SOLA: Continual Learning with Second-Order Loss Approximation

Dong Yin  
DeepMind  
Mountain View, CA 94043  
dongyin@google.com

Mehrdad Farajtabar  
DeepMind  
Mountain View, CA 94043  
farajtabar@google.com

Ang Li  
DeepMind  
Mountain View, CA 94043  
anglili@google.com

Abstract

Neural networks have achieved remarkable success in many cognitive tasks. However, when they are trained sequentially on multiple tasks without access to old data, it is observed that their performance on old tasks tend to drop significantly after the model is trained on new tasks. Continual learning aims to tackle this problem often referred to as catastrophic forgetting and to ensure sequential learning capability. We study continual learning from the perspective of loss landscapes and propose to construct a second-order Taylor approximation of the loss functions in previous tasks. Our proposed method does not require any memorization of raw data or their gradients, and therefore, offers better privacy protection. We theoretically analyze our algorithm from an optimization viewpoint and provide a sufficient and worst-case necessary condition for the gradient updates on the approximate loss function to be descent directions for the true loss function. Experiments on multiple continual learning benchmarks suggest that our method is effective in avoiding catastrophic forgetting and in many scenarios, outperforms several baseline algorithms that do not explicitly store the data samples.

1 Introduction

Neural networks are achieving human-level performance on many cognitive tasks including image classification [23] and speech recognition [16]. However, as opposed to humans, their acquired knowledge is comparably volatile and can be easily dismissed. Especially, the catastrophic forgetting phenomenon refers to the case when a neural network forgets the past tasks if it is not allowed to retrain or reiterate on them again [13, 28].

Continual learning is a research direction that aims to solve the catastrophic forgetting problem. Recent works tried to tackle this issue from a variety of perspectives. Regularization methods (e.g., [21, 47]) aim to consolidate the weights that are important to previous tasks while expansion based methods (e.g., [40, 46]) typically increase the model capacity to cope with the new tasks. Repetition based methods (e.g., [26, 5]) usually do not require additional and complex modules, however, they have to maintain a small memory of previous data and use them to preserve knowledge. Unfortunately, the performance boost of repetition based methods comes at the cost of storing previous data which may be undesirable whenever privacy is important. To address this issue, authors in [8] proposed a method to work with the gradients of the previous data to constrain the weight updates; however, this may still be subject to privacy issues as the gradient associated with each individual data point may disclose information about the raw data.

In this paper, we study the continual learning problem from the perspective of loss landscapes. We explicitly target minimizing an average over all tasks’ loss functions. The proposed method stores neither the data samples nor the individual gradients on the previous tasks. Instead, we propose to construct an approximation to the loss surface of previous tasks. More specifically, we approximate
the loss function by estimating its second-order Taylor expansion. The approximation is used as a surrogate added to the loss function of the current task. Our method only stores information based on the statistics of the entire training dataset, such as full gradient and full Hessian matrix (or its low rank approximation), and thus better protects privacy. In addition, since we do not expand the model capacity, the neural network structure is less complex than that of expansion based methods.

We study our algorithm from an optimization perspective, and make the following theoretical contributions:

• We prove a sufficient and worst-case necessary condition under which by conducting gradient descent on the approximate loss function, we can still minimize the actual loss function.

• We further provide convergence analysis of our algorithm for both non-convex and convex loss functions. Our results imply that early stopping can be helpful in continual learning.

• We make connections between our method and elastic weight consolidation (EWC) [21].

In addition, we make the following experimental contributions:

• We conduct a comprehensive comparison among our algorithm and several baseline algorithms [21, 5, 8] on a variety of combinations of datasets and models. We observe that in many scenarios, especially when the learner is not allowed to store the raw data samples, our proposed algorithm outperforms them. We also discuss the conditions under which the proposed method or any of the alternatives are effective.

• We provide experimental evidence validating the importance of accurate approximation of the Hessian matrix and discuss scenarios in which early stopping is helpful for our algorithm.

2 Related work

Avoiding catastrophic forgetting in continual learning [34, 3] is an important milestone towards achieving artificial general intelligence (AGI) which entails developing measurements [43, 20], evaluation protocols [9, 7], and theoretical understanding [32, 10] of the phenomenon. Generally speaking, three classes of algorithms exist to overcome catastrophic forgetting [8].

The expansion based methods allocate new neurons or layers or modules to accommodate new tasks while utilizing the shared representation learned from previous ones [40, 45, 46, 25, 17]. Although being a very natural approach the mechanism of dynamic expansion can be quite complex and can add considerable overhead to the training process.

The repetition and memory based methods store previous data or, alternatively, train a generative model of them and replay samples from them interleaved with samples drawn from the current task [41, 19, 48, 38, 27, 26, 8]. They achieve promising performance however at the cost of higher risk of users’ privacy by storing or learning a generative model of their data.

The regularization based approaches impose limiting constraints on the weight updates of the neural network according to some relevance score for previous knowledge [21, 33, 42, 39, 29, 47, 55]. These methods provide a better privacy guarantee as they do not explicitly store the data samples. In general, SOLA also belongs to this category as we use the second-order Taylor expansion as the regularization term in new tasks. Many of the regularization methods are derived from a Bayesian perspective of estimating the posterior distribution of the model parameters given the data from a sequence of tasks [21, 33, 42, 39]; some of these methods use other heuristics to either estimate the importance of the weights of the neural network [47, 55] or implicitly limit the capacity of the network [29]. Similar to our approach, several regularization based methods use quadratic functions as the regularization term, and many of them use the diagonal form of quadratic functions [21, 47, 55]. In Section 5.3, we demonstrate that in some cases, the EWC algorithm [21] can be considered as the diagonal approximation of our approach. Here, we note that the diagonal form of quadratic regularization has the drawback that it does not take the interaction between the weights into account.

Among the regularization based methods, the online Laplace approximation algorithm [39] is the most similar one to our proposed method. Despite the similarity in the implementations, the two algorithms are derived from very different perspectives: the online Laplace approximation algorithm uses a Bayesian approach that approximates the posterior distribution of the weights with a Gaussian
distribution, whereas our algorithm is derived from an optimization viewpoint using Taylor approximation of loss functions. More importantly, the Gaussian approximation in [39] is proposed as a heuristic; whereas in this paper, we provide rigorous theoretical analysis on how the approximation error affects the optimization procedure. We believe that our analysis provides deeper insights to the loss landscape of continual learning problems, and explains some important implementation details such as early stopping.

We also note that continual learning is broader than just solving the catastrophic forgetting and is connected to many other areas such as meta learning [37], few-shot Learning [44] [12], learning without explicit task identifiers [36] [2], to name a few.

3 Problem formulation

We consider a sequence of $K$ supervised learning tasks $\mathcal{T}_k$, $k \in [K]$. For task $\mathcal{T}_k$, there is an unknown distribution $\mathcal{D}_k$ over the space of feature-label pairs $\mathcal{X} \times \mathcal{Y}$. Let $\mathcal{W} \subseteq \mathbb{R}^d$ be a model parameter space, and for the $k$-th task, let $\ell_k(w; x, y): \mathcal{W} \rightarrow \mathbb{R}$ be the loss function of $w$ associated with data point $(x, y)$. The population loss function of task $\mathcal{T}_k$ is defined as $L_k(w) := \mathbb{E}_{(x, y) \sim \mathcal{D}_k} \ell_k(w; x, y)$. Our general objective is to learn a parametric model with minimized population loss over all the $K$ tasks. More specifically, in continual learning, the learner follows the following protocol: When learning on the $k$-th task, the learner obtains access to $n_k$ data points $(x_{k,i}, y_{k,i}), i \in [n_k]$ sampled i.i.d. according to $\mathcal{D}_k$ and we define $\hat{L}_k(w) := \frac{1}{n_k} \sum_{i=1}^{n_k} \ell_k(w; x_{k,i}, y_{k,i})$ as the empirical loss function; the learner then updates the model parameter $w$ using these $n_k$ training data, and after the training procedure is finished, the learner loses access to the training data, but can store some side information about the task. Our goal is to avoid forgetting previous tasks when trained on new tasks by utilizing the side information. We provide details of our algorithm design in the next section.

4 Our approach

To measure the effectiveness of a continual learning algorithm, we use a simple criterion that after each task, we hope the average population loss over all the tasks that have been trained on to be small, i.e., for every $k \in [K]$, after training on $\mathcal{T}_k$, we hope to solve $\min_{w \in \mathcal{W}} \frac{1}{k} \sum_{k=1}^{K} L_k(w)$. Since minimizing the loss function is the key to training a good model, we propose a straightforward method for continual learning: storing the second-order Taylor expansion of the empirical loss function, and using it as a surrogate of the loss function for an old task when training on new tasks. We start with a simple setting. Suppose that there are two tasks, and at the end of $\mathcal{T}_1$, the we obtain a model $\hat{w}_1$. Then we compute the gradient and Hessian matrix of $\hat{L}_1(w)$ at $\hat{w}_1$, and construct the second-order Taylor expansion of $\hat{L}_1(w)$ at $\hat{w}_1$:

$$\hat{L}_1(w) = \hat{L}_1(\hat{w}_1) + (w - \hat{w}_1)\nabla \hat{L}_1(\hat{w}_1) + \frac{1}{2}(w - \hat{w}_1)^\top \nabla^2 \hat{L}_1(\hat{w}_1)(w - \hat{w}_1).$$

When training on $\mathcal{T}_2$, we try to minimize $\frac{1}{2}(\hat{L}_1(w) + \hat{L}_2(w))$. The basic idea of this design is that, we hope in a neighborhood around $\hat{w}_1$, the quadratic function $\hat{L}_1(w)$ stays as a good approximation of $\hat{L}_1(w)$, and thus approximately we still minimize the average of the empirical loss functions $\frac{1}{2}(\hat{L}_1(w) + \hat{L}_2(w))$, which in the limit generalizes to the population loss function $\frac{1}{2}(L_1(w) + L_2(w))$.

We rely on the assumption that the second-order Taylor approximation of loss function can capture their local geometry well. For a general nonlinear function and arbitrary displacement, this approximation can be over-simplistic, however, we refer to the abundance of observations for modern neural networks that are seen to be well-behaved with flat and wide minima [6] [14]. Moreover, the assumption of well-behaved loss around tasks’ local minima also forms the basis of a few other continual learning algorithms such as EWC [21] and OGD [8].

1 For any positive integer $N$, we define $[N] := \{1, 2, \ldots, N\}$.
2 In most cases, we consider $\mathcal{W} = \mathbb{R}^d$. 

3
Algorithm 1

Continual learning with second-order loss approximation (SOLA)

1: **Input:** initial weights \( \hat{w}_0 \), learning rate \( \eta \), the number of tasks \( K \), the rank of Hessian approximation \( r \) (for option II)
2: for \( k = 1, 2, \ldots, K \) do
3: access training data for the \( k \)-th task \( (x_{k,i}, y_{k,i}) \), \( i \in [n_k] \), \( w \leftarrow \hat{w}_{k-1} \)
4: while termination condition not satisfied do
5: compute (stochastic) gradient of current loss \( \nabla \tilde{L}_k(w) \)
6: compute gradient of loss function approximation \( \nabla \tilde{L}_{k-1}(w) \) \( (\tilde{L}_0(w) \equiv 0) \)
7: \( w \leftarrow w - \frac{\eta}{\epsilon} (\nabla \tilde{L}_k(w) + \nabla \tilde{L}_{k-1}(w)) \)
8: end while
9: \( \hat{w}_k \leftarrow w \), and \( H_k \leftarrow \begin{cases} \nabla^2 \tilde{L}_k(\hat{w}_k) & \text{option I} \\ \text{rank } r \text{ approximation of } \nabla^2 \tilde{L}_k(\hat{w}_k) & \text{option II} \end{cases} \)
10: \( \tilde{L}_k(w) \leftarrow \tilde{L}_{k-1}(w) + (w - \hat{w}_k)^\top \nabla \tilde{L}_k(\hat{w}_k) + \frac{1}{2} (w - \hat{w}_k)^\top H_k(w - \hat{w}_k) \)
11: end for

5 Theoretical analysis

In this section, we provide theoretical analysis of our algorithm. As we can see, the key idea in our algorithm is to approximate the loss functions of previous tasks using quadratic functions. This leads to the following theoretical question: By running gradient descent algorithm on an approximate loss function, can we still minimize the actual loss that we are interested in?

For the purpose of theoretical analysis, we make a few simplifications to our setup. Without loss of generality, we study the training process of the last task \( T_K \), and still use \( \hat{w}_k \) to denote the model parameters that we obtain at the end of the \( k \)-th task. We use the loss function approximation in \( \{\} \), but for simplicity we ignore the finite-sample effect and replace the empirical loss function with the
We also assume that the error between the matrices \( \nabla^2 \tilde{L}_k(\tilde{w}_k) \) or its low rank approximation. The reason for this simplification is that our focus is the optimization aspect of the problem, while the generalization aspect can be tackled by tools such as uniform convergence [30]. As discussed, during the training of the last task, we have access to the approximate loss function \( \tilde{F}(w) := \frac{1}{K} (\tilde{L}_{K-1}(w) + L_K(w)) \), whereas the actual loss function that we care about is \( F(w) := \frac{1}{K} \sum_{k=1}^{K} L_k(w) \). We also focus on gradient descent instead of its stochastic counterpart. In particular, let \( \tilde{w}_0 := \tilde{w}_{K-1} \) be the initial model parameter for the last task. We run the following update for \( t = 1, 2, \ldots, T \):

\[
w_t = w_{t-1} - \eta \nabla \tilde{F}(w_{t-1}).
\]

We use the following standard notions for differentiable function \( f : \mathcal{W} \to \mathbb{R} \).

**Definition 1.** \( f \) is \( \mu \)-smooth if \( \|\nabla f(w) - \nabla f(w')\|_2 \leq \mu \|w - w'\|_2, \forall w, w' \in \mathcal{W} \).

**Definition 2.** \( f \) is \( \rho \)-Hessian Lipschitz if \( \|\nabla^2 f(w) - \nabla^2 f(w')\|_2 \leq \rho \|w - w'\|_2, \forall w, w' \in \mathcal{W} \).

We make the assumptions that the loss functions are smooth and Hessian Lipschitz. We note that the Hessian Lipschitz assumption is standard in analysis of non-convex optimization [31][18].

**Assumption 1.** We assume that \( L_k(w) \) is \( \mu \)-smooth and \( \rho \)-Hessian Lipschitz \( \forall k \in [K] \).

We also assume that the error between the matrices \( H_k \) and \( \nabla^2 L_k(\tilde{w}_k) \) is bounded.

**Assumption 2.** We assume that for every \( k \in [K] \), \( \|H_k\|_2 \leq \mu \), where \( \mu \) is defined in Assumption 1 and that \( \|H_k - \nabla^2 L_k(\tilde{w}_k)\|_2 \leq \delta \) for some \( \delta \geq 0 \).

### 5.1 Sufficient and worst-case necessary condition for one-step descent

We begin with analyzing a single step during training. Our goal is to understand by running a single step of gradient descent on \( \tilde{F}(w) \), whether we can minimize the actual loss function \( F(w) \). More specifically, we have the following result.

**Theorem 1.** Under Assumptions 1 and 2 and suppose that in the \( t \)-th iteration, we observe

\[
\|\nabla \tilde{F}(w_{t-1})\|_2 \geq \frac{c}{K} \sum_{k=1}^{K-1} \delta \|w_{t-1} - \tilde{w}_k\|_2 + \rho \|w_{t-1} - \tilde{w}_k\|_2, \text{ for some } c > 1,
\]

and the learning rate satisfies \( \eta \leq \frac{2(1-1/c)}{\mu} \), then we have

\[
F(w_t) \leq F(w_{t-1}) - \eta(1 - \frac{1}{c} - \frac{\mu \eta}{2})\|\nabla \tilde{F}(w_{t-1})\|_2.
\]

We prove Theorem 1 in Appendix B. Here, we emphasize that this result does not assume any convexity of the loss functions. The theorem provides a sufficient condition (4), under which by running gradient descent on \( \tilde{F} \), we can still minimize the true loss function \( F \). Intuitively, this condition requires the gradient of \( \tilde{F} \) to be large enough, such that the magnitude of the gradient is larger than the error caused by the inexactness of the loss function. In Proposition 1 below, we will see that this condition is also necessary in the worst-case scenario, at least for the case where \( K = 2 \). More specifically, we can construct cases in which (4) is violated and the gradients of \( F(w) \) and \( \tilde{F}(w) \) have opposite directions.

**Proposition 1.** Suppose that \( K = 2, d = 1, \mathcal{W} = [0, 1] \). Then, there exists \( \tilde{w}_1, L_1(w), \tilde{L}_1(w) \), and \( L_2(w) \) such that if \( \|\tilde{F}'(w)\| < \frac{1}{2}[\delta \|w - \tilde{w}_1\| + \rho (w - \tilde{w}_1)^2] \), then \( \tilde{F}'(w) \cdot F'(w) < 0 \).

We prove Proposition 1 in Appendix B. In addition, we note that Theorem 1 also implies that as training going on and \( \|\nabla \tilde{F}(w_t)\|_2 \) decreasing, it is beneficial to decrease the learning rate \( \eta \), since when \( c \) decreases, the upper bound on \( \eta \) that guarantees the decay of \( F \) (i.e., \( 2(1-1/c)/\mu \)) also decreases. We notice that the importance of learning rate decay for continual learning has been observed in some empirical study recently [29].
5.2 Convergence analysis

Although the condition in (\ref{eq:cond}) provides us with insights on the dynamics of the training algorithm, it is usually hard to check this condition in every step, since we may not have good estimates of $\delta$ and $\rho$. A practical implementation is to choose a constant learning rate along with an appropriate number of training steps. In this section, we provide bounds on the convergence behavior of our algorithm with a constant learning rate and $T$ iterations, both for non-convex and convex loss functions. These results imply that early stopping can be helpful, and provide a theoretical treatment of the very intuitive fact that the more iterations one optimizes for the current task the more forgetting can happen for the previous ones. We begin with a convergence analysis for non-convex loss functions in Theorem\ref{thm:non-convex} in which we use the common choice of learning rate $1/\mu$ for gradient descent on smooth functions \cite{bottou2008tribes}.

Theorem 2 (non-convex). Let $F_0 := F(w_0)$, $F^* := \min_{w \in \mathcal{W}} F(w)$, $\tilde{F}_0 := \tilde{F}(w_0)$, and $\tilde{F}^* := \min_{w \in \mathcal{W}} \tilde{F}(w)$. Then, under Assumptions\ref{asm:smooth} and\ref{asm:convex} after running $T$ iterations of the gradient descent update (\ref{eq:grad_update}) with learning rate $\eta = 1/\mu$, we have

$$
\frac{1}{T} \sum_{t=1}^{T} \|\nabla F(w_{t-1})\|_2 \leq \frac{\alpha}{\sqrt{T}} + \beta + \gamma_1 \sqrt{T} + \gamma_2 T,
$$

where $\alpha = \sqrt{2\mu (F_0 - F^*)}$, $\beta = \sqrt{\frac{\mu}{K}} \sum_{k=1}^{K-2} \delta \|w_0 - \tilde{w}_k\|_2 + 2\rho \|w_0 - \tilde{w}_k\|_2^2$, $\gamma_1 = \frac{3}{\mu} (\tilde{F}_0 - \tilde{F}^*)$, and $\gamma_2 = \frac{4\rho}{\mu} (\tilde{F}_0 - \tilde{F}^*)$.

We prove Theorem\ref{thm:non-convex} in Appendix\ref{app:support}. Unlike standard optimization analysis, the average norm of the gradients does not always decrease as $T$ increases, when $\delta \neq 0$ or $\rho \neq 0$. Intuitively, as we move far from the points where we conduct Taylor expansion, the gradient of $\tilde{F}$ becomes more and more inaccurate, and thus we need to stop early. In Section\ref{sec:experiments} we provide experimental evidence.

When the loss functions are convex, we can prove a better guarantee which does not have the $O(\sqrt{T})$ and $O(T)$ terms as in Theorem\ref{thm:non-convex}. More specifically, we have the following assumption and theorem.

Assumption 3. $L_k(w)$ is convex and $H_k \succeq 0$, $\forall k \in [K]$.

Theorem 3 (convex). Suppose that Assumptions\ref{asm:smooth},\ref{asm:convex} hold, and define $F^* = \min_{w \in \mathcal{W}} F(w)$, $w^* \in \arg \min_{w \in \mathcal{W}} F(w)$, $\tilde{w}^* \in \arg \min_{w \in \mathcal{W}} \tilde{F}(w)$, and $\tilde{D} := \|w_0 - \tilde{w}^*\|_2$. After running $T$ iterations of the gradient descent update (\ref{eq:grad_update}) with learning rate $\eta = 1/\mu$, we have

$$
F(w_T) - F^* \leq \frac{\alpha}{T} + \beta,
$$

where $\alpha = 2\mu \tilde{D}$, and $\beta = \frac{1}{4\mu} \sum_{k=1}^{K-1} \delta (\|w^* - \tilde{w}_k\|_2^2 + 2\tilde{D}^2 + 2\|\tilde{w}^* - \tilde{w}_k\|_2^2 + \rho (\|w^* - \tilde{w}_k\|_2^2 + 4\tilde{D}^2 + 4\|\tilde{w}^* - \tilde{w}_k\|_2^2)$. We prove Theorem\ref{thm:convex} in Appendix\ref{app:convex}. As we can see, if $\delta \neq 0$ or $\rho \neq 0$, we still cannot guarantee the convergence to the true minimum of $F$, due to the inexactness of $\tilde{F}$. On the other hand, if the loss functions are quadratic and we save the full Hessian matrices, i.e., $\delta = \rho = 0$, as we have full information about previous loss functions, we can recover the standard $O(1/T)$ convergence rate for gradient descent on convex and smooth functions.

5.3 Connection to EWC

The elastic weight consolidation (EWC) algorithm\cite{nilsback2017exemplar} for continual learning is proposed based on the Bayesian idea of estimating the posterior distribution of the model parameters. Interestingly, we notice that our algorithm has a connection with EWC, although their basic ideas are quite different. More specifically, we show that in some cases, the regularization technique that the EWC algorithm uses can be considered as a diagonal approximation of the Hessian matrix of the loss function. Suppose that in the $k$-th task, the data points are samples from a probabilistic model with the likelihood function being $p_k(x, y | w_k^*)$, and we use negative log-likelihood as the loss function, i.e., $\ell_k(w; x, y) = -\log p_k(x, y | w)$. Suppose that at the end of this task, we obtain the ground truth model parameter $w_k^*$. Then we know that $\nabla L_k(w_k^*) = 0$, and that the Fisher information of the $i$-th coordinate of $w_k^*$ is $I_i(w_k^*) = \partial^2 L_k(w_k^*)$. The EWC algorithm constructs a regularization term
We implement the experiments with TensorFlow [1]. When computing the exact or the low rank approximation of the Hessian matrix, we treat each tensor in the model independently; in other words, we compute the block diagonal approximation of the Hessian matrix. This technique has the benefit that the Hessian computation is independent of the model architecture and has been used in recent studies on second-order optimization [15]. We use the recursive implementation for SOLA with low rank approximation in Section 4. We formally present this approach in Algorithm 2. In our experiments in Section 7, we use the recursive implementation for SOLA with low rank approximation.

Algorithm 2 Recursive implementation of SOLA with low rank approximation

\begin{algorithm}
\begin{algorithmic}
\State \textbf{Input:} initial weights $\tilde{w}_0$, learning rate $\eta$, the number of tasks $K$, rank $r$
\For{$k = 1, 2, \ldots, K$}
\State access training data for the $k$-th task $(x_{k,i}, y_{k,i})$, $i \in [n_k]$, $w \leftarrow \tilde{w}_{k-1}$
\While{termination condition not satisfied}
\State compute (stochastic) gradient of current loss $\tilde{\nabla}L_k(w)$
\State compute gradient of loss function approximation $\nabla \tilde{L}_{k-1}(w)$ ($\tilde{L}_0(w) \equiv 0$)
\State $w \leftarrow w - \frac{\eta}{k}(\tilde{\nabla}L_k(w) + \nabla \tilde{L}_{k-1}(w))$
\EndWhile
\State $\tilde{w}_k \leftarrow w$, $\tilde{L}_{k-\frac{1}{2}}(w) \leftarrow \tilde{L}_{k-1}(w) + \tilde{L}_k(w)$, $Q_{k-\frac{1}{2}} \leftarrow \text{rank-}$ $r$ approximation of $\nabla^2 \tilde{L}_{k-\frac{1}{2}}(\tilde{w}_k)$
\State $\tilde{L}_{k}(w) \leftarrow \tilde{L}_{k-\frac{1}{2}}(\tilde{w}_k) + (w - \tilde{w}_k)^\top \nabla \tilde{L}_{k-\frac{1}{2}}(\tilde{w}_k) + \frac{1}{2}(w - \tilde{w}_k)^\top Q_{k-\frac{1}{2}}(w - \tilde{w}_k)$
\EndFor
\end{algorithmic}
\end{algorithm}

7 Experiments

We implement the experiments with TensorFlow [1]. When computing the exact or the low rank approximation of the Hessian matrix, we treat each tensor in the model independently; in other words, we compute the block diagonal approximation of the Hessian matrix. This technique has the benefit that the Hessian computation is independent of the model architecture and has been used in recent studies on second-order optimization [15]. We use the recursive implementation for SOLA with low rank approximation in Section 4. We formally present this approach in Algorithm 2. In our experiments in Section 7, we use the recursive implementation for SOLA with low rank approximation.

Datasets. We use multiple standard continual learning benchmarks created based on MNIST [24] and CIFAR-10 [22] datasets, i.e., Permuted MNIST [13], Rotated MNIST [26], Split MNIST [47], and Split CIFAR (similar to a dataset in [5]). In Permuted MNIST, for each task, we choose a random permutation of the pixels of MNIST images, and reorder all the images according to the permutation. We use 5-task Permuted MNIST in the experiments. In Rotated MNIST, for each task, we rotate the MNIST images by a particular angle. In our experiments, we choose a 5-task Rotated MNIST, with the rotation angles being 0, 10, 20, 30, and 40 degrees. For Split MNIST, we Split the 10 labels
EWC when training on new tasks; the regularization based we only optimize the cross-entropy loss over the logits and labels of the corresponding output head.

In OGD, we store 200 gradient samples for each task, and in SOLA-prox, we use $r = 500$; for other models, in OGD, we store 280 gradient samples for each task, and in SOLA-prox, we use $r = 700$.

Table 1: Average test accuracy (%) ± std. P-MNIST, R-MNIST, and S-MNIST represent Permuted, Rotated, and Split MNIST datasets, respectively. Boldface numbers correspond to best result among algorithms that do not store raw data points, i.e., excluding multi-task and A-GEM. For MLP[10, 10], in OGD, we store 200 gradient samples for each task, and in SOLA-prox, we use $r = 500$; for other models, in OGD, we store 280 gradient samples for each task, and in SOLA-prox, we use $r = 700$.

| Dataset        | P-MNIST       | P-MNIST       | R-MNIST       | R-MNIST       | R-MNIST       | R-MNIST       | S-MNIST       | S-MNIST       |
|----------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| Model size     | [10, 10]      | [100, 100]    | [10, 10]      | [100, 100]    | [100, 100]    | 4-conv        | [10, 10]      | [100, 100]    |
| Multi-task     | 91.8 ± 0.4    | 97.0 ± 0.1    | 91.4 ± 0.4    | 97.5 ± 0.1    | 98.8 ± 0.1    | 98.9 ± 0.3    | 99.3 ± 0.1    |
| A-GEM          | 84.1 ± 1.1    | 93.2 ± 0.4    | 83.6 ± 1.0    | 92.6 ± 0.4    | 95.3 ± 0.3    | 91.2 ± 4.9    | 97.8 ± 0.4    |
| Vanilla        | 69.2 ± 3.1    | 81.1 ± 1.6    | 76.8 ± 0.9    | 86.0 ± 0.5    | 89.5 ± 0.6    | 86.4 ± 6.6    | 97.2 ± 0.9    |
| EWC            | 69.1 ± 3.7    | 80.2 ± 1.4    | 76.9 ± 1.0    | 86.1 ± 0.6    | 89.4 ± 0.7    | 87.7 ± 9.2    | 97.7 ± 0.8    |
| OGD            | 68.9 ± 3.3    | 81.5 ± 1.7    | 81.1 ± 1.3    | 88.0 ± 0.7    | 89.5 ± 0.7    | 97.1 ± 1.8    | 98.8 ± 0.1    |
| SOLA-exact     | **90.0 ± 0.9**| **88.6 ± 0.9**| **85.0 ± 0.6**| **90.4 ± 0.5**| **92.2 ± 1.5**| **96.1 ± 2.5**| **99.0 ± 0.2**|
| SOLA-prox      | 86.2 ± 1.5    | **87.8 ± 0.6**| 86.5 ± 0.9    | **90.4 ± 0.5**| **92.2 ± 1.5**| **96.1 ± 2.5**| **99.0 ± 0.2**|

Table 2: Average test accuracy (%) ± std on Split CIFAR. Boldface numbers correspond to best result among algorithms that do not store raw data, i.e., excluding multi-task and A-GEM. For OGD we store 200 gradient samples for each task, and for SOLA-prox, we choose $r = 200$.

| Model         | Multi-task | A-GEM     | Vanilla | EWC      | OGD      | SOLA-exact | SOLA-prox |
|---------------|------------|-----------|---------|----------|----------|------------|-----------|
| CNN-2         | 75.9 ± 0.9 | 65.8 ± 2.1| 57.2 ± 4.2| 55.6 ± 4.6| 56.5 ± 4.2| **62.0 ± 5.4**| 59.4 ± 3.8|
| CNN-6         | 78.6 ± 1.4 | 68.1 ± 2.3| 57.5 ± 4.6| 57.7 ± 3.8| 58.3 ± 4.8| –          | **58.6 ± 5.2**|
| MLP[200, 200] | 69.2 ± 0.5 | 66.1 ± 0.7| 63.5 ± 1.6| 63.8 ± 2.1| **65.8 ± 1.2**| –          | 55.7 ± 3.2 |

of the MNIST dataset to disjoint subsets, and for each task, we use the MNIST data whose labels belong to a particular subset. In this paper, we use a 5-task Split MNIST, and the subsets of labels are \{0, 1\}, \{2, 3\}, \{4, 5\}, \{6, 7\}, and \{8, 9\}. Split CIFAR is defined similar to Split MNIST, and we use a 2-task Split CIFAR with the label subsets being \{0, 1, 2, 3, 4\} and \{5, 6, 7, 8, 9\}.

Architecture. We use both multilayer perceptron (MLP) and convolutional neural network (CNN). In most cases, we use MLP with two hidden layers, sometimes denoted by MLP[x, y], with x and y being the number of hidden units. We may use CNN-x to denote a CNN model with x convolutional layers, and provide details of the model in Appendix F. For Split MNIST and Split CIFAR, we use MLP and CNN models with a multi-head structure similar to what has been used in [5, 8]. In the multi-head model, instead of having 10 logsit in the output layer, we use separate heads for different tasks, and each head corresponds to the classes of the associated task. During training, for each task, we only optimize the cross-entropy loss over the logits and labels of the corresponding output head.

Baselines. We compare SOLA algorithm with several baselines: the vanilla algorithm which runs SGD over all the tasks without storing any side information; the multi-task algorithm which assumes access to all the training data of previous tasks; the repetition based A-GEM algorithm [5], which stores a subset of data samples from the previous tasks and forms constrained optimization algorithms when training on new tasks; the regularization based EWC algorithm [21] discussed in Section 5.3 and the orthogonal gradient descent (OGD) algorithm [8] that stores the gradients in previous tasks and forms a constrained optimization algorithm. Among them, our algorithm, along with the vanilla, EWC, and OGD algorithms do not explicitly store the raw data samples. Following prior works [5, 8, 21], we choose a learning rate of $10^{-3}$ and a batch size of 10. For all the results that we report, we present the average result over 10 independent runs, as well as the standard deviation (as the shaded areas in the figures).

Results. We provide a comprehensive comparison among SOLA and the baseline algorithms with a variety of combinations of datasets and models. Tables 1 and 2 present the results for MNIST-based datasets and Split CIFAR, respectively. We make a few notes before discussing the results. First, the multi-task algorithm uses all the data of previous tasks, which serves as an upper bound for the

\footnote{Comparison between OGD and SOLA-prox: Since OGD has a memory cost that grows linearly with the number of tasks but SOLA-prox does not, we keep the average memory cost of them the same.}
performance of continual learning algorithms. Second, since the A-GEM algorithm stores a subset of data samples from previous tasks, it is not completely fair to compare A-GEM with algorithms that do not store raw data. However, here we still report the results for A-GEM for reference, and in A-GEM we store 200 data points for each task. Third, since the performance of the algorithms depends on the number of epochs that we train for each task, we treat this quantity as a tuning parameter, and for each algorithm, we report the result corresponding to the best epoch choice for its performance. In particular, for MNIST-based datasets, we choose epoch from \{5, 10, 20\}, and for Split CIFAR, we choose from \{1, 5, 10, 20, 40\}. Due to memory constraints, we only implement SOLA-exact on small models such as MLP\{10, 10\} and CNN-2. We conclude from the results as follows:

- If it is allowed to store raw data, repetition based algorithm such as A-GEM should be the choice. This remarks the importance of the information contained in the raw data samples. In some cases we observe that SOLA outperforms A-GEM, e.g., on MLP\{10, 10\}. However, we expect that the performance of A-GEM can be improved if more data are stored in memory.

- If it is not allowed to store raw data due to privacy concerns, then in many scenarios, SOLA outperforms the baseline algorithms. In particular, on MNIST-based datasets, SOLA-exact or SOLA-prox achieves the best performance in 6 out of 7 settings.

- On Split CIFAR, we observe mixed results. When the model is relatively small (CNN-2) and we can store the exact Hessian matrix, SOLA-exact achieves the best performance. On a relatively large CNN model (CNN-6), we observe that none of the continual learning algorithms (EWC, OGD, SOLA) significantly outperforms the vanilla algorithm. On a large MLP, we observe that OGD performs the best and the result for SOLA-prox becomes worse. We believe the reason is that since in this experiment we only use 200 eigenvectors to approximate a Hessian matrix with very high dimensions, the approximation error is so large that SOLA-prox cannot find a descent direction that is close to the true gradient. This remarks the importance of future study of SOLA on models with more complicated structure or higher dimensions.

**Performance vs approximation.** We study how the approximation of Hessian matrices affects the performance of SOLA-prox. In particular, we choose different values of the rank \( r \) in SOLA-prox and investigate its correlation with the final average test accuracy. Our theory implies that when the approximation of Hessian matrices is better, i.e., smaller \( \delta \), the final performance is better. Our experiments validate this point. Figure 1a and Figure 1b show that, as we increase \( r \), i.e., using more eigenvectors to approximate the Hessian matrix, the average test accuracy over all tasks improves.

**Early stopping.** Our theoretical analysis in Section 5 implies that early stopping can be helpful for SOLA. Here, we discuss empirical evidence. As we can see from Figure 1c on Permutated MNIST with MLP\{10, 10\}, the average test accuracy of SOLA-prox becomes worse if we train more than 5 epochs per task; similarly, from Figure 1d we can also see that training each task for more epochs can hurt the performance of MLP\{100, 100\} on Permutated MNIST. However, this phenomenon is less severe on Rotated MNIST. In Figure 1e for SOLA-prox with \( r = 100 \), we observe one case where the average test accuracy gradually decreases as we increase the number of epochs per task.
Moreover, we notice that we did not observe this phenomenon for SOLA-exact. Hence, we draw the conclusion that the importance of early stopping for SOLA depends on how different the tasks are and how well we approximate the Hessian matrix. In Permuted MNIST, the pixels are randomly shuffled when switching to new tasks, whereas in Rotated MNIST we only rotate the images by 10 degrees; thus early stopping is more important for Permuted MNIST. On the other hand, if we store the exact value of the Hessian matrix ($\delta = 0$ in Theorem 1), the approximation error of the gradients can be small, and thus we can train more epochs on new tasks. In addition, we note that it has been observed that early stopping is typically helpful for other continual learning algorithms [8].

8 Conclusions

We propose the SOLA algorithm based on the idea of loss function approximation. We establish theoretical guarantees, make connections to the EWC algorithm, and present experimental results showing that in many scenarios, our algorithm outperforms several baseline algorithms, especially among the ones that do not explicitly store the raw data samples. Future directions include studying SOLA on broader classes of neural network architectures and parameter spaces with higher dimensions.

Acknowledgements

We would like to thank Dilan Gorur, Alex Mott, Clara Huiyi Hu, Nevena Lazic, Nir Levine, and Michalis Titsias for helpful discussions.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pages 265–283, 2016.
[2] R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio. Online continual learning with no task boundaries. arXiv preprint arXiv:1903.08671, 2019.
[3] S. Beaulieu, L. Frati, T. Miconi, J. Lehman, K. O. Stanley, J. Clune, and N. Cheney. Learning to continually learn. arXiv preprint arXiv:2002.09571, 2020.
[4] S. Bubeck. Convex optimization: Algorithms and complexity. arXiv preprint arXiv:1405.4980, 2014.
[5] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny. Efficient lifelong learning with A-GEM. arXiv preprint arXiv:1812.00420, 2018.
[6] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In Artificial intelligence and statistics, pages 192–204, 2015.
[7] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonards, G. Slabaugh, and T. Tuytelaars. Continual learning: A comparative study on how to defy forgetting in classification tasks. arXiv preprint arXiv:1909.08383, 2019.
[8] M. Farajtabar, N. Azizan, A. Mott, and A. Li. Orthogonal gradient descent for continual learning. In International Conference on Artificial Intelligence and Statistics, 2020.
[9] S. Farquhar and Y. Gal. Towards robust evaluations of continual learning. arXiv preprint arXiv:1805.09733, 2018.
[10] S. Farquhar and Y. Gal. A unifying Bayesian view of continual learning. arXiv preprint arXiv:1902.06494, 2019.
[11] B. Ghorbani, S. Krishnan, and Y. Xiao. An investigation into neural net optimization via Hessian eigenvalue density. arXiv preprint arXiv:1901.10159, 2019.
[12] S. Gidaris and N. Komodakis. Dynamic few-shot visual learning without forgetting. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4367–4375, 2018.
[13] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[14] I. J. Goodfellow, O. Vinyals, and A. M. Saxe. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.

[15] V. Gupta, T. Koren, and Y. Singer. Shampoo: Preconditioned stochastic tensor optimization. *arXiv preprint arXiv:1802.09568*, 2018.

[16] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[17] G. Jerfel, E. Grant, T. L. Griffiths, and K. A. Heller. Reconciling meta-learning and continual learning with online mixtures of tasks. In *Advances in Neural Information Processing Systems*, 2019.

[18] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan. How to escape saddle points efficiently. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1724–1732. JMLR.org, 2017.

[19] N. Kamra, U. Gupta, and Y. Liu. Deep generative dual memory network for continual learning. *arXiv preprint arXiv:1710.10368*, 2017.

[20] R. Kemker, M. McClure, A. Abitino, T. L. Hayes, and C. Kanan. Measuring catastrophic forgetting in neural networks. In *Thirty-second AAAI Conference on Artificial Intelligence*, 2018.

[21] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[22] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[25] X. Li, Y. Zhou, T. Wu, R. Socher, and C. Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. *arXiv preprint arXiv:1904.00310*, 2019.

[26] D. Lopez-Paz and M. Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017.

[27] B. Lüders, M. Schläger, and S. Risi. Continual learning through evolvable neural turing machines. In *NIPS 2016 Workshop on Continual Learning and Deep Networks (CLDL 2016)*, 2016.

[28] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[29] S.-I. Mirzadeh, M. Farajtabar, and H. Ghasemzadeh. Dropout as an implicit gating mechanism for continual learning. *arXiv preprint arXiv:2004.11545*, 2020.

[30] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[31] Y. Nesterov and B. T. Polyak. Cubic regularization of newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006.

[32] C. V. Nguyen, A. Achille, M. Lam, T. Hassner, V. Mahadevan, and S. Soatto. Toward understanding catastrophic forgetting in continual learning. *arXiv preprint arXiv:1908.01091*, 2019.

[33] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[34] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter. Continual lifelong learning with neural networks: A review. 2018.
[35] D. Park, S. Hong, B. Han, and K. M. Lee. Continual learning by asymmetric loss approximation with single-side overestimation. In Proceedings of the IEEE International Conference on Computer Vision, pages 3335–3344, 2019.

[36] D. Rao, F. Visin, A. Rusu, R. Pascanu, Y. W. Teh, and R. Hadsell. Continual unsupervised representation learning. In Advances in Neural Information Processing Systems, pages 7645–7655, 2019.

[37] M. Riemer, I. Cases, R. Ajemian, M. Liu, I. Rish, Y. Tu, and G. Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. arXiv preprint arXiv:1810.11910, 2018.

[38] A. Rios and L. Itti. Closed-loop GAN for continual learning. arXiv preprint arXiv:1811.01146, 2018.

[39] H. Ritter, A. Botev, and D. Barber. Online structured Laplace approximations for overcoming catastrophic forgetting. In Advances in Neural Information Processing Systems, pages 3738–3748, 2018.

[40] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. arXiv preprint arXiv:1606.04671, 2016.

[41] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. In Advances in Neural Information Processing Systems, pages 2990–2999, 2017.

[42] M. K. Titsias, J. Schwarz, A. G. d. G. Matthews, R. Pascanu, and Y. W. Teh. Functional regularisation for continual learning using Gaussian Processes. arXiv preprint arXiv:1901.11356, 2019.

[43] M. Toneva, A. Sordoni, R. T. d. Combes, A. Trischler, Y. Bengio, and G. J. Gordon. An empirical study of example forgetting during deep neural network learning. arXiv preprint arXiv:1812.05159, 2018.

[44] J. Wen, Y. Cao, and R. Huang. Few-shot self reminder to overcome catastrophic forgetting. arXiv preprint arXiv:1812.00543, 2018.

[45] T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In Proceedings of the 22nd ACM International Conference on Multimedia, pages 177–186. ACM, 2014.

[46] J. Yoon, E. Yang, J. Lee, and S. J. Hwang. Lifelong learning with dynamically expandable networks. In International Conference on Learning Representations. ICLR, 2018.

[47] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 3987–3995. JMLR, 2017.

[48] M. Zhang, T. Wang, J. H. Lim, and J. Feng. Prototype reminding for continual learning. arXiv preprint arXiv:1905.09447, 2019.
Appendix

A Proof of Theorem 1

We first provide a bound for the difference between the gradients of $\nabla \hat{F}(w)$ and $\nabla F(w)$.

Lemma 1. Let $\Delta(w) = \nabla \hat{F}(w) - \nabla F(w)$. Then we have

$$\|\Delta(w)\|_2 \leq \frac{1}{K} \sum_{k=1}^{K-1} \delta \|w - \hat{w}_k\|_2 + \rho \|w - \hat{w}_k\|_2^2.$$  

We prove Lemma 1 in Appendix A.1. Since the loss functions for all the tasks $L_k(w)$ are $\mu$-smooth, we know that $F(w)$ is also $\mu$-smooth. Then we have

$$F(w_t) \leq F(w_{t-1}) + \langle \nabla F(w_{t-1}), w_t - w_{t-1} \rangle + \frac{\mu}{2} \|w_t - w_{t-1}\|_2^2$$

$$= F(w_{t-1}) + \langle \nabla \hat{F}(w_{t-1}) - \Delta(w_{t-1}), -\eta \nabla \hat{F}(w_{t-1}) \rangle + \frac{\mu \eta^2}{2} \|\nabla \hat{F}(w_{t-1})\|_2^2$$

$$\leq F(w_{t-1}) - \eta(1 - \frac{\mu \eta}{2}) \|\nabla \hat{F}(w_{t-1})\|_2^2 + \eta \|\nabla \hat{F}(w_{t-1})\|_2 \|\Delta(w_{t-1})\|_2.$$  

Therefore, as long as $\|\nabla \hat{F}(w_{t-1})\|_2 \geq c \|\Delta(w_{t-1})\|_2$ for some $c > 1$, we have

$$F(w_t) \leq F(w_{t-1}) - \eta(1 - \frac{1}{c} - \frac{\mu \eta}{2}) \|\nabla \hat{F}(w_{t-1})\|_2^2.$$  

(7)

Then we can complete the proof by combining (7) with Lemma 1.

A.1 Proof of Lemma 1

By the definition of $\hat{F}(w)$, for some $\xi_k \in [0, 1]$, $k \in [K - 1]$, we have

$$\Delta(w) = \frac{1}{K} \sum_{k=1}^{K-1} \nabla L_k(\hat{w}_k) + H_k(w - \hat{w}_k) - \nabla L_k(w)$$

$$= \frac{1}{K} \sum_{k=1}^{K-1} \left( H_k - \nabla^2 L_k(\hat{w}_k + \xi_k (w - \hat{w}_k)) \right) (w - \hat{w}_k),$$

where the second equality is due to Lagrange’s mean value theorem. Then, according to Assumptions 1 and 2 we have

$$\|H_k - \nabla^2 L_k(\hat{w}_k + \xi_k (w - \hat{w}_k))\|_2 \leq \delta + \|\nabla^2 L_k(\hat{w}_k) - \nabla^2 L_k(\hat{w}_k + \xi_k (w - \hat{w}_k))\|_2$$

$$\leq \delta + \rho \|w - \hat{w}_k\|_2.$$  

(8)

Then, according to triangle inequality, we obtain

$$\|\Delta(w)\|_2 \leq \frac{1}{K} \sum_{k=1}^{K-1} \delta \|w - \hat{w}_k\|_2 + \rho \|w - \hat{w}_k\|_2^2.$$  

B Proof of Proposition 1

We first note that it suffices to construct $F(w)$ and $\hat{F}(w)$, as one can always choose $L_2(w) \equiv 0$ and then the construction of $F(w)$ and $\hat{F}(w)$ is equivalent to that of $L_1(w)$ and $\hat{L}_1(w)$. Let $\hat{w}_1 = 0$,

$$F(w) = (w - 1)^2 + \frac{\rho}{6} w^3, \ w \in [0, 1],$$

$$\hat{F}(w) = (w - 1)^2 - \frac{\delta}{4} w^2, \ w \in [0, 1].$$
We then proceed to bound which implies we know that both $\parallel F(w) \parallel$ is equivalent to $\parallel \tilde{F}(w) \parallel$ is smooth. Since $F''(w) \equiv \rho$, we know that $F(w)$ is $\rho$-Hessian Lipschitz. Therefore, $F(w)$ and $\tilde{F}(w)$ satisfy all of our assumptions.

Since $\tilde{F}'(w) = 2(w-1) - \frac{\delta}{2}w$, we know that $\tilde{F}'(w) < 0, \forall w \in [0, 1]$, and then

$$|\tilde{F}'(w)| < \frac{\delta}{2}w + \frac{\rho}{2}w^2$$

is equivalent to $\frac{\delta}{2}w - 2(w-1) < \frac{\delta}{2}w + \frac{\rho}{2}w^2$, which implies that $F(w) = 2(w-1) + \frac{\rho}{2}w^2 > 0$.

**C Proof of Theorem 2**

Similar to Appendix A, we define $\Delta(w) = \nabla \tilde{F}(w) - \nabla F(w)$. According to Assumptions 1 and 2 we know that both $F(w)$ and $F$ are $\mu$-smooth. By the smoothness of $F(w)$ and using the fact that $\eta = 1/\mu$, we get

$$F(w_t) \leq F(w_{t-1}) + \langle \nabla F(w_{t-1}), w_t - w_{t-1} \rangle + \frac{\mu}{2} \parallel w_t - w_{t-1} \parallel^2$$

$$= F(w_{t-1}) - \langle \nabla F(w_{t-1}), \eta(\nabla F(w_{t-1}) + \Delta(w_{t-1})) \rangle + \frac{\mu \eta^2}{2} \parallel \nabla F(w_{t-1}) + \Delta(w_{t-1}) \parallel^2$$

$$= F(w_{t-1}) - \frac{1}{2\mu} \parallel \nabla F(w_{t-1}) \parallel^2 + \frac{1}{2\mu} \parallel \Delta(w_{t-1}) \parallel^2,$$

which implies

$$\parallel \nabla F(w_{t-1}) \parallel^2 \leq 2\mu(F(w_{t-1}) - F(w_t)) + \parallel \Delta(w_{t-1}) \parallel^2.$$

By averaging (9) over $t = 1, \ldots, T$, we get

$$\frac{1}{T} \sum_{t=1}^{T} \parallel \nabla F(w_{t-1}) \parallel^2 \leq \frac{2\mu(F_0 - F^*)}{T} + \frac{1}{T} \sum_{t=1}^{T} \parallel \Delta(w_{t-1}) \parallel^2.$$

By taking square root on both sizes, and using Cauchy-Schwarz inequality as well as the fact that $\sqrt{a + b} \leq \sqrt{a} + \sqrt{b}$, we get

$$\frac{1}{T} \sum_{t=1}^{T} \parallel \nabla F(w_{t-1}) \parallel \leq \sqrt{\frac{2\mu(F_0 - F^*)}{\sqrt{T}}} + \sqrt{\frac{1}{T} \sum_{t=1}^{T} \parallel \Delta(w_{t-1}) \parallel^2}.$$

We then proceed to bound $\parallel \Delta(w_{t-1}) \parallel^2$. According to Lemma 1, we have

$$\parallel \Delta(w_{t-1}) \parallel^2 \leq \frac{1}{K} \sum_{k=1}^{K-1} \delta(\parallel w_{t-1} - w_{0} \parallel^2 + \parallel w_{0} - \tilde{w}_{k} \parallel^2) + 2\rho(\parallel w_{t-1} - w_{0} \parallel^2 + \parallel w_{0} - \tilde{w}_{k} \parallel^2)$$

$$:= C + \delta \parallel w_{t-1} - w_{0} \parallel^2 + 2\rho \parallel w_{t-1} - w_{0} \parallel^2,$$

where $C := \frac{1}{K} \sum_{k=1}^{K-1} \delta \parallel w_{0} - \tilde{w}_{k} \parallel^2 + 2\rho \parallel w_{0} - \tilde{w}_{k} \parallel^2$ does not depend on the iteration count $t$. By Cauchy-Schwarz inequality, we get

$$\parallel \Delta(w_{t-1}) \parallel^2 \leq 3C^2 + 3\delta^2 \parallel w_{t-1} - w_{0} \parallel^2 + 12\rho^2 \parallel w_{t-1} - w_{0} \parallel^2.$$

Then we bound $\parallel w_{t-1} - w_{0} \parallel^2$. By triangle inequality and Cauchy-Schwarz inequality, we obtain

$$\parallel w_{t-1} - w_{0} \parallel^2 = \parallel -\eta \sum_{\tau=0}^{t-2} \nabla \tilde{F}(w_{\tau}) \parallel^2 \leq \eta \sum_{\tau=0}^{t-2} \parallel \nabla \tilde{F}(w_{\tau}) \parallel^2 \leq \eta \sqrt{(t-1) \sum_{\tau=0}^{t-2} \parallel \nabla \tilde{F}(w_{\tau}) \parallel^2},$$

and therefore

$$\parallel w_{t-1} - w_{0} \parallel^2 \leq \eta^2(t-1) \sum_{\tau=0}^{t-2} \parallel \nabla \tilde{F}(w_{\tau}) \parallel^2.$$
On the other hand, since $\tilde{F}(w)$ is also $\mu$-smooth, we have for every $t \geq 1$,
\[
\tilde{F}(w_t) \leq \tilde{F}(w_{t-1}) + \langle \nabla \tilde{F}(w_{t-1}), w_t - w_{t-1} \rangle + \frac{\mu}{2} \|w_t - w_{t-1}\|_2^2
\]
\[
= \tilde{F}(w_{t-1}) - \frac{1}{2\mu} \|\nabla \tilde{F}(w_{t-1})\|_2^2,
\]
where in the equality we use the fact that $\eta = 1/\mu$. This implies that
\[
\sum_{\tau=0}^{t-2} \|\nabla \tilde{F}(w_{t-1})\|_2^2 \leq 2\mu(\tilde{F}_0 - \tilde{F}^*).
\] (13)
By combining (12) and (13), we obtain
\[
\|w_{t-1} - w_0\|_2^2 \leq \frac{2}{\mu}(\tilde{F}_0 - \tilde{F}^*)(t-1),
\] (14)
and combining (11) and (14), we get
\[
\|\Delta(w_{t-1})\|_2^2 \leq 3C^2 + \frac{6\delta^2}{\mu}(\tilde{F}_0 - \tilde{F}^*)(t-1) + \frac{48\rho^2}{\mu^2}(\tilde{F}_0 - \tilde{F}^*)^2(t-1)^2.
\] (15)
By averaging (15) over $t = 1, \ldots, T$ and plugging the result in (10), we obtain
\[
\frac{1}{T} \sum_{t=1}^{T} \|\nabla F(w_{t-1})\|_2 \leq \frac{\sqrt{2\mu(\tilde{F}_0 - \tilde{F}^*)}}{\sqrt{T}} + \sqrt{3C + \delta} \sqrt{\frac{3}{\mu}(\tilde{F}_0 - \tilde{F}^*)\sqrt{T} + \frac{48\rho^2}{\mu^2}(\tilde{F}_0 - \tilde{F}^*)T},
\]
which completes the proof.

D Proof of Theorem 3

Let $\tilde{F}^* = \min_{w \in \mathcal{W}} \tilde{F}(w)$. Since we run gradient descent with learning rate $\eta = 1/\mu$ on the convex and $\mu$-smooth function $\tilde{F}$, according to standard results in convex optimization [4], we have
\[
\tilde{F}(w_T) - \tilde{F}^* \leq \frac{2\mu D^2}{T}.
\] (16)
We provide the following lemma that bounds the difference between $\tilde{F}(w)$ and $F(w)$.

Lemma 2. For any $w \in \mathcal{W}$, we have
\[
|F(w) - \tilde{F}(w)| \leq \frac{1}{2K} \sum_{k=1}^{K-1} \delta \|w - \hat{w}_k\|_2^2 + \rho\|w - \hat{w}_k\|_2^3
\]
We prove Lemma 2 in Appendix D.1. Here, we proceed to analyze $F(w_T) - F^*$. We have
\[
F(w_T) - F^* = F(w_T) - \tilde{F}(w_T) + \tilde{F}(w_T) + \tilde{F}^* + \tilde{F}^* - F^*
\]
\[
\leq F(w_T) - \tilde{F}(w_T) + \tilde{F}^* - F^* + \frac{2\mu D^2}{T}.
\] (17)
To bound $F(w_T) - \tilde{F}(w_T)$, we use the fact that for any convex and smooth functions, when we run gradient descent with learning rate $1/\mu$, the iterates only move closer to the minimum of the function, i.e., $\|w_T - \tilde{w}^*\|_2 \leq \tilde{D}$. Then, by triangle inequality, for any $k \in [K-1]$
\[
\|w_T - \tilde{w}_k\|_2 \leq \|w_T - \tilde{w}^*\|_2 + \|\tilde{w}^* - \tilde{w}_k\|_2 \leq \tilde{D} + \|w_0 - \tilde{w}_k\|_2.
\]
Using the fact that for any two positive numbers $a$ and $b$, $(a + b)^2 \leq 2(a^2 + b^2)$ and $(a + b)^3 \leq 4(a^3 + b^3)$, we obtain
\[
\|w_T - \tilde{w}_k\|_2 \leq 2(\tilde{D}^2 + \|\tilde{w}^* - \tilde{w}_k\|_2^2),
\] (18)
\[
\|w_T - \tilde{w}_k\|_2^3 \leq 4(\tilde{D}^3 + \|\tilde{w}^* - \tilde{w}_k\|_2^3).
\] (19)
By combining (18) and (19) with Lemma 2, we obtain

\[ F(w_T) - \tilde{F}(w_T) \leq \frac{1}{K} \sum_{k=1}^{K-1} \delta(\tilde{D}^2 + \|\tilde{w}^* - \tilde{w}_k\|_2^2) + 2\rho(\tilde{D}^3 + \|\tilde{w}^* - \tilde{w}_k\|_2^3). \] (20)

We can use a similar argument to bound \( \tilde{F}^* - F^* \). Note that \( \tilde{F}^* - F^* \leq \tilde{F}(w^*) - F(w^*) \). Therefore, according to Lemma 2, we have

\[ \tilde{F}^* - F^* \leq \frac{1}{2K} \sum_{k=1}^{K-1} \delta\|w^* - \tilde{w}_k\|_2^2 + \rho\|w^* - \tilde{w}_k\|_2^3. \] (21)

Then we can complete the proof by combining (17), (20), and (21).

### D.1 Proof of Lemma 2

By definition of \( F(w) \) and \( \tilde{F}(w) \), we have

\[
F(w) - \tilde{F}(w) \leq \frac{1}{K} \sum_{k=1}^{K-1} L_k(w) - \left( L_k(\tilde{w}_k) + (w - \tilde{w}_k)^\top \nabla L_k(\tilde{w}_k) + \frac{1}{2}(w - \tilde{w}_k)^\top H_k(w - \tilde{w}_k) \right)
\]

\[
= \frac{1}{K} \sum_{k=1}^{K-1} \frac{1}{2} \left( w - \tilde{w}_k \right)^\top \left( \nabla^2 L_k(\tilde{w}_k + \xi_k(w - \tilde{w}_k)) - H_k \right) (w - \tilde{w}_k),
\]

for some \( \xi_k \in [0, 1] \). Then according to [8], we get

\[ |F(w) - \tilde{F}(w)| \leq \frac{1}{2K} \sum_{k=1}^{K-1} \|\nabla^2 L_k(\tilde{w}_k + \xi_k(w - \tilde{w}_k)) - H_k\|_2 \|w - \tilde{w}_k\|_2^2 \leq \frac{1}{2K} \sum_{k=1}^{K-1} \delta\|w - \tilde{w}_k\|_2^2 + \rho\|w - \tilde{w}_k\|_2^3. \]

### E Power method for low rank approximation of a Hessian matrix

#### Algorithm 3 Power method for low rank approximation of Hessian matrix

**Input:** rank \( r \), number of power iterations \( p \), loss function \( \hat{L}(\cdot) \), evaluation point \( w \), set of eigenvalues \( \Lambda \leftarrow \emptyset \), set of eigenvectors \( V \leftarrow \emptyset \)

**while** \( |\Lambda| < r \) **do**

\( v \leftarrow N(0, I) \)

**for** \( p' = 1, \ldots, p \) **do**

\( v \leftarrow \nabla(v^\top \nabla \hat{L}(w)), \lambda = \|v\|_2 \)

\( v \leftarrow \text{ProjectAndNormalize}(v, V) \)

end **for**

\( \Lambda \leftarrow \Lambda \cup \{\lambda\}, V \leftarrow V \cup \{v\} \)

**end while**

**return** \( \Lambda, V \)

**ProjectAndNormalize**\((v, V)\):

**for every** \( v_i \) in \( V \) **do**

\( v \leftarrow v - \langle v, v_i \rangle v_i \)

**end for**

**return** \( v/\|v\|_2 \)

We note that for any fixed vector \( v \) and function \( F(w), \nabla(v^\top \nabla F(w)) = \nabla^2 F(w)v \). Therefore, it is easy to compute Hessian-vector products in TensorFlow using its automatic differentiation function. Then, we are able to compute the rank-\( r \) approximations of Hessian matrices via power method. We present details in Algorithm 3. In our experiments, we use 5 power iterations for each eigenvector.
F Experiment details

In this section, we provide details for the neural network architecture of the CNN models that we use in this paper. We denote a convolutional layer with $x$ output channels, a kernel of size $y$, and a stride of $z$ by $\text{conv}(x, y, z)$. We denote a fully connected layer with $x$ output units as $\text{full}(x)$. We denote a max pooling layer with kernel size $x$ and stride $y$ by $\text{pool}(x, y)$. We denote ReLU activation by $\text{ReLU}$.

In the results in Table 1, we use a CNN with 4 convolutional layers for Rotated MNIST. The architecture of this network is as follows:

$$
\text{conv}(32, 4, 1) \rightarrow \text{ReLU} \rightarrow \text{conv}(32, 4, 1) \rightarrow \text{ReLU} \rightarrow \text{pool}(2, 2) \rightarrow \text{conv}(64, 4, 1) \rightarrow \text{ReLU} \rightarrow \\
\text{conv}(64, 4, 1) \rightarrow \text{ReLU} \rightarrow \text{pool}(2, 2) \rightarrow \text{full}(10) \rightarrow \text{ReLU} \rightarrow \text{full}(10).
$$

In the results in Table 2, we use two CNN models for Split CIFAR. The first model has 2 convolutional layers (CNN-2):

$$
\text{conv}(16, 3, 2) \rightarrow \text{ReLU} \rightarrow \text{conv}(16, 3, 2) \rightarrow \text{ReLU} \rightarrow \text{full}(10) \rightarrow \text{ReLU} \rightarrow \text{full}(10),
$$

and the second model has 6 convolutional layers (CNN-6):

$$
\text{conv}(16, 3, 1) \rightarrow \text{ReLU} \rightarrow \text{conv}(16, 3, 1) \rightarrow \text{ReLU} \rightarrow \text{pool}(2, 2) \rightarrow \text{conv}(32, 3, 1) \rightarrow \text{ReLU} \rightarrow \\
\text{conv}(32, 3, 1) \rightarrow \text{ReLU} \rightarrow \text{pool}(2, 2) \rightarrow \text{conv}(64, 3, 1) \rightarrow \text{ReLU} \rightarrow \text{conv}(64, 3, 1) \rightarrow \text{ReLU} \rightarrow \\
\text{pool}(2, 2) \rightarrow \text{full}(10) \rightarrow \text{ReLU} \rightarrow \text{full}(10).
$$