Large-Scale Differentially Private BERT

Anonymous ACL submission

Abstract

In this work, we study the large-scale pretraining of BERT-Large (Devlin et al., 2019) with differentially private SGD (DP-SGD). We show that combined with a careful implementation, scaling up the batch size to millions (i.e., mega-batches) improves the utility of the DP-SGD step for BERT; we also enhance the training efficiency by using an increasing batch size schedule. Our implementation builds on the recent work of Subramani et al. (2020), who demonstrated that the overhead of a DP-SGD step is minimized with effective use of JAX (Bradbury et al., 2018; Frostig et al., 2018) primitives in conjunction with the XLA compiler (XLA team and collaborators, 2017). Our implementation achieves a masked language model accuracy of 60.5% at a batch size of 2M, for $\epsilon = 5$, which is a reasonable privacy setting. To put this number in perspective, non-private BERT models achieve an accuracy of ∼70%.

1 Introduction

The widespread deployment of machine learning in recent years has raised serious concerns about the privacy of users whose data is used during the training process (see, e.g., (Kearns and Roth, 2019)). These concerns are exacerbated by the well-known memorization behavior exhibited by deep neural networks (Carlini et al., 2019) and in particular large language models (Carlini et al., 2021). To mitigate these concerns, the framework and properties of differential privacy (DP) (Dwork et al., 2006b,a) provide a compelling approach for rigorously controlling and preventing the leakage of sensitive user information present in the training dataset. Loosely speaking, DP guarantees that the output distribution of a (randomized) algorithm does not noticeably change if a single training example is added or removed; this change is parameterized by two numbers (ϵ, δ)—the smaller these parameter values, the more private the algorithm. We refer the reader to Section 2 for the formal definition of DP, and to Dwork and Roth (2014) for a thorough overview.

Motivated by these concerns, there has been a significant body of work on training private ML models. Notably, Abadi et al. (2016) presented a generic recipe for training ML models with DP. While their DP-SGD framework is quite robust as it applies to arbitrary neural network architectures, it faces two challenges that have significantly limited its practical deployment:

(i) the gap between its accuracy and that of the best non-private methods can be significant, and

(ii) the training time overhead (due to per-example gradient clipping) is considerable.

Note that on simple tasks such as MNIST digit classification, the accuracy of DP models is not too far from that of non-private models. However, for more complex tasks such as CIFAR-10, the accuracy gap is very large, ∼25% for reasonable privacy parameter settings. For even more complicated tasks such as CIFAR-100, the inefficiency of DP-SGD has for several years precluded the training of DP neural networks. These limitations have made the DP training of a complex language model such as the Bidirectional Encoder Representation (BERT) (Devlin et al., 2019) a daunting task.

Very recently Hoory et al. (2021) tackled the challenge of fine-tuning BERT with DP and relied on non-private pretraining on the combined Wikipedia and BooksCorpus (Zhu et al., 2015) datasets. In this work, we take a step further and consider the task of pretraining a BERT-Large model with DP. Obtaining a pre-trained model with DP guarantee allows us to employ it for multiple downstream tasks without violating the privacy of the data used in pre-training[1]. Pretraining BERT-Large, however, is a computationally intensive task even without privacy; with privacy, using DP-SGD to pretrain BERT-Large poses significantly more

[1]This is a byproduct of the post-processing property of DP.

computational challenges. Our work shows how to overcome these barriers. We present an implementation of a variant of DP-SGD that, surprisingly, can train (relatively) quickly on state-of-the-art hardware and achieve good accuracy.

1.1 Contributions

In this work, we establish a high accuracy baseline for DP BERT-Large pretraining. Our primary contributions are:

(i) *Negative interaction of naive DP-SGD with scale-invariant layers:* We discuss the importance of a large weight decay parameter (in Adam optimizer) and its interactions with layers that are scale-invariant individually and jointly. This insight allows us to tune hyper-parameters effectively, thereby achieving higher accuracies.

(ii) *Mega-batches improve accuracy for DP BERT-Large:* We demonstrate that scaling up the batch sizes to ∼2M improves the utility of every step of DP-SGD empirically. This batch size is 32× larger than previously used for non-private training of BERT (Nado et al., 2021). We achieve a masked language modeling accuracy of 60.5%.

Complementing this, we also provide a theoretical justification as to why large batch sizes are advantageous in DP-SGD.

(iii) *Increasing batch size schedule improves training efficiency:* We show that an increasing batch size schedule can improve the efficiency of the training procedure while matching the accuracy of a fixed batch size schedule. We motivate our approach using a notion of gradient-SNR (signal-to-noise ratio). Our proof-of-concept experiments show up to 14% reduction in the total number of examples visited to achieve the same accuracy.

1.2 Related work

In previous work McMahan et al. (2018) trained recurrent language models with DP. Other works also considered adaptive clipping in the context of DP-SGD (Andrew et al., 2019; Pichapati et al., 2019) as well as adaptive learning rates (Koskela and Honkela, 2020). In the non-private literature, increasing batch sizes have been considered, e.g., in Smith et al. (2018). Finally, a significant speedup of DP-SGD was shown to be possible in Subramani et al. (2020) via large leaps in software (Bradbury et al., 2018; Frostig et al., 2018; XLA team and collaborators, 2017) for machine learning; this serves as our foundation for scaling pretraining for BERT.

Memorization properties of the DP trained model have been investigated in (Carlini et al., 2021; Brown et al., 2021); Feldman (2020) studies the generalization properties of DP models.

Organization. We start with some background in Section 2. The details of the algorithm are described in Section 3. Our experimental setup and results are presented in Section 4 and Section 5. We conclude with some future directions in Section 6.

2 Preliminaries

Similar to most previous works on DP ML, we say that two datasets X and X' are *neighboring*, if X' results from adding or removing a single *training example* from X.

Definition 1 (Differential Privacy (Dwork et al., 2006b,a))**.** *For any real numbers $\epsilon \geq 0$ and $\delta \in [0,1]$, a randomized algorithm A is (ϵ, δ)-*differentially private (DP) *if for every pair X, X' of neighboring datasets and every subset S of outputs of A, it is the case that*

$$\Pr[A(X) \in S] \leq e^{\epsilon} \cdot \Pr[A(X') \in S] + \delta,$$

where the probabilities are over the randomness in the algorithm A. When $\delta = 0$, we simply use ϵ-DP.

DP has seen significant interest in the literature due to its nice mathematical properties such as composition, post-processing, and group privacy (see, e.g., the book of Dwork and Roth (2014)).

3 Algorithm

At a high-level, we use the DP-SGD algorithm (Abadi et al., 2016) with the Adam optimizer (Kingma and Ba, 2015). Our choice of Adam follows the work of Nado et al. (2021), who showed that tuning Adam works well up to large batch sizes of 65K. As our goal is to establish a baseline for BERT pretraining with DP, we leave the investigation of higher-order methods (Anil et al., 2020; Amid et al., 2021) to future work.

At each step of training, we randomly select a prespecified number of examples, compute and clip their gradients and add appropriate noise to the average gradient to ensure privacy. To compute the noise multiplier, we use the privacy loss distribution (PLD) method (e.g., (Gopi et al., 2021)) implemented in Google DP Accounting Library[2].

[2]https://github.com/google/differential-privacy/tree/main/python/dp_accounting.

Algorithm 1 DP-SGD using Adam optimizer with weight decay

```
Require: Examples $x_1, \dots, x_n$, loss $\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\theta; x_i)$.
Ensure: number $T$ of steps, batch sizes $q_1, \dots, q_T$, clipping norm $C$, noise multiplier $\sigma$, momentum
    parameter $\beta_1$, second-moment averaging parameter $\beta_2$, weight decay $\lambda$, learning rate $\eta_t$.
  for $t = 1, \dots, T$ do
      $B_t \leftarrow$ random $q_t$ training examples.
      $g_t \leftarrow \frac{1}{|B_t|}\left(\mathcal{N}(0, \sigma^2 C^2 I) + \sum_{x_j \in B_t} \mathrm{clip}(\nabla\mathcal{L}(\theta; x_j), C)\right)$
      $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
      $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
      $\hat{m}_t \leftarrow m_t/(1-\beta_1^t)$
      $\hat{v}_t \leftarrow v_t/(1-\beta_2^t)$
      $\theta_t \leftarrow \theta_{t-1} - \eta_t\left(\hat{m}_t/(\sqrt{\hat{v}_t} + \xi) + \lambda \cdot \theta_{t-1}\right)$        ▷ $\xi$ is set to $10^{-11}$
  return $\theta_T$
```

This gives tighter privacy parameters ϵ, δ compared to the Rényi DP method in (Abadi et al., 2016; Mironov, 2017). These noisy average gradients are then used to update the parameters via the Adam update (Kingma and Ba, 2015) rule with weight decay (Loshchilov and Hutter, 2017). See Algorithm 1 for a self-contained detailed description.

For the increasing batch size schedule, we modify the DP accounting procedure to handle multiple batch sizes, in a straightforward manner.

4 Experimental Setup and Tuning

We present results on pretraining a BERT architecture (Devlin et al., 2019), focusing on its larger variant, aka "BERT-Large", which is a transformer model (Vaswani et al., 2017) containing 24 transformer blocks with 1024 hidden dimensions and 16 self attention heads. It has 340M parameters (1.297 GiB). The training setup is a reimplementation of the official BERT codebase[3] in the JAX framework (Bradbury et al., 2018; Frostig et al., 2018) with the FLAX library (Heek et al., 2020). Our choice of JAX was motivated by the recent results of Subramani et al. (2020), where they demonstrate that JAX features such as Just-In-Time (JIT) compilation and compiler fusion result in low runtime overheads for the DP-SGD step and match or commonly exceeds the performance of various other frameworks. All experiments were carried out on Google TPUs (Jouppi et al., 2017), using the TPUv3-1024 configuration.

[3] https://github.com/google-research/bert

4.1 Pretraining dataset

The pretraining dataset is the combined Wikipedia and Books corpus (Zhu et al., 2015) datasets with 2.5B and 800M words, respectively. It consists of about 346M examples, each containing two sentences (389M unique sentences). To have a finite vocabulary and address out-of-vocabulary words gracefully, the words in the sentences are segmented into word-pieces (Sennrich et al., 2016). There are 32K tokens in the vocabulary. Each pair of sentences has 128 tokens. 20 tokens from each example were replaced with masked tokens (15% of the tokens). The objective of the model is to predict the masked tokens and which of the two sentences precedes the other. A typical pretraining model achieves 70% in masked-language model (MLM) accuracy. After pretraining, the model parameters are used for fine-tuning on small amounts of data to solve specific natural language tasks (Wang et al., 2019).

4.2 On Advantage of Large Batch Sizes

Our training employs fairly large batch sizes. Here we provide a justification for these large batch sizes. In particular, we argue below that once we fix the number of steps T, there is a minimum batch size below which should not be used for DP-SGD.

For simplicity of exposition, we assume that the batch sizes are fixed to q in each step. To formalize the aforementioned intuition, we will define the notion of *normalized noise multiplier* $\tilde{\sigma}$ which is defined as $\frac{\sigma}{q}$—this is the noise multiplier added in each DP-SGD step after the average of gradients.

Again, for simplicity, we assume that we are using the advanced composition theorem (Dwork et al., 2010) and the generic amplification-by-subsampling theorem (Balle et al., 2018) for privacy calculations. In actual training, we use the tighter PLD accounting; nonetheless, as we empirically show below, the asymptotic behaviors remain the same as in our simplified setting. When applying the advanced composition theorem (Dwork et al., 2010), we get that it suffices to make each step of DP-SGD (ϵ_0, δ_0)-DP in order for the entire algorithm be (ϵ, δ)-DP. Here ϵ_0, δ_0 depend only on ϵ, δ, T and not on the batch size q. We assume in the calculation below that $\epsilon_0 < 1$ and $\delta_0 < 1/n$. These are the standard parameters of interest.

Now, in order for each step be (ϵ_0, δ_0)-DP, we may apply the amplification-by-subsampling theorem (Balle et al., 2018) with sampling probability $p = q/n$. This requires the (non-subsampled) Gaussian mechanism to be (ϵ_1, δ_1)-DP where

$$\epsilon_1 = \log(1 + (e^{\epsilon_0} - 1)/p), \qquad \delta_1 = \delta_0/p.$$

This allows us to set the noise multiplier σ to be[4]

$$\sigma = \Theta\left(\frac{\sqrt{\log(1/\delta_0)}}{\log(1 + (e^{\epsilon_0} - 1)/p)}\right),$$

giving a normalized noise multiplier of

$$\tilde{\sigma} = \frac{\sigma}{q} = \Theta\left(\frac{1}{q} \cdot \frac{\sqrt{\log(1/\delta_0)}}{\log(1 + (e^{\epsilon_0} - 1)/p)}\right). \quad (1)$$

Now, consider two cases based on the batch size.

Case I: $q \geq \epsilon_0 n$. In this large batch size case, $p > \epsilon_0$. This implies $\log(1 + (e^{\epsilon_0} - 1)/p) = \Theta(\epsilon_0/p)$. Plugging this into (1), we get $\tilde{\sigma} = \Theta\left(\frac{\sqrt{\log(1/\delta_0)}}{\epsilon_0 n}\right)$. In other words, the normalized noise multiplier remains roughly *constant* when the batch size is sufficiently large.

Case II: $q < \epsilon_0 n$. In the small batch size case, $p < \epsilon_0$, which implies $\log(1 + (e^{\epsilon_0} - 1)/p) = \Theta(\log(\epsilon_0/p))$. Plugging this into (1), we have $\tilde{\sigma} = \Theta\left(\frac{\sqrt{\log(1/\delta_0)}}{q \log(n\epsilon_0/q)}\right)$. This means that the normalized noise multiplier keeps *increasing* as the batch size decreases. In fact, the decrease rate is almost inverse linear in q here, i.e., the unnormalized noise multiplier remains almost constant even when the batch size decreases.

To summarize, the normalized noise multiplier is constant when the batch size is sufficiently large, but at a critical point $\Theta(\epsilon_0 n)$, the normalized noise multiplier starts increasing as the batch size decreases. This effect can also be seen empirically for noise multiplier computed from the PLD accounting method (Figure 1), and is indeed one of the reasons large batch sizes are more effective for DP-SGD in large datasets.

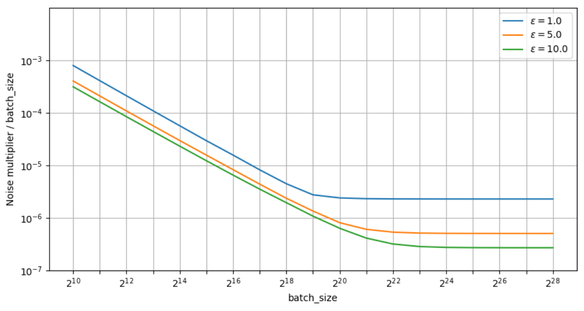


Figure 1: Normalized noise multiplier with varying batch sizes. We see that there is a critical point after which the value becomes roughly a constant.

4.3 Hyper-parameter tuning

We tune hyper-parameters for Adam at a batch size of 32K and transfer the tuned hyper-parameters to all other batch sizes. For tuning the hyper-parameters, we tune with grid search, and use a total of 288 trials. Note that we do not train all 288 trials to completion as many of the hyper-parameter combination either fail (model blow-ups) or produce lower accuracies (<15%) even after 4000 steps. Hyper-parameter tuning spaces are listed in Table 1. Note that we can account for the privacy cost of hyper-parameter tuning using known tools from the literature. Specifically, we follow Appendix G of Abadi et al. (2016) and apply Theorem 10.2 of Gupta et al. (2010).

Theorem 1 (Theorem 10.2 of Gupta et al. (2010)). Let $M : D \to R$ be an ϵ-DP mechanism such that for a query function $q : D \times R \to \mathbb{R}_{\geq 0}$ of sensitivity 1 with respect to D (meaning that for any fixed $r \in R$, the function $q(\cdot, r)$ has sensitivity 1), and a parameter Q, it holds that $\Pr[q(A, M(A)) \geq Q] \geq p$ for some $p \in (0, 1)$. Then, for any $\kappa > 0$ and $\epsilon' \in (0, \frac{1}{2})$, there is a mechanism M', which satisfies the following:

(i) $\Pr[q(A, M(A)) \geq Q - \frac{4}{\epsilon'} \log(\frac{1}{\epsilon' \kappa p})] \geq 1 - \kappa$.

(ii) M' makes $O((\frac{1}{\epsilon' \kappa p})^2 \ln(\frac{1}{\epsilon' \kappa p}))$ calls to M.

M' satisfies $(\epsilon + 8\epsilon')$-DP.

Moreover, in the case of grid search, the new DP parameter can be improved to $\max(\epsilon, 8\epsilon')$ (Abadi et al., 2016). In our case, we set $\kappa = 0.1$ and $p =$

[4]See e.g. (Dwork and Roth, 2014, Appendix A).

$1/K$, where K is the number of hyper-parameters on the grid. Then, the above theorem implies that our training methods can be privatized with a loss of accuracy of at most $\frac{4}{10000\epsilon'}\ln(\frac{1}{\epsilon'\kappa p})$ with probability at most 0.9, where we have used the fact that the number of validation points is equal to 10000. For $\epsilon = 5$, we could set ϵ' as large as $\epsilon/8 = 0.625$. For a grid search over K parameters, the loss in accuracy (with probability at least 0.9) will be upper-bounded by: $\frac{4}{10000\epsilon'}\ln(\frac{1}{\epsilon'\kappa p}) = 0.00064{\cdot}\ln(16{\cdot}K)$. For $K = 289$ choices of hyper-parameters, the drop in accuracy would be 0.0054, or 0.54%.

A linear learning rate warmup followed by a quadratic decay schedule is used for training. The tuning objective was to maximize the MLM accuracy over 10k examples following Nado et al. (2021). Non-private training of BERT-Large reaches 70% accuracy at 32K batch size in 14,063 steps. For all experiments, we train for 20K steps, with a 7.5K step warmup. A key insight from tuning is that a very large weight decay λ needs to be set to achieve high accuracy. We will discuss this next section

4.4 Scale-invariance and large weight decay

The primary observation that led to larger search space for weight decay is that BERT-Large has several scale-invariant layers. A *scale-invariant* layer is one in which increasing the norm of the weights in the layer by a positive scalar has no effect on the function output. However, the norm of the gradient for the weights of the layer shrinks as it is inversely proportional to the norm of the weights. This has been noted in the literature (Hoffer et al., 2019; Cho and Lee, 2017; Davody et al., 2020; Heo et al., 2021), and Heo et al. (2021) propose a modification to Adam to handle these layers for non-private training, and Davody et al. (2020) introduce batch normalization layers to make networks scale-invariant in order to improve accuracy, but do not consider the trainability issue of standard DP-SGD addressed here.

To make this concrete, consider a fully connected layer with parameters $W \in \mathbb{R}^{m\times n}$, where $Wx = s$. The gradient for the layer $G_t \in \mathbb{R}^{m\times n}$ can be written via chain rule as $\nabla_s \ell(s_t, y_t)x^\intercal$. Under layer normalization, the preactivation vector s is normalized to have zero mean and unit variance:

$$f(s) = \left(\frac{s - \mathbb{E}[s]}{\sqrt{\text{Var}[s]} + \xi}\right).$$

A key observation is that layer normalization makes the layer's output independent of the scale of W, i.e., multiplying W by non-zero $\alpha \in \mathbb{R}$ has no effect on the function output. With DP training, the Gaussian noise added to the gradient tends to increase the Frobenius norm $||W||_F$ of the weights over training which unintentionally shrinks the gradients, making training ineffective! Thus, to stabilize training, we set the weight decay parameter to be much larger than non-private training. This interaction is quite crucial for practitioners of DP to be aware of when applying DP to any neural network, as normalization layers such as layer-norm (Ba et al., 2016), batch-norm (Ioffe and Szegedy, 2015), and weight-norm (Salimans and Kingma, 2016) are common.

Notice that the difficulty in employing a more straightforward solution for the problem through gradient projection, i.e., projecting the component of the noise vector orthogonal to the weight vector, is that we need to infer the scale invariance property of the layers apriori, which is difficult on a broad range of models. Moreover, for BERT-Large, the three types of embedding layers—wordpiece, positional, and token type—are scale-invariant together but not individually due to layer normalization being applied after aggregating these embeddings in the input layer. Thus, the tuning of weight decay is preferable to the heuristics from Heo et al. (2021). Finally, handling the addition of standard DP noise to the scale-invariant layers efficiently by identifying them and including conjoint ones is an exciting avenue for further research.

5 Experimental Results

In this section, we study the effect of the privacy parameter ϵ and fix a particular value of ϵ to aim for the highest attainable MLM accuracy, by changing the batch size.

5.1 Varying privacy parameter ϵ

In Figure 2, we present the MLM accuracy results for varying the ϵ parameter. We set the batch size to 65,536 and trained for 20K steps with a 7500 steps warmup. Hyper-parameters were transferred from the tuning experiments at 32,768 batch size from Table 1. As expected, at $\epsilon = 1.08$, the accuracy drops to 33.2% while achieving up to 42.85% at $\epsilon = 10.6$. We identify a sweet spot of $\epsilon = 5$ and use it for further experimentation; we also use $\delta = 2.89 \times 10^{-9}$, the reciprocal of the number of

Hyper-parameter	Grid	Best trial
η (learning rate)	$\{5\times10^{-4}, 1\times10^{-4}, 5\times10^{-3}, 1\times10^{-3}\}$	5×10^{-4}
$1-\beta_1$ (momentum parameter)	$\{0.25, 0.1, 0.05\}$	0.25
$1-\beta_2$ (second-moment parameter)	$\{0.25, 0.1, 0.05\}$	0.1
λ (weight decay)	$\{10^{-1}, 1\}$	1.0
C (clipping norm)	$\{1\times10^{-3}, 5\times10^{-3}, 1\times10^{-2}, 5\times10^{-2}\}$	5×10^{-3}

Table 1: Hyper-parameter tuning search space, where search is done with grid-search. Learning rate schedule is a linear warmup followed by a quadratic decay. DP parameters ϵ is set to 5 and δ to 2.89×10^{-9}.

examples, throughout.

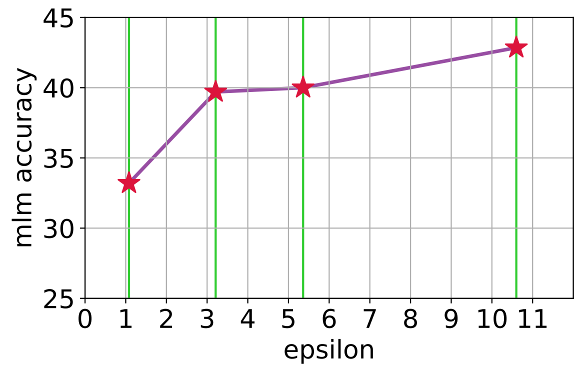


Figure 2: MLM accuracy tradeoff with varying ϵ. We use $\epsilon = 5$ for the rest of the paper. Batch size is 65,536, trained for 20K steps with 7500 steps of warmup.

5.2 Varying batch size with fixed step budget

We now study the effect of batch size on MLM accuracy. In non-private training, typically batch size scaling is studied under a fixed epoch budget (Shallue et al., 2019; Nado et al., 2021) rather than a step budget. In contrast, we carry out this study at a fixed step budget for two reasons:

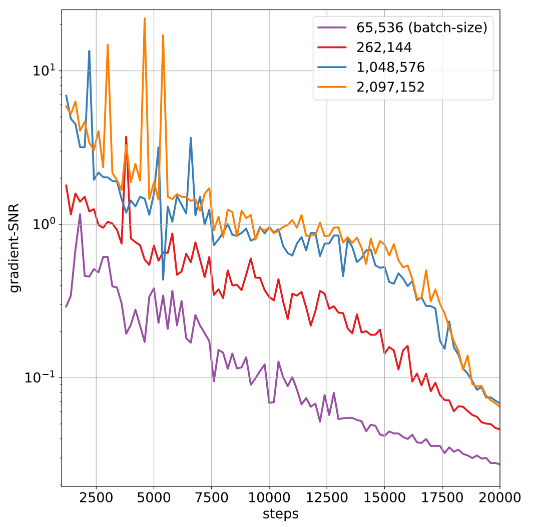
(i) our primary motivation is to establish a high accuracy private baseline, and

(ii) fixed epoch budget study requires re-tuning hyper-parameters efficiently for very large batch sizes, an open challenge for private training.

Improving optimization efficiency of the training procedure is a natural next step, and along these lines, we propose an increasing batch size schedule that improves the efficiency and is described in the later sections. (See Section 6 for other possibilities for efficiency improvement.) Notice that for all our experiments, we transfer the hyper-parameters from the 32K hyper-parameter tuning and naively increase the batch size while fixing ϵ to 5.

5.2.1 Effect of batch size on DP gradients

As described in Section 3, DP training relies on clipping individual gradients, then aggregating them, followed by the addition of a noise vector. We measure the ratio between the norms of the aggregated gradient of the network and the noise vector. We call this quantity *gradient signal-to-noise* ratio (gradient-SNR) and measure it over the training run; the results are presented in Figure 3a and Figure 4a. We observe that using a larger batch size yields favorable gradient-SNR through training and overall higher accuracy. Gradient-SNR can be seen to shrink as training progresses, which motivates the batch size schedule that we discuss next.

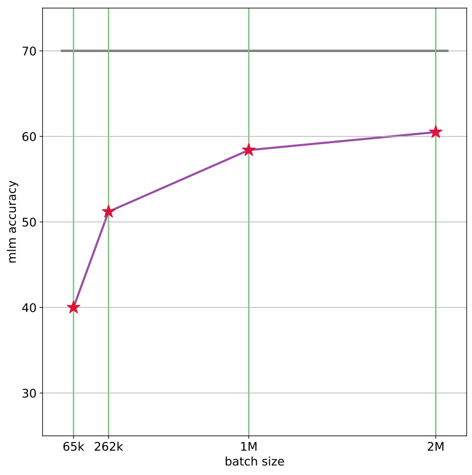


(a) Gradient-SNR

Figure 3: Gradient-SNR at several batch sizes at a fixed step budget of 20K steps using hyper-parameters from Table 1. Larger batch sizes improve overall accuracy. Target solution quality is 70% and is achieved by BERT-Large at modest batch sizes of 32K in 14,063 steps.

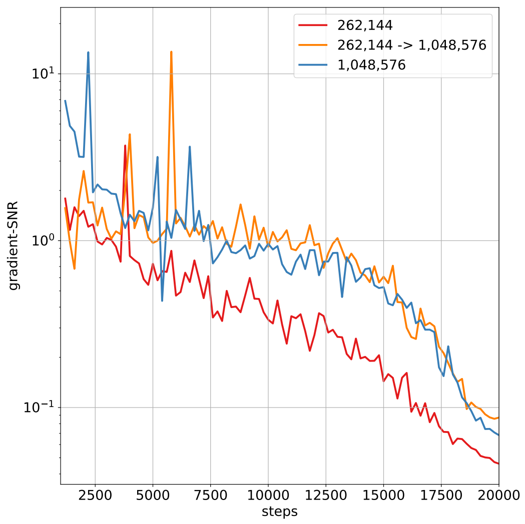
5.2.2 Batch size schedule improves efficiency

To improve the efficiency of training, we propose an increasing batch size schedule. As seen in Figure 3a, the gradient norm decreases over time, and



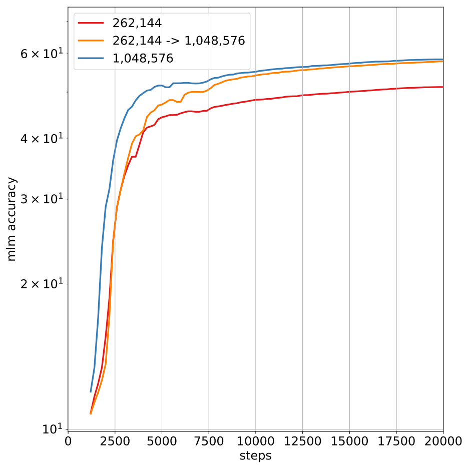
(a) MLM accuracy

Figure 4: MLM accuracy at several batch sizes at a fixed step budget of 20K steps using hyper-parameters from Table 1. Larger batch sizes improve overall accuracy. Target solution quality is 70%, achieved by BERT-Large at modest batch sizes of 32K in 14,063 steps.



(a) Gradient-SNR

Figure 5: Gradient-SNR for fixed batch sizes 262K, and 1M, and increasing schedule starting at 262K to 1M over 7500 steps.



(a) MLM accuracy

Figure 6: MLM accuracy for fixed batch sizes 262K, and 1M, and increasing schedule starting at 262K to 1M over 7500 steps.

the noise starts dominating, which leads to slower convergence of DP training. This is primarily in the DP framework, the noise added does not grow proportional to the batch size under fixed steps budget. Based on this insight, we devise an increasing batch size schedule from 262,144 (262K) to 1,048,576 (1M) over 7.5K steps. Every one-quarter of the 7.5K steps, we increase the batch size by ∼196K examples, reaching 1M at 7.5 steps. Overall we observe improved efficiency in training as seen in Figure 5a and match the accuracy of fixed batch size training in Figure 6a. This technique reduces examples seen in training by 14%. We note that increasing batch sizes have been proposed in the literature, e.g., in Smith et al. (2018), where the technique is used as a substitute for decreasing the learning rate. In contrast, our proposal is motivated by the specific DP-SGD setting, i.e., the fact that the norm of the gradients tends to decrease as training progresses, so naturally, increasing the batch size allows us to improve gradient-SNR.

5.3 On scaling up to mega batch sizes

The primary bottleneck with data-parallel training at large batch sizes (>65K) for BERT-Large is memory. Recall that You et al. (2019) resort to using smaller batch sizes of (32K) when training with longer sequence lengths (512) when using equivalent amount of hardware resources used in this work. Motivated by the fact it is clear that increasing batch sizes improves gradient-SNR and thereby improves the overall accuracy of the model, we handle mega-batch sizes by simply using gradient accumulation across examples to form large batches. We rely heavily on JAX (Bradbury et al., 2018; Frostig et al., 2018) primitives, and with them it is straightforward to implement this functionality by accumulating gradients over batches of examples via a `jax.lax.fori_loop` and `jax.vmap` (vectorized map).

Large batch training can be quite beneficial in improving overall efficiency when training with slow interconnects (on typical GPU setup) as they amortize the cost of gradient reduction. We have found that increasing batch size is useful as long as the progress per example (utility) does not shrink. Another avenue is in the training of larger models (>5B parameters) that requires model parallelism. In this case, weights are split across devices, and efficiency is improved via pipeline parallelism where smaller batches are used to pipeline computations (forward, backward, update) across devices, as described in Xu et al. (2021); Lepikhin et al. (2021). Integrating DP-SGD in that setup is the natural next step when training models much larger than BERT-Large. Another interesting question is how accuracy differs across various sizes of BERT models, which we leave for future work. To conclude, we use the gradient accumulation to scale up the batch size to 2,097,152 (2M), which is 32× larger than previously reported in the literature for non-private training, and obtain an MLM accuracy of **60.5%** with DP.

5.4 DP and memorization

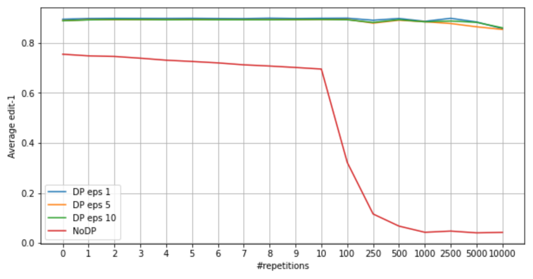
Carlini et al. (2021) showed that large language models do memorize training data, so we investigate whether DP leads to less memorization. Our experimental setup is as follows: we create a set of training examples similar to the original BERT dataset, but from a disjoint public domain corpus. However instead of masking out random tokens, here in each example we mask out a sequence of tokens corresponding to a phrase in the text. Each example was added to the training set with some frequency: 25,000 examples each were added with frequencies 1 through 10, and we also added some examples with higher frequencies. In addition, 25,000 examples were held out as the control set.

We trained the model on this augmented dataset with $\epsilon = 1, 5, 10$, as well as without DP, and then predicted masked tokens in the new examples. We computed various metrics on these predictions. Two of these are shown in Figure 7:

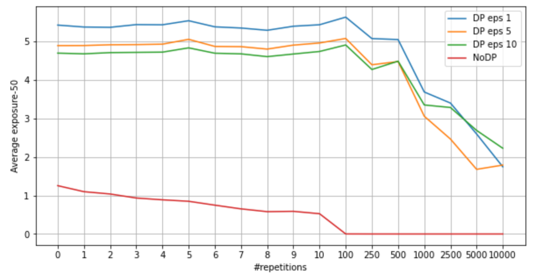
(i) The edit distance between the missing phrase and top prediction.

(ii) The exposure-50 metric of Carlini et al. (2019)—this is the negative log of the estimated rank of the missing phrase among all predictions.

It is clear from the slope of the line graphs that non-private training memorizes significantly more than DP training. However ϵ did not seem to make a significant difference—the predictions for a few repetitions was similar to the holdout set, though of course the models memorized examples that were repeated a lot.



(a) Edit distance



(b) Exposure-50

Figure 7: Metrics for various values of ϵ.

6 Conclusions

We build off the recent advances in software (XLA team and collaborators, 2017; Bradbury et al., 2018; Frostig et al., 2018) and hardware (Jouppi et al., 2017) and establish a baseline for BERT-Large pretraining with DP. We achieve high accuracy for the model by scaling up the batch size to millions of examples (mega-batches) and using additional insights such as improving the trainability of networks under normalization layers and measuring the gradient-SNR metric. We proposed a proof of concept batch size increasing schedule and demonstrate an efficiency improvement. An interesting direction is to improve the efficiency of DP training further by leveraging recent advances such as higher-order methods (Anil et al., 2020; Gupta et al., 2018), local loss optimization (Amid et al., 2021), scaling techniques that exploit increased parallelism: such as online distillation (Anil et al., 2018) and ones that use lower memory (Shazeer and Stern, 2018; Anil et al., 2019), automatic tuning of hyper-parameters meta optimization (Amid et al., 2020), more efficient architectures such as MLP-Mixer, and F-NETs (Tolstikhin et al., 2021; Lee-Thorp et al., 2021) for longer sequences, multi-step training at mega-batch sizes (Choi et al., 2019; Agarwal et al., 2020), loss functions that are robust to label noise (Amid et al., 2019).

7 Ethical considerations & Limitations

This work addresses the accuracy gap typically seen when pretraining a moderately sized language model (BERT-Large) with DP. We find that larger batches, with good training metrics (Gradient-SNR) and architecture-aware tuning insights (the negative interaction between layer norm and DP-SGD) closes this gap. While this work has focused solely on pretraining, there are several limitations and questions on the behavior of pretrained DP models for fine-tuning and around what DP parameters (ϵ) to choose in practice to balance quality and privacy hope the community takes on. The training time of the largest experiment: BERT-Large at 2M batch size on TPUv3-1024 is 72 hours, which is expensive. Pretraining cost is amortized as it is only carried out once compared to several downstream use cases of the model. Moreover, we think several follow-up work has the potential to reduce the cost, for example, through more efficient architectures and optimizers.

References

Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *CCS*, pages 308–318.

Naman Agarwal, Rohan Anil, Tomer Koren, Kunal Talwar, and Cyril Zhang. 2020. Stochastic optimization with laggard data pipelines. In *NeurIPS*, pages 10282–10293.

Ehsan Amid, Rohan Anil, Christopher Fifty, and Manfred K. Warmuth. 2020. Step-size adaptation using exponentiated gradient updates. In *ICML Workshop on Beyond First-Order Methods in ML Systems*.

Ehsan Amid, Rohan Anil, and Manfred K Warmuth. 2021. Locoprop: Enhancing backprop via local loss optimization. *CoRR*, abs/2106.06199.

Ehsan Amid, Manfred K. Warmuth, Rohan Anil, and Tomer Koren. 2019. Robust bi-tempered logistic loss based on Bregman divergences. In *NeurIPS*.

Galen Andrew, Om Thakkar, H Brendan McMahan, and Swaroop Ramaswamy. 2019. Differentially private learning with adaptive clipping. *CoRR*, abs/1905.03871.

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. 2020. Scalable Second Order Optimization for Deep Learning. *CoRR*, abs/2002.09018.

Rohan Anil, Vineet Gupta, Tomer Koren, and Yoram Singer. 2019. Memory efficient adaptive optimization. In *NeurIPS*.

Rohan Anil, Gabriel Pereyra, Alexandre Passos, Róbert Ormándi, George E. Dahl, and Geoffrey E. Hinton. 2018. Large scale distributed neural network training through online distillation. In *ICLR*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *CoRR*, abs/1607.06450.

Borja Balle, Gilles Barthe, and Marco Gaboardi. 2018. Privacy amplification by subsampling: Tight analyses via couplings and divergences. In *NeurIPS 2018*, pages 6280–6290.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: composable transformations of Python+NumPy programs.

Gavin Brown, Mark Bun, Vitaly Feldman, Adam Smith, and Kunal Talwar. 2021. When is memorization of irrelevant training data necessary for high-accuracy learning? In *STOC*, pages 123–132.

Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. 2019. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *USENIX Security*, pages 267–284.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. 2021. Extracting training data from large language models. In *USENIX Security*.

Minhyung Cho and Jaehyung Lee. 2017. Riemannian approach to batch normalization. In *NIPS*.

Dami Choi, Alexandre Passos, Christopher J Shallue, and George E Dahl. 2019. Faster neural network training with data echoing. *CoRR*, abs/1907.05550.

Ali Davody, David Ifeoluwa Adelani, Thomas Kleinbauer, and Dietrich Klakow. 2020. Robust differentially private training of deep neural networks. *CoRR*, abs/2006.10919.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186.

Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006a. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT*, pages 486–503.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006b. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284.

Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407.

Cynthia Dwork, Guy N. Rothblum, and Salil P. Vadhan. 2010. Boosting and differential privacy. In *FOCS*, pages 51–60.

Vitaly Feldman. 2020. Does learning require memorization? A short tale about a long tail. In *STOC*, pages 954–959.

Roy Frostig, Matthew Johnson, and Chris Leary. 2018. Compiling machine learning programs via high-level tracing. *MLSys*.

Sivakanth Gopi, Yin Tat Lee, and Lukas Wutschitz. 2021. Numerical composition of differential privacy. In *NeurIPS*.

Anupam Gupta, Katrina Ligett, Frank McSherry, Aaron Roth, and Kunal Talwar. 2010. Differentially private combinatorial optimization. In *SODA*, pages 1106–1125.

Vineet Gupta, Tomer Koren, and Yoram Singer. 2018. Shampoo: Preconditioned stochastic tensor optimization. In *ICML*, pages 1842–1850.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. 2020. Flax: A neural network library and ecosystem for JAX.

Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. 2021. Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights. In *ICLR*.

Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. 2019. Norm matters: efficient and accurate normalization schemes in deep networks. In *NeurIPS*, pages 2164–2174.

Shlomo Hoory, Amir Feder, Avichai Tendler, Alon Cohen, Sofia Erell, Itay Laish, Hootan Nakhost, Uri Stemmer, Ayelet Benjamini, Avinatan Hassidim, and Yossi Matias. 2021. Learning and evaluating a differentially private pre-trained language model. In *PrivateNLP*, pages 21–29.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456.

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, pages 1–12.

Michael Kearns and Aaron Roth. 2019. *The Ethical Algorithm: The Science of Socially Aware Algorithm Design*. Oxford University Press.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.

Antti Koskela and Antti Honkela. 2020. Learning rate adaptation for differentially private learning. In *AISTATS*, pages 2465–2475.

James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. 2021. Fnet: Mixing tokens with Fourier transforms. *CoRR*, abs/2105.03824.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. Gshard: Scaling giant models with conditional computation and automatic sharding. In *ICLR*.

Ilya Loshchilov and Frank Hutter. 2017. Fixing weight decay regularization in Adam. *CoRR*, abs/1711.05101.

H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2018. Learning differentially private language models without losing accuracy. In *ICLR*.

Ilya Mironov. 2017. Rényi differential privacy. In *CSF*, pages 263–275.

Zachary Nado, Justin Gilmer, Christopher J. Shallue, Rohan Anil, and George E. Dahl. 2021. A large batch optimizer reality check: Traditional, generic optimizers suffice across batch sizes. *CoRR*, abs/2102.06356.

Venkatadheeraj Pichapati, Ananda Theertha Suresh, Felix X Yu, Sashank J Reddi, and Sanjiv Kumar. 2019. Adaclip: Adaptive clipping for private SGD. *CoRR*, abs/1908.07643.

Tim Salimans and Durk P Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *NIPS*, pages 901–909.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *ACL*, pages 1715–1725.

Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. 2019. Measuring the effects of data parallelism on neural network training. *JMLR*, 20(112):1–49.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *ICML*, pages 4596–4604.

Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. 2018. Don't decay the learning rate, increase the batch size. In *ICLR*.

Pranav Subramani, Nicholas Vadivelu, and Gautam Kamath. 2020. Enabling fast differentially private SGD via just-in-time compilation and vectorization. *CoRR*, abs/2010.09063.

Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. 2021. MLP-Mixer: An all-MLP architecture for vision. *CoRR*, abs/2105.01601.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*, pages 5998–6008.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*.

XLA team and collaborators. 2017. XLA: Optimizing compiler for machine learning. `https://developers.googleblog.com/2017/03/xla-tensorflow-compiled.html`.

Yuanzhong Xu, HyoukJoong Lee, Dehao Chen, Blake Hechtman, Yanping Huang, Rahul Joshi, Maxim Krikun, Dmitry Lepikhin, Andy Ly, Marcello Maggioni, et al. 2021. GSPMD: General and scalable parallelization for ML computation graphs. *CoRR*, abs/2105.04663.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large batch optimization for deep learning: Training BERT in 76 minutes. In *ICLR*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *ICCV*, page 19–27.