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
dvdb@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.

∗Equal contribution.

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

![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
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 \(M(\theta)\) is typically trained over a given dataset \(D\) with SGD (or one of its variants), estimating the gradient of a loss function with respect to the model weights \(\theta\), by iterating over the dataset in batches of fixed size. If \(D\) 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,
i.e. at different stages of the model parameters evolution. While (i) explains why we should de-duplicate 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 \( B^\text{virtual} \) of batch \( b \) containing \( B \) unique objects \( o_i \) as

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

Let \( N^\text{dup} \) 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 \( N^\text{dup} \), we have

\[
\mathbb{E}\{B^\text{virtual}\} = B + \mathbb{E}\{N^\text{dup}\}.
\]

The expected relative increase in batch size is then \( B^\text{inc} = \mathbb{E}\{B^\text{virtual}\} / B \). 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 \( \mathbb{E}\{B^\text{virtual}\} \). By definition, we have

\[
\mathbb{E}\{M'\} = \left[ \frac{N}{\mathbb{E}\{B^\text{virtual}\}} \right] = \left[ \frac{N}{B/B^\text{inc}} \right] = \left[ \frac{M}{B^\text{inc}} \right].
\]

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

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

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

\[
= \mathcal{O}\left( \frac{1}{B^\text{inc}} \frac{Nd}{B} \right)
\]

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

\[
\text{frequency}(o_i, b) = \frac{\text{occurrences}(o_i, b)}{B^\text{virtual}}
\]

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 \( \mathbb{E}\{B^\text{virtual}\} \) 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

\[
\mathcal{D} = \{ x_i \mid x_i \in \mathcal{C}, i = 1, \ldots, N \}
\]
of $N$ samples, some of which may be duplicates, taken from a collection $C = \{c_1, \ldots, c_C\}$ of $C$ distinct objects. Let $k_1, \ldots, k_C$ denote the number of occurrences in $D$, such that $k_1 + \cdots + k_C = N$. Then, we can show that:

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

(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 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$_{MS}$, Internal$_{VS}$, Internal$_{XS}$, 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$_{MS}$, MIT$_{VS}$, MIT$_{XS}$, 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 $\alpha$ is set to 3 for MIT$_{MS}$ and MIT$_{VS}$, while it is set to 5 for MIT$_{XS}$. 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-
word embeddings from a pre-trained DistilBERT 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.

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-

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>Redundancy</th>
<th># Named Entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>MITRestaurant</td>
<td>9180</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MITMS</td>
<td>18360</td>
<td>0.5</td>
<td>17</td>
</tr>
<tr>
<td>MITVS</td>
<td>30600</td>
<td>0.7</td>
<td>17</td>
</tr>
<tr>
<td>MITXS</td>
<td>91801</td>
<td>0.9</td>
<td>17</td>
</tr>
</tbody>
</table>

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

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.

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-
A without deduplication, among the tested deduplicators. 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.

Table 3: Comparison of the deduplicators on the internal datasets. Highlighting in bold best results column-wise. Absolute results are available in table 9. 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.

Table 4: Comparison of the deduplicators on the public datasets. Following the same data presentation conventions as in table 3.

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.
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


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


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


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


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,


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 ← []
Ib ← [] → IDs seen in batch
sample_posb ← [] → sample pos in batch
occCb ← [] → occurrences in batch
C ← B → capacity of current batch

for all sample s in data do
    s_pos ← position of the sample
    s_id ← unique id of the sample
    if s_id ∉ Ib then
        schedule += s_pos
        Ib += s_id
        sample_posb[s_id] ← len(occCb)
        occCb.append(1)
        C -= 1
    else
        occCb[sample_posb[s_id]] += 1
    end if
    if (C = 0) ∨ (s is the last sample) then
        occurrences.append(occCb)
        C ← B
        occCb ← []
        sample_posb ← []
        Ib ← []
    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 $z \in \mathbb{R}^{N \times C \times L}$, 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

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

(10)

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.

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

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) giving the distribution of duplicates in the dataset, reported in section 4.

Problem Formalisation Consider a dataset $D = \{x_1, \ldots, x_N\}$ with $N$ objects, some of which might be repeated more than once. As described in section 3, we fill a batch $b$ by sampling $B_{\text{virtual}}$ objects $O = \{x_{p_1}, \ldots, x_{p_{B_{\text{virtual}}}}\}$ so that $O$ contains $B$ distinct elements. We want to compute the expected value $E\{B_{\text{virtual}}\}$. 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 $n = 1, \ldots, N$, up to the one that yields $u = B$ unique samples.

Estimate Derivation Consider now a dataset $D$ of size $N$ instead as a collection $C = \{o_1, \ldots, o_C\}$ of $C$ distinct objects each associated with its number of occurrences $k_1, \ldots, k_C$ such that $k_1 + \cdots + k_C = N$. Consider a random sample without replacement $X_1, \ldots, X_n$ over $D$, with $n < N$ representing the batch size. Now introduce $C$ counting...
variables $Z_1, \ldots, Z_C$, such that $Z_i$ counts occurrences of $o_i$ in the sample. Formally:

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

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

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

and expected value

$$\mathbb{E}(Z_i) = n \frac{k_i}{N}.$$  \hspace{1cm} (13)

We want to compute the expected number of duplicates in the sample $d = \mathbb{E}(N_{\text{dup}})$. 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:

$$N_{\text{dup}} = \sum_{i=1}^{C} \max(Z_i - 1, 0).$$  \hspace{1cm} (14)

By linearity of expectations, it holds that:

$$\mathbb{E}(N_{\text{dup}}) = d = \sum_{i=1}^{C} \mathbb{E}(\max(Z_i - 1, 0))$$  \hspace{1cm} (15)

Let $d_i = \mathbb{E}(\max(Z_i - 1, 0))$, 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

$$d_i = \sum_{j=2}^{k_i} \max(j-1,0)p_{Z_i}(j) = \mathbb{E}(Z_i) - p_{Z_i}(1) \cdot [1 - p_{Z_i}(0) - p_{Z_i}(1)]$$  \hspace{1cm} (16)

and substituting from eq. (13) we get

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

$$p_{Z_i}(0) = \frac{(N-k_i)}{\binom{n}{N}}$$  \hspace{1cm} (18)

and putting it all together, we get:

$$d = \sum_{i=1}^{C} \left( n \frac{k_i}{N} - 1 + \frac{(N-k_i)}{\binom{n}{N}} \right).$$  \hspace{1cm} (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 $k_1, \ldots, k_C$, batch size $B$

**Output:** estimated boost $\hat{N}$

$\text{while } r > l \text{ do}$

$r \leftarrow N$

$\text{while } r > l \text{ do}$

$n = \left\lfloor \frac{(r-l)}{2} \right\rfloor$

$\text{Compute } d \text{ with eq. (19)}$

if $n - d = B$ then

$r \leftarrow \frac{n}{2}$ \hspace{1cm} \triangleright Increase \text{ in batch size}$

return $(1 - \frac{1}{r})$

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 different 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 unweighted 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.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Rate</td>
<td>1e-3</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Max epochs</td>
<td>30</td>
</tr>
<tr>
<td>Embedding size</td>
<td>768</td>
</tr>
<tr>
<td>Hidden size</td>
<td>256</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.5</td>
</tr>
<tr>
<td>Activation</td>
<td>ReLU</td>
</tr>
<tr>
<td>Validation split</td>
<td>0.1</td>
</tr>
<tr>
<td>Early stopping metric</td>
<td>Validation loss</td>
</tr>
<tr>
<td>Early stopping delta</td>
<td>1e-4</td>
</tr>
<tr>
<td>Early stopping patience</td>
<td>3</td>
</tr>
<tr>
<td>Clipping gradient norm</td>
<td>10</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>deduplicator</th>
<th>BS = 512</th>
<th>BS = 1024</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Steps</td>
<td>Time</td>
</tr>
<tr>
<td>Base</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DU</td>
<td>45.0%</td>
<td>-44.0%</td>
</tr>
<tr>
<td>DWU</td>
<td>81.0%</td>
<td>-58.0%</td>
</tr>
<tr>
<td>DL</td>
<td>36.0%</td>
<td>-34.0%</td>
</tr>
<tr>
<td>BU</td>
<td>3.0%</td>
<td>+4.0%</td>
</tr>
<tr>
<td>BWU</td>
<td>3.0%</td>
<td>+4.0%</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>deduplicator</th>
<th>BS = 512</th>
<th>BS = 1024</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Steps</td>
<td>Time</td>
</tr>
<tr>
<td>Base</td>
<td>504</td>
<td>16s</td>
</tr>
<tr>
<td>DU</td>
<td>387</td>
<td>12s</td>
</tr>
<tr>
<td>DWU</td>
<td>297</td>
<td>10s</td>
</tr>
<tr>
<td>DL</td>
<td>392</td>
<td>12s</td>
</tr>
<tr>
<td>BU</td>
<td>451</td>
<td>15s</td>
</tr>
<tr>
<td>BWU</td>
<td>398</td>
<td>14s</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>deduplicator</th>
<th>BS = 512</th>
<th>BS = 1024</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Steps</td>
<td>Time</td>
</tr>
<tr>
<td>Base</td>
<td>696</td>
<td>99.2s</td>
</tr>
<tr>
<td>DU</td>
<td>473</td>
<td>87.8s</td>
</tr>
<tr>
<td>DWU</td>
<td>419</td>
<td>78.2s</td>
</tr>
<tr>
<td>DL</td>
<td>514</td>
<td>94.8s</td>
</tr>
<tr>
<td>BU</td>
<td>580</td>
<td>93.8s</td>
</tr>
<tr>
<td>BWU</td>
<td>657</td>
<td>108.2s</td>
</tr>
</tbody>
</table>

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