting from eq. (13) we get

$$d_i = n\frac{k_i}{N} + p_{Z_i}(0) - 1. \tag{17}$$

The value $p_{Z_i}(0)$ denotes the probability of object o_i having zero elements in the sample. Now, if the batch size is larger than the number of objects different from o_i, i.e. $k_1 + \cdots + k_{i-1} + k_{i+1} + \cdots + k_C < B$, then the sample will contain at least one occurrence of object o_i, and in that case $p_{Z_i}(0) = 0$. However in real-world scenarios this is quite unlikely, since usually $B \ll k_1 + \cdots + k_{i-1} + k_{i+1} + \cdots + k_C$, therefore we have

$$p_{Z_i}(0) = \frac{\binom{N-k_i}{n}}{\binom{N}{n}} \tag{18}$$

and putting it all together, we get:

$$d = \sum_{i=1}^{C} \left(n\frac{k_i}{N} - 1 + \frac{\binom{N-k_i}{n}}{\binom{N}{n}} \right). \tag{19}$$

As mentioned before, with this closed-form expression we now have a proxy to numerically compute the actual quantity of interest, that is $\mathbb{E}\{B^{\text{virtual}}\} = n = u + d = B + N^{\text{dup}}$. See algorithm 2 for the procedure to compute this numerical solution.

Procedure 2 Estimated boost computation

```
Input: dataset distribution $k_1, \dots k_C$, batch size $B$
Output: estimated boost
  $N \leftarrow \sum_{i=1}^{C} k_1$                ▷ dataset size
  $l \leftarrow 0$
  $r \leftarrow N$
  while $r > l$ do                  ▷ binary search
      $n = \left\lceil \frac{(r-l)}{2} \right\rceil$
      Compute $d$ with eq. (19)
      if $n - d = B$ then
          $r \leftarrow \frac{n}{B}$           ▷ Increase in batch size
          return $\left(1 - \frac{1}{r}\right)$
      else if $n - d > B$ then
          $r = n - 1$
      else
          $l = n + 1$
      end if
  end while
```

A.3 Effect of frequency distribution

We remark that the reduction factor depends directly on the number of duplicates in the batch, and not on the overall redundancy factor. This can be appreciated in fig. 4. Three datasets with the same redundancy factor, but different skewness, lead to differt virtual batch sizes. In fact, the distribution with most skewness in frequency results in the largest number of duplicates, as it is easier to draw repeatedly a very frequent object than doing so for one of the many different objects contributing equally to the total redundancy in the dataset.

A.4 Implementation details

The experiments have been run on `p3.2xlarge` EC2 instances[2], equipped with a NVIDIA Tesla V100 GPU[3]. As optimization framework, PyTorch (Paszke et al., 2019) has been used, along with PyTorch Lightning (Falcon, 2019) and Hydra (Yadan, 2019) for easier and faster development and experiment executions. Table 6 reports the hyperparameters used to train the models with the various deduplicators. We remark that the deduplicators themselves have no hyperparameters, therefore the table lists the hyperparameters of the models. We used pretrained distilled BERT (Sanh et al., 2019) models from HuggingFace (Wolf et al., 2019) as well as LSTM-based (Hochreiter and Schmidhuber, 1997) models trained from scratch. Both types of models feature a two-layer, fully-connected MLP mapping word embeddings into label-space. As for the BERT-based models, the encoder is a `distilbert-base-cased`[4] for the English-only public dataset, while a `distilbert-base-multilingual-cased`[5] has been used for the internal data. In both cases the encoder weights are not updated during training. Subword token-level embeddings are obtained by summing the hidden states of the last 3 layers of the encoder; then, average pooling is used to obtain word-level embeddings. As for the LSTM-based models, they use an embedding layer exploiting a simple word-level vocabulary (built considering all the corpus).

The models are trained with the Adam (Kingma and Ba, 2014) optimizer to convergence with early stopping, monitoring the loss values on a held-out validation set. Learning rate is increased for the models trained with the Batchwise Weighted Unique deduplicator, as mentioned in section 5, to reflect the increase in (virtual) batch size. Note that this is not the case for the Batchwise Unique deduplicator, since while both fill the batch with unique samples, ignoring duplicates, only the former has an impact on the training dynamics due to the introduction of the sample-wise weight in the loss. In fact, the latter neglects the duplicates' contribution to the loss, therefore does not lead to an actual increase in the batch size, while the former does, hence the need to increase the learning rate accordingly.

A.5 Additional Results

After observing that sample weighting does not result in a significant improvement against the un-weighted batch-wise unique deduplicator, we investigate whether this is a flaw of the proposed deduplication technique or a consequence of using a pre-trained BERT encoder. As mentioned in Section 6, the latter turns out to be the case. In fact, training a simpler LSTM-based model from scratch, we get the results reported in Table 8. We can see how indeed the weighted variant is consistently on-par or superior to the un-weighted variant in terms of predictive performance, despite needing consistently less training steps, for both tested batch sizes. The effect becomes more noticeable as the redundancy and skewness in the dataset increases, as we would expect, with BU and BWU being almost comparable on the MS dataset with batch size 512, while the latter overcomes the former using almost half the training steps on the XS dataset with batch size 1024.

While the margin between the BU and BWU deduplicator is not as close on internal data, as observed on the public data, we repeat the experiment on the LSTM-based model also on the former. The results are reported in Table 7 and are quite similar to the ones reported in Table 8 on the public dataset.

[2] https://aws.amazon.com/ec2/instance-types/p3/

[3] https://www.nvidia.com/en-us/data-center/v100/

[4] https://huggingface.co/distilbert-base-cased

[5] https://huggingface.co/distilbert-base-multilingual-cased

Parameter	Value	
	BERT	**LSTM**
Learning Rate	1e-3	
Optimizer	Adam	
Max epochs	30	
Embedding size	768	50
Hidden size	256	
Dropout	0.5	0.2
Activation	ReLU	
Validation split	0.1	
Early stopping metric	Validation loss	
Early stopping delta	1e-4	
Early stopping patience	3	
Clipping gradient norm	10	

Table 6: Hyperparameter values for the two types of deep neural network used in the experiments.

deduplicator	BS = 512									BS = 1024								
	Internal$_{\text{MS}}$			Internal$_{\text{VS}}$			Internal$_{\text{XS}}$			Internal$_{\text{MS}}$			Internal$_{\text{VS}}$			Internal$_{\text{XS}}$		
	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑
Base	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
DU	-45.0%	-44.0%	-0.1%	-77.0%	-76.0%	-0.1%	-84.0%	-83.0%	-0.2%	-45.0%	-44.0%	-0.2%	-75.0%	-75.0%	-0.2%	-81.0%	-81.0%	-0.2%
DWU	-61.0%	-58.0%	-3.6%	-87.0%	-85.0%	-1.2%	-92.0%	-92.0%	-0.8%	-60.0%	-56.0%	-3.5%	-85.0%	-82.0%	-1.1%	-90.0%	-89.0%	-0.6%
DL	-36.0%	-34.0%	-0.1%	-72.0%	-72.0%	-0.1%	-84.0%	-83.0%	-0.1%	-33.0%	-31.0%	-0.2%	-71.0%	-70.0%	-0.1%	-79.0%	-79.0%	-0.1%
BU	-3.0%	+4.0%	+0.0%	-32.0%	-21.0%	+0.0%	-69.0%	-57.0%	+0.0%	+0.0%	+13.0%	+0.0%	-42.0%	-27.0%	-0.0%	-69.0%	-50.0%	-0.0%
BWU	-7.0%	+4.0%	+0.1%	-37.0%	-27.0%	+0.0%	-77.0%	-67.0%	-0.1%	-19.0%	-6.0%	-0.0%	-46.0%	-28.0%	-0.0%	-80.0%	-68.0%	-0.0%

Table 7: Comparison of the deduplicators on the internal datasets, when training a LSTM-based model from scratch.

deduplicator	BS = 512									BS = 1024								
	MIT$_{\text{MS}}$			MIT$_{\text{VS}}$			MIT$_{\text{XS}}$			MIT$_{\text{MS}}$			MIT$_{\text{VS}}$			MIT$_{\text{XS}}$		
	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑
Base	504	16s	0.925	662	20s	0.953	1526	47s	0.974	354	15s	0.92	489	21s	0.952	979	43s	0.973
DU	387	12s	0.912	422	13s	0.939	441	14s	0.945	240	11s	0.9	240	11s	0.913	240	11s	0.903
DWU	297	10s	0.828	316	11s	0.867	377	13s	0.903	211	10s	0.821	227	11s	0.857	240	12s	0.888
DL	392	12s	0.92	475	15s	0.948	595	19s	0.969	270	12s	0.912	330	15s	0.942	360	16s	0.964
BU	451	15s	0.924	546	18s	0.951	655	26s	0.973	352	17s	0.919	437	22s	0.947	421	28s	0.971
BWU	398	14s	0.925	483	16s	0.954	403	16s	0.975	304	15s	0.922	340	17s	0.949	223	15s	0.974

Table 8: Comparison of the deduplicators on public datasets, when training a LSTM-based model from scratch.

deduplicator	BS = 512									BS = 1024								
	MIT$_{\text{MS}}$			MIT$_{\text{VS}}$			MIT$_{\text{XS}}$			MIT$_{\text{MS}}$			MIT$_{\text{VS}}$			MIT$_{\text{XS}}$		
	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑	Steps ↓	Time ↓	F1 ↑
Base	696	99.2s	0.915	1046	136.0s	0.929	1987	224.6s	0.955	450	124.6s	0.914	700	178.4s	0.927	1800	394.8s	0.956
DU	473	87.8s	0.905	510	94.6s	0.91	480	99.2s	0.893	240	87.2s	0.901	270	93.2s	0.899	240	98.0s	0.872
DWU	419	78.2s	0.887	493	91.8s	0.902	294	62.2s	0.841	233	87.2s	0.874	241	85.8s	0.817	238	96.2s	0.753
DL	514	94.8s	0.912	660	113.6s	0.925	657	121.2s	0.946	270	97.2s	0.907	330	113.2s	0.915	360	123.6s	0.943
BU	580	93.8s	0.914	854	124.2s	0.928	1082	154.0s	0.958	360	113.8s	0.915	480	142.2s	0.926	525	151.2s	0.954
BWU	657	108.2s	0.917	756	112.2s	0.928	918	133.4s	0.955	360	114.2s	0.914	480	144.0s	0.927	532	153.6s	0.955

Table 9: Absolute results for the comparison of the deduplicators on the public datasets presented in table 4.