Budgeted Gradient Descent: Selective Gradient Optimization for Addressing Misclassifications in DNNs

by

Arghasree Banerjee

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Artificial neural networks have become a popular learning approach for their ability to generalize well to unseen data. However, misclassifications can still occur due to various data-related issues, such as adversarial inputs, out-of-distribution samples, and model-related challenges, such as underfitting and overfitting. While retraining and fine-tuning on misclassified samples are common corrective approaches, they can reduce generalizability and lead to sample memorization.

In this thesis, we propose Budgeted Gradient Descent (BGD), an approach for correcting misclassifications by introducing sparse changes to network parameters. Our approach attempts to answer the question: *What is the minimal set of network changes necessary to correctly predict a previously misclassified sample?* BGD minimizes both the number of parameters updated and the magnitude of changes, aiming to correct misclassifications while preserving generalizability. Additionally, BGD does not require access to the training data to preserve said generalizability.

We observe that sparse updates can effectively correct misclassifications while preserving learned representations, as not all gradients contribute equally to classifying difficult or out-of-distribution samples. Through empirical comparisons with existing approaches, we investigate the optimal level of sparsity for maintaining network performance and generalizability. Our results suggest that while second-order gradient updates can minimize the number of parameter changes, excessive sparsity can negatively impact the network. The contri-

butions of this thesis include a novel approach to correcting misclassifications, insights into the relationship between parameter updates and generalizability, and a detailed examination of how different sparsity levels affect the long-term performance of neural networks in an *online supervised learning* setting.

To all my teachers

Acknowledgements

Firstly, I would like to thank my supervisor, Dr. Matthew Guzdial. Your feedback, constructive criticism, and encouragement were key to gaining confidence in myself and progress in research. Secondly, I would like to express gratitude for my examining committee, to take the time to read my thesis. Thridly, I am thankful to Dr. James Martens; our conversations helped me think about my research problem better. I also am thankful for the Graduate Google DeepMind Scholarship that supported my master's journey. Finally, I would like to thank my parents, Arnab Banerjee and Shawli Banerjee for being my rock. I am so grateful for my friends, Anushka, Sayani, Kushankur, Shashank, and Deep. I am made of parts of myself found by you.

Contents

List of Tables

List of Figures

List of Symbols

s_i	Input sample
y_i	Output label
D	Training dataset comprising samples and labels, (D_x, D_y)
C	Number of class labels
L	Number of layers
$O^{(L)}$	Last layer output
N	Neural network
$\mathbf{W}$	Set of network parameters
$\mathbf{W}_{\mathrm{R}}$	Set of network parameters after retraining
$\mathbf{W}'$	Subset of network parameters
$\nabla_{\mathbf{W}}\mathcal{L}$	Gradient of the loss function calculated with respect to the network parameters
$\bar{\mathbb{G}}$	Gradients of the loss function calculated with respect to the network parameters and averaged over entire training set
$\mathcal{L}(D_y, N(\mathbf{W}, D_x))$	Training loss
$\mathcal{L}(y_i, N(\mathbf{W}, s_i))$	Loss for sample s_i
H	Hessian matrix
P_t	Probability distribution over network parameters at t-th iteration
$\mathcal{V}(\mathbf{W}')$	Network fitness after updating parameters in set $\mathbf{W}'$

Chapter 1

Introduction

Artificial neural networks are a popular learning approach that learns mappings between inputs and outputs through calculations in a high-dimensional feature space. The learnt mappings, after convergence, are generalizable and therefore can reach near-perfect performance for inputs from a data distribution similar to the training data distribution. However, due to high-dimensional calculations and data dependencies during training, it is difficult to explain how the network arrives at its predictions. Understanding these predictions is especially important in the event of a misclassification.

With the popularity of deep neural networks in real-life applications, misclassifications can lead to serious repercussions [7], [42]. For example, failure to correctly classify racial and ethnic populations in American Indian/Alaska Natives can lead to incorrect estimates in federal statistics such as access to healthcare and health status. In order to address such issues, prior researchers have tried to identify the root cause. Data related issues such as the presence of adversarial inputs [37], out of distribution samples [9], [43], backdoor trigger patterns [29], or model related issues such as underfitting and overfitting [22] are examples of such misclassifcation root causes. However, as an end-user, it is not enough to understand the causes of misclassifications. It is necessary to fix the mistake.

A common approach to solve misclassifications is to retrain the entire network or a part of the network on the misclassified sample [21], [27]. However, this may lead to two unwanted consequences: reduced adaptability to the

misclassified sample, causing the network to keep misclassifying the sample, or disrupting previous learned representations to memorize the misclassified sample. These problems are often due to the 'stability-plasticity' dilemma in neural networks [4], [18]. For example, in an image classification setting, re-training on a misclassified image of the 'cat' class may cause the network to overwrite existing 'cat' class features with new, potentially non-generalizable features extracted from the misclassified sample. These might include irrelevant background details rather than distinctive 'cat' features. As a result, the network, by overfitting to the single sample, risks losing its previously learned representations. To restrict making changes to the extracted features, fine-tuning parameters in a particular layer stands as an alternative option. For example, prior researchers often fine-tune the network by performing gradient descent steps on the penultimate fully connected layer [46]. However, both re-training or fine-tuning update the network based on gradients calculated on a single sample, and are often associated with sample memorization, thus disrupting learnt representations [9], [15], [17]. Therefore, apart from solving misclassifications, preserving/increasing generalizability is a desirable characteristic of an optimization approach.

To preserve learnt features, it maybe helpful to re-train the network on the entire training data along with the misclassified sample. However, this may be unattainable in the absence of the training data. Retraining is also expensive, especially in an *online supervised learning setting* [17], where new data inputs are learnt by the network periodically. A common alternative to re-training is regularization where changes to the network are restricted by additional penalty terms to the loss function [26]. For example, the additional loss term can relate to the magnitudes of individual weights; this will result in attempting to learn the misclassified sample while minimizing parameter magnitudes [31]. This might be helpful in preserving previously learnt representations. While such techniques can resist significant changes to the network, it does not guarantee the correct prediction of the misclassified sample.

In this thesis, we propose an alternative approach to correcting the misclassification of a sample. Instead of constraining magnitude in all parameters,

we constrain the number of parameters that can be changed. In our proposed approach, Budgeted Gradient Descent (BGD), we aim to identify a subset of the total parameter set. By making targeted updates to this subset, we attempt to resolve misclassifications by reducing the loss for the misclassified samples. Additionally, BGD does not require access to the training data, and can be applied to any pre-trained network.

Motivated by the challenges highlighted by (full or partial) re-training approaches, the extracted parameter subset ought to have a few preferred characteristics: (a) all parameters in the subset correspond to the correct classification of the previously misclassified sample, i.e, there does not exist a smaller subset of the chosen subset that can solve the misclassification, (b) generalization abilities of the network is retained, i.e., there is no detrimental loss in the network's performance for both training and testing data. In later sections, we analyze if there exists a relationship between property (a) and property (b): if changes to smaller parameter subsets cause lower loss in generalizability.

In an ideal exhaustive setting, all combinations of parameters could be updated and evaluated to find the best solution to correct the misclassification. However, doing so is computationally infeasible given the size of the total parameter set for even small networks. In identifying the correct combination of parameters, we confront two main sub-problems: (a) how many parameters should be considered, and (b) which parameters out of all the parameters are better suited to solve the misclassification?

Our approach is guided by the gradients calculated on the misclassified sample. We use the calculated gradient as an 'utility' factor to estimate the importance of updating a certain parameter. Updating high gradient parameters can be produce lower sample loss, and thereby lead to correcting the misclassifications. Although the gradients provide intuition for solving the misclassification, the effect on generalizability cannot be estimated given the absence of training data. To bypass this issue we provide an additional workaround that assumes access to the training gradients, i.e., gradients to the loss averaged across the entire training data. We investigate if avoiding parameters that are highly sensitive to the training loss, can better preserve

network generalizability.

1.1 Thesis Contributions

The thesis contributions can be summarized as follows:

C1 We present a novel approach, Budgeted-Gradient Descent (BGD), that attempts to update the network to avoid misclassifications. The novelty of our approach lies in the sparsity of the parameter updates required to achieve this correction. We empirically show that sparse updates can be beneficial to preserve the generalizability of networks, even in the absence of training data.

C2 We explore whether the loss in generalizability can be minimized by avoiding updates to high-gradient parameters, with gradients calculated over the entire training dataset. While direct access to the training data may be restricted, we investigate if the preservation of learned representations is possible by assuming access to the gradients derived from the training data.

C3 We investigate effects on network generalizability by introducing sparse changes to the network in two different ways: (a) Aggressively changing a smaller subset of network parameters, (b) Updating a relatively larger subset of network parameters by relatively smaller magnitude updates. This provides an opportunity to infer which approach for introducing sparsity is better for preserving/increasing network generalizability.

C4 We investigate the long-term impact of excessive sparsity on the network's generalizability. By introducing sparse changes to correct multiple misclassifications, we compare the effect on generalizability with other approaches that have varying levels of sparsity. Sparse regularization techniques typically encourage the network to make predictions using fewer features. In our approach, we intentionally keep parameter changes minimal, making only the necessary adjustments to correct misclassifications. Given the small number of these changes, we examine

how excessive sparsity affects the network's generalizability over time as multiple misclassifications are incrementally corrected.

1.2 Thesis Outline

The thesis is outlined as follows:

Chapter 2: In this chapter, we provide the necessary background on neural networks and sparse regularization techniques.

Chapter 3: In this chapter, we first mention the Idealized Subset Selector, enumerate the corresponding complexities and introduce our proposed method, Budgeted Gradient Descent (BGD). BGD attempts to correct previous misclassifications without detrimental loss in generalizability.

Chapter 4: In this chapter, we evaluate our approach on the MNIST dataset for a fully connected network and AlexNet in *non-reloading* and *reloading* settings. We compare our approach to other relevant baselines. Additionally, we introduce two new variants of BGD and explore their corresponding advantages to correctly predict misclassified samples and preserve generalizability.

Chapter 5: In this chapter, we conclude this thesis by summarizing our findings, and discuss research questions that needs to be addressed in the future.

Chapter 2

Background

In this chapter, we provide essential background information on artificial neural networks. To address the misclassification problem, we first explain the methodology of neural networks, detailing how parameter values are learned iteratively via first-order and second-order update rules. Finally, we discuss additional optimization techniques from the literature that are used to overcome issues such as sample memorization and catastrophic forgetting.

2.1 Neural Networks

Neural Networks (NNs) are networks of interconnected artificial neurons that simulate the biological networks found in animal brains. In biological systems, neurons communicate information to other neurons via synapses after processing it. Similarly, in artificial neural networks, information is propagated between neurons through edges, with each edge having an associated weight that scales the input. In NN literature, weights are also referred to as parameters.

In a fully-connected network architecture, neurons are organized into layers, with each neuron in a layer connected to every neuron in the previous layer without intra-layer connections. Within each layer, neurons perform independent linear or non-linear transformations on the input using predefined activation functions. The output of each neuron, before being propagated forward, is scaled by the parameter value associated with the edge connecting it to the neurons in the next layer. Because these transformations incorporate

information from all neurons in the previous layer, each neuron in a layer performs its own distinct transformation. However, the transformation remains unique only if the network's parameters are initialized with different values for each neuron. This entire process of converting input data into an output vector through successive transformations across layers is known as a forward pass.

In a supervised classification setting, the input s_i, from an input-output pair, (s_i, y_i) is fed to a network N of L total layers. In the forward pass, s_i is subjected to linear transformations by parameters at the first layer, $\mathbf{W}^{(1)}$, and potentially non-linear transformations by the activation functions associated with each neuron. The input is then propagated through each layer, till it reaches the final layer. The final layer, also called the output layer, has dimensions equal to the number of classes. The total number of classes is therefore pre-determined, and based on the dataset D, comprising all the input-output pairs, $D = \{(s_0, y_0), (s_1, y_1), ..., (s_n, y_n)\}$. If there are a total of C classes, then the output vector is also of size C. The output, $O_j^{(L)}$, where $j = 1, ..., C$, represents the networks' confidence that the sample s_i belongs to the class j. The predicted output class is thereby calculated as $\underset{j}{argmax}(O_j^{(L)})$.

After the forward pass, if an incorrect class is predicted, a loss value quantifies how far off the network's prediction is from the actual target. This loss is indicative of how the parameters need to change to correct the misclassification, and is, therefore, back-propagated to individual parameters. During training, the network learns generalizable patterns, via backpropagation, that represent the training data distribution and can also ideally apply to unseen test data samples.

The fundamental idea of backpropagation is aimed at finding values for parameters, such that it minimizes the loss for the observed data samples (hereby refereed to as the training loss). To minimize said training loss, the network's parameter values need to be optimized. In gradient-descent (GD) based optimization approaches, the directional derivation of the loss function, also known as the gradient, indicates required changes to the parameters, such

that the loss is minimized. In the Taylor series approximation [1], a function f at $x + \Delta x$ is approximated without computing $f(x + \Delta x)$ directly by the following:

$$f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)(\Delta x)^2$$

In the context of neural networks, this can be re-formulated to approximate the loss function as:

$$\mathcal{L}(D_y, N(\mathbf{W}^{(t+1)}, D_x)) \simeq \mathcal{L}(D_y, N(\mathbf{W}^{(t)}, D_x)) + \nabla_{\mathbf{W}}\mathcal{L}\ \Delta\mathbf{W} + \frac{1}{2}\Delta\mathbf{W}^{\mathbf{T}}\mathrm{H}\Delta\mathbf{W}$$

Here, $\mathcal{L}(\mathbf{W}^{(t+1)}, N(D))$ is the training loss for the changed parameters, $\nabla_{\mathbf{w}}\mathcal{L}\ \Delta\mathbf{W}$ are the partial derivatives of the training loss computed with respect to individual parameters, and H is the Hessian matrix of size (d, d), where d is the total number of network parameters. The training loss, at iteration $t + 1$, $\mathcal{L}(\mathbf{W}^{(t+1)}, N(D))$ approximates the local minima. Therefore, taking the gradient at $\mathbf{W}^{(t+1)}$ will be zero.

$$\begin{aligned}\nabla_{\mathbf{W}}\mathcal{L}(D_y, N(\mathbf{W}^{(t+1)}, D_x)) &\simeq \nabla_{\mathbf{W}}\mathcal{L}(D_y, N(\mathbf{W}^{(t)}, D_x)) + \mathrm{H}\Delta\mathbf{W} \\ 0 &\simeq \nabla_{\mathbf{W}}\mathcal{L}(D_y, N(\mathbf{W}^{(t)}, D_x)) + \mathrm{H}\Delta\mathbf{W} \\ \Delta\mathbf{W} &\simeq -\mathrm{H}^{-1}\nabla_{\mathbf{W}}\mathcal{L}(D_y, N(\mathbf{W}^{(t)}, D_x))\end{aligned}$$

The Hessian matrix, H, estimates the local geometry of the loss curve, offering crucial insights into the optimization landscape. For instance, if the curvature of the loss is mostly flat, parameter values can be adjusted with larger steps without causing significant changes in the loss value. Conversely, a steeper curvature indicates that aggressive parameter updates should be avoided to prevent large increases in the loss. Natural gradient descent (NGD) uses such second-order information to calculate the updates to network parameters. In Eq 2.1 H^{-1} is the inverse of a (d, d) matrix. Since the number of parameters in deep neural networks are in the order of millions, computing the inverse of the hessian is computationally infeasible. Previous works have found that the Hessian is low in rank, and can therefore be estimated. One of such Hessian approximations [19], [38], [44] is the Empirical Fisher approximation [5], [13], given as:

$$\mathrm{H} \simeq \hat{F} = \frac{1}{|D|}\sum_{i=1}^{|D|}\nabla_{\mathbf{W}}\mathcal{L}(y_i, N(\mathbf{W}^{(t+1)}, s_i)) \cdot \nabla_{\mathbf{W}}\mathcal{L}(y_i, N(\mathbf{W}^{(t+1)}, s_i))^{\mathrm{T}} \quad (2.1)$$

Here, $\cdot$ is the outer product of gradients computed over individual training samples.

We can also make parameter updates without the second-order information. Stochastic gradient descent (SGD), a variant of Gradient Descent (GD), is a form of said back-propagation that has seen enormous success in the past years [28]. To avoid computing the inverse of the Hessian, SGD optimizes the loss function iteratively, and uses the first-order gradient information to update individual parameters. Instead of updating parameters by $-\mathrm{H}^{-1}\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W}^{(t)}, N(D))$, a learning rate, α is substituted in place of the inverted Hessian. The update rule in SGD is as follows:

$$\Delta\mathbf{W} = -\alpha\nabla_{\mathbf{W}}\mathcal{L}(y_i, N(\mathbf{W}^{(t)}, s_i)) \tag{2.2}$$

While in GD, the parameters are updated after computing the average gradients for the entire training set D, SGD updates parameters for each training sample, $s_i \in D$.

As the training process progresses, ideally, the transformations computed by neurons in the ultimate and penultimate layers would diverge, leading to specialized encodings that are distinct for different classes. This property is particularly useful for creating representations that capture the unique features of each class, enhancing the network's ability to differentiate between classes. However, in overparameterized network settings, correlated features are common, and can cause overfitting [32]. Therefore, works on sparse representations [6] focus on more generalized feature for network predictions. Such sparsity can be achieved by regularization methods. We elaborate more about these regularization methods in an upcoming section 2.2.

After training, the network's generalizability to predict unseen samples is empirically verified against test data. The test data distribution is ideally close to the training one, which allows a trained network to predict the majority of the test data accurately. However, a test dataset can also have outlier examples far from the training data distribution, resulting in misclassifications during inference.

2.2 Regularization Techniques

Neural networks commonly get stuck in the stability-plasticity dilemma, especially in *continual learning* and *online learning* settings. Neural networks learn and retain knowledge by attaining a balance between the two. Stability is when the neural network retains knowledge and is unaffected by small changes to the input. Too much stability will render the network incapable of learning new patterns from new data. Plasticity, on the other hand, allows the network to learn new patterns from new data. Too much of plasticity can lead to constant forgetting of past learned patterns and bias towards recent data patterns. For instance, when some input data is out of the training data distribution, the network must learn connections between features in a new way. This can cause the network to catastrophically forget past representations, and *memorize* that input data. Sample memorization is especially high when the network learns from outliers [12].

To achieve a balance between stability and plasticity, researchers commonly use constraints on learning objectives. An example of such constraint techniques, also known as regularization techniques, is L2-regularization, which is used to avoid extreme network plasticity [35]. As defined in Eq 2.3, optimizing the new loss, $\mathcal{L}_{l2}$, penalizes changes to the parameter magnitude along with the loss term associated with misclassifications. This forces the network to learn unique patterns from recent data, while ensuring small changes to the network parameters.

$$\mathcal{L}_{l2} = \mathcal{L}(D_y, N(\mathbf{W}, D_x)) + \lambda \sum_{i=1}^{|\mathbf{W}|} w_i^2 \tag{2.3}$$

2.2.1 Sparse Regularization Techniques

Neural networks have a large parameter space, which often leads to redundant and correlated structures. While L2-regularization shrinks magnitudes of parameters towards zero, it does not differentiate between important and unimportant parameters in predictions. On the other hand, L1-regularization (defined in Eq 2.4) introduces sparsity by making parameter magnitudes to

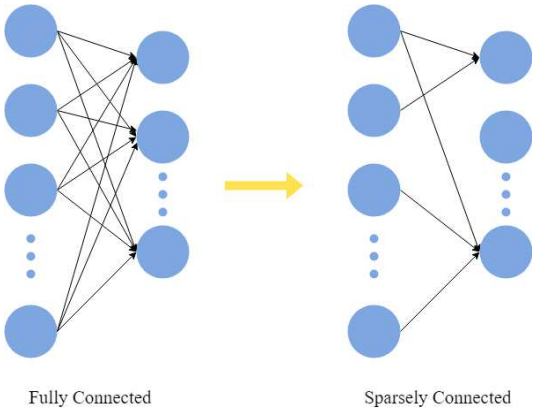


Figure 2.1: Converting a fully connected network into a sparse one. Sparse networks are achieved by approaches such as pruning, sparsifying gradients, and dropout.

zero. Therefore, parameters responsible for a prediction is limited to a sparser set. As such, L1-regularization is considered a form of sparse optimization [23].

$$\mathcal{L}_{l1} = \mathcal{L}(D_y, N(\mathbf{W}, D_x)) + \lambda \sum_{i=1}^{|\mathbf{W}|} |w_i| \tag{2.4}$$

As the term 'sparse' means 'scarce' or 'limited', the objective of sparse optimization approaches is to minimize the objective function, while making limited changes to the network. Regularization techniques similar to L1-regularization aim to constrain the number of parameters required for a certain prediction, and therefore limit over-reliance on a relatively large set of parameters [34]. Consecutively, this leads to a reduced risk of overfitting. Figure 2.1 shows a sparsely connected network, that is obtained after removing a part of the network.

Sparse regularization techniques aim to use a subset of the network, leading to increase in computational and space efficiency without significant loss in performance. Network pruning is a perfect example [16]. Based on certain predetermined criteria, the network prunes branches of its architecture. Research in pruning techniques include pruning based on activation values, gradients, and parameter magnitudes: (a) Activation-trace criterion eliminate neurons that do not contribute significant value to the prediction [11]. (b) Gradient based criterion eliminate neurons that have relatively low gradients on the loss

function. Works such as [14], [25], [36], [39], [41] use second order information to approximate changes to the loss values and therefore make better decisions about which parameters to prune. (c) Parameter based criterion eliminates parameters with low magnitudes. During the forward pass, high magnitude parameters are more likely to produce high activation values, therefore low magnitude parameters that are unnecessary can reduce complexity. After a subset of the network is pruned based on a pre-determined criterion, the resultant subnetwork needs to be re-trained to learn a denser representation.

Another example of constructing a sparse network popular in recent years is Dropout [32]. Dropout does not require a pre-specified criterion and only inputs p as a hyperparameter, where $1-p$ describes the probability of dropping a neuron (by making the outputs of neurons to be zero). This forces the network to learn more efficient connections between neurons.

Other sparse approaches need not prune the entire network. For example, sparsifying the gradients allows all parameters to participate in a forward pass, but restricts backpropagation to only a subset of parameters, thus minimizing risks of overfitting. Sun et al. [33] sparsify the gradient vector by selecting the top-k gradients, i.e., only the top k gradient parameters are updated during backpropagation, while the rest remain unchanged. Sparsification methods reduce computations without harming the accuracy [2] and have been shown to converge [3]. Few sparse optimization techniques have been found to produce better generalizability than in the original networks [40].

Although methods like top-k, dropout, and pruning can sparsify the network, and increase functionality in neurons, the network is still dependent on updates to a relatively large fraction of neurons to learn new information. For example, dropout with a Bernoulli(0.5) can make approximately 50% of neurons inactive. In addition, with aggressive pruning, up to 50-70% of the network can be pruned. 50-70% of millions of neurons is a large number. While sparsity decrease complexity for computing predictions, there is no fixed degree of sparsification that works for all networks. Therefore, in our approach, we aim to identify an aggressive sparse network that is beneficial to merely obtain a correction of misclassifications. We investigate this extreme form of

sparsification and its associated advantages and disadvantages.

Chapter 3

Budgeted-Gradient Descent

Learning new samples and correcting misclassifications via gradient descent is an important part of neural network training. During training using mini-batch SGD, the changes to parameters are averaged across multiple batches. SGD is typically performed on a particular sample, and can lead to restructuring the parameters into a solution that is biased on the features of said sample. This phenomenon is also called sample memorization, and re-training a pre-trained network on a particular sample can lead to sample memorization. This typically occurs when the sample is an outlier of the training data distribution and is therefore misclassified by the network. To avoid these negative repercussions, and motivated by sparse optimization techniques, we aim to regularize the network by limiting the number of altered parameters.

In this chapter we introduce our novel approach, Budgeted-Gradient Descent (BGD), where we avoid running gradient descent on all network parameters, and focus instead on altering a subset of the parameters with the motivation of correcting misclassifications. Apart from correct classifications of misclassifications, we aim to investigate if too much sparsity can cause generalization problems in the network. Therefore, in BGD, we propose an approach that can select the least number of required parameters that, after appropriate updates, can lead to solving misclassifications.

First, we start with introducing the idealized subset selector. An idealized subset selector would select different combinations of all parameters, thereby, making a total number of 2^n such subsets. An approximated ideal subset

is described in subsection 3.1. After discussing why this algorithm is too expensive as well to apply to real-life applications, we introduce our novel approach, which reduces the search space to a much smaller subset, such that the ideal selector can be applied to this subset.

3.1 Idealized Subset Selection

A misclassified sample s_i can be correctly predicted after several iterations of stochastic gradient descent (See Eq. 2.2), or a single iteration of natural gradient descent (See Eq 2.1). The network parameters that misclassify s_i, $\mathbf{W}_0$, after the re-training steps are represented by $\mathbf{W}_\mathrm{R}$. However, we aim to extract a minimal solution subset, such that all parameters in the subset, $\mathbf{W}'_{\mathbf{final}} \subseteq \mathbf{W}_R$, contribute to the correct classification of s_i. In an idealized subset selection scenario, the selector searches for the subset in an exhaustive manner.

To measure the contribution of each parameter $w_j \in \mathbf{W}_R$, the idealized subset selector evaluates if s_i can be correctly predicted without updating w_j. Here j refers to the positional index in the set of parameters W or W_R. It runs through the entire parameter set, $\mathbf{W}_\mathrm{R}$, and restores each parameter $(\mathbf{W}'_{\mathbf{final}})_j$ by $(\mathbf{W}_0)_j$, where $w_j = (\mathbf{W}_\mathrm{R})_j$. After restoring the value of w_j, if s_i is still correctly predicted, then the parameter w_j does not contribute to the correct prediction of the sample, and is therefore removed from the subset $\mathbf{W}'_{\mathbf{final}}$. This step is repeated for all network parameters to extract the smallest subset that can correctly classify the misclassified sample s_i. The Algorithm 1 describes the pseudocode for the Idealized subset selection.

The idealized selector updates the network for each parameter in a leave one out approach. If the number of parameters in $\mathbf{W}_R$ is m, and the computational cost for doing a forward pass in the network is K, then the idealized selector has a time complexity of $\mathcal{O}(Km)$. Since the number of parameters in DNNs are usually in the order of millions, the idealized selector is not scalable to real-life applications.

To reduce the computational cost, we propose Budgeted Gradient Descent

Algorithm 1 Pseudocode of the Approximate Idealized Subset Selector

Input: $\mathbf{W}_R$: Retrained parameter set; $\mathbf{W}_0$: Original parameter set; (s_i, y_i) : Sample;

Output: $\mathbf{W}'_{\mathbf{final}}$: Final subset;

$\mathbf{W}'_{\mathbf{final}} = \mathbf{W}_R$
for $j \leftarrow 1$ **to** $|\mathbf{W}_R|$ **do**
  $\mathbf{W}'_{sub} = \mathbf{W}'_{\mathbf{final}} - \{(\mathbf{W}_\mathrm{R})_j\} + \{(\mathbf{W}_0)_j\}$
  if $N(\mathbf{W}'_{sub}, s_i) == y_i$ **then**
    $\mathbf{W}'_{\mathbf{final}} = \mathbf{W}'_{sub}$ // if s_i is correctly predicted
  end if
end for
return $\mathbf{W}'_{\mathbf{final}}$

(BGD), which seeks to minimize the number of forward passes. Unlike a bottom-up approach that retrains all network parameters to correctly predict the sample and then does step-wise backward elimination to filter out those that do not contribute to correcting misclassifications, our method starts with the original network parameters $\mathbf{W}_0$, and iteratively samples a subset of parameters. By updating this subset, we aim to correctly classify the misclassified sample s_i. To achieve this, we need to address two key questions: (1) Which parameters should be included in the subset? (2) How much change should be introduced to the selected parameters to rectify the misclassification? These questions are answered in the following subsections. We end in Section 3.4, where we build upon the introduced concepts, to explain our algorithm.

3.2 Which parameters should be included into the subset?

During training, the network adjusts its parameter values by a fraction of the gradient calculated on the loss function. Before we introduce our approach, we will explain the importance of taking these gradient descent steps. The gradient is indicative of the direction which could lead to the maxima of the loss function, therefore following the opposite direction would lead to a lower sample loss. Intuitively, a lower loss for the misclassified sample would increase the likelihood that the network will correctly predict it. Therefore, the key

here is to follow the gradient. Ideally, by doing so, the network would learn the necessary features to correctly classify the sample without retraining the whole network on it.

Therefore, to estimate which parameters are most beneficial for rectifying the misclassification, we look at the gradients of the loss function calculated for the individual misclassified sample s_i, denoted as $\nabla_{\mathbf{W}}\mathcal{L}$. As the gradient is computed solely on the misclassified sample, it indicates the sensitivity of each parameter to the loss function for that sample.

Updating a relatively higher gradient is not always beneficial for the network's generalizability [30]. High gradient parameters are often associated with noise, and therefore selecting the top-k gradients may not yield a correct prediction. Additionally, this approach lacks the transparency needed to understand the relationship between individual parameters and the loss function. For example, consider the following scenario: altering the 8^{th} highest gradient parameter may cause the largest decrease in the sample loss, leading to the correct classification of the sample by adjusting the corresponding activation values. However, in a top-k scenario, it may be necessary to change up to all top 8 parameters to correctly predict the sample. Since our aim is to rectify the misclassification with the fewest changes possible, we avoid updating all top-k gradient parameters.

Another scenario where gradients can be misleading is when the loss landscape of the sample might vary from the expected loss landscape of the other samples from the same class, especially in cases of difficult samples. This can cause undesirable network behaviors such as sample memorization and the forgetting of learned representations. Therefore, while following the magnitude of the gradient may lead to a lower sample loss and correct sample prediction, it can potentially also disrupt learned representations by entering narrow valleys in the loss landscape [8].

Since the gradients (computed on a single misclassified sample) can potentially cause ineffectual updates to parameters, we assign probabilities to individual parameters. In our approach, we use the magnitude of gradients as a *priori* to select parameters. By doing so, we introduce stochasticity to

parameter sampling to promote variance in the gradient values of chosen parameters. For example, the initial probability of w_j is described by $P_0(\mathbf{W})$:

$$P_0(\mathbf{W}) = \left[\frac{\left| \frac{\partial \mathcal{L}}{\partial w_j} \right|}{\sum_{i=1}^{d} \left| \frac{\partial \mathcal{L}}{\partial w_i} \right|} \right]_{j=0}^{d}$$

Here $w_j \in \mathbf{W}$, for all $j = 1, 2, ..., d$, as $\mathbf{W} \in \mathbb{R}^d$. If the subset size is n, then we can sample n parameters given the probability distribution, and then evaluate if the corresponding changes to those n parameters succeed in lowering the sample loss and correctly predict the sample. In the next section, we describe how changes are introduced to the chosen parameters.

3.3 How much change should be introduced to the selected parameters to rectify the misclassification?

To evaluate a solution to the misclassification, it is crucial to modify the chosen parameters. Typically, during training, the loss is back-propagated to all parameters in the network. The gradient descent step defined in Eq 2.2 assumes a constant and small learning rate for all parameters. However, this approach can result in less beneficial and inadequate updates to the parameters, leading to inaccurate estimates of the parameters' potential to decrease the sample loss.

Next we determine estimates of parameters' importance. Such importance estimates are necessary for measuring individual parameters' value to the subset. In other words, it is necessary to understand if one parameter is better than the other, as it determines which of the two parameters brings the subset closer to the solution. For example, on updating the network N while changing only parameter w_a, the sample loss is given by $\mathcal{L}(y_k, N(\{w_a\}, s_i))$. On the other hand, on updating the network by changing only the parameter w_b, the sample loss is given by $\mathcal{L}(y_k, N(\{w_b\}, s_i))$. On comparing these separate sample losses, if $\mathcal{L}(y_k, N(\{w_a\}, s_i)) < \mathcal{L}(y_k, N(\{w_b\}, s_i))$, then the importance estimate of w_a will be higher than that of parameter w_b. Therefore, it is nec-

essary to have better estimates of the importance of each parameter. The estimate can be imperfect especially if the updates to parameters are made by only considering the gradient and not the curvature information. Therefore better estimates of the changes in loss can be obtained from second-order information. Therefore, the parameters undergo second-order updates (See Eq. 2.1).

Second-order updates are aggressive; updating parameters based on the loss calculated on a single sample alone s_i has the potential to disrupt previous learned representations. However, we use these updates to limit the number of parameter changes. In future, we determine the usefulness of both first-order and second-order updates. Therefore, our approach can be used on both first and second order update rules. We investigate in later sections which of the two is beneficial.

3.4 Parameter Subset Selection

Our proposed approach can be found in Algorithm 2. Over T iterations, an initially defined subset dynamically increases in size to include parameters necessary for correcting the network's decision for s_i. We start with an initial subset size n and sample the parameters using a prior gradient distribution $P_0(\mathbf{W})$, defined in Eq. 3.2. The n sampled parameters are stored in the initial subset $\mathbf{W}'_0$. These sampled parameters might not provide a solution to rectify the misclassification, and therefore our algorithm (PSS) continues to iteratively search for the appropriate parameters.

A pre-determined and static subset size n does not ensure a correct prediction of a misclassified sample. To avoid pre-determining the ideal subset size hyper-parameter n (n can be different for correctly predicting different misclassified samples), we introduce a dynamic subset size selection approach that attempts to find the minimum number of parameters to correctly classify s_i. At the start of every iteration, PSS samples from a pre-determined action space: 'Expand' or 'Replace'. 'Expand', as the name suggests, increases the subset size and samples x more parameters from the probability distribution

Algorithm 2 Pseudocode of the **P**arameter **S**ubset **S**election (**PSS**)

Input: s_i: Sample; n: Initial subset size; $P_0(\mathbf{W})$: Prior probability distribution; T: Total iterations

Output: $\mathbf{W}'_{\mathbf{final}}$: Final subset;

Select initial set $\mathbf{W}'_0 = \{w_j | w_j \sim P_0, \quad j \in \{1, 2, ..., n\}\}$
for $t \leftarrow 1$ **to** T **do**
  $\mathbf{C} \leftarrow$ Random choice between 'Replace' and 'Expand'
  if $\mathbf{C}$ = Expand **then**
    for $i \leftarrow 1$ **to** x **do**
      $\mathbf{W}'_t = \mathbf{W}'_{t-1} + \{w\}$, where $w \sim P_t(\mathbf{W})$
    end for
  end if
  if $\mathbf{C}$ = Replace **then**
    for $i \leftarrow 1$ **to** $\left|\mathbf{W}'_{t-1}\right|$ **do**
      $w \sim P_t(\mathbf{W})$
      if $\mathcal{V}(\mathbf{W}'_{t-1} \cup \{w\}) > \mathcal{V}(\mathbf{W}'_{t-1} \setminus (\mathbf{W}'_{t-1})_j)$ **then**
        $\mathbf{W}'_t = \mathbf{W}'_{t-1} \cup \{w\}$
      end if
    end for
  end if
end for
$\mathbf{W}'_{\mathbf{final}}$ = ApproximateIdealizedSubsetSelector($\mathbf{W}'_T$)
return $\mathbf{W}'_{\mathbf{final}}$

of the parameters at that iteration, $P_t(\mathbf{W})$. Here x is a hyperparameter that describes the number of extra parameters to be sampled into the subset. This is because at the end of all iterations, PSS re-verifies the contribution of each parameter to the prediction.

When the 'Replace' action is chosen, the number of parameters in the subset remains the same. However, given there are n parameters in $\mathbf{W}'_t$, n better parameters are chosen, i.e. after the action is taken, the new parameters (after second-order updates) result in increasing the fitness of the subset.

Ideally the parameters replaced in every iteration should be more *important* to solve the misclassification of s_i. The *importance* of a parameter in the subset can be determined by the difference in sample loss when the parameter is updated versus when it is not, given by $\mathcal{L}(y_i, N(\{w_a\}, s_i)) - \mathcal{L}(y_i, N(\{\}, s_i)$. However, evaluating this for each parameter individually, and then sampling the top-k *important* parameters requires millions of calculations based on the

parameter set. Instead we measure the fitness of the subset alone by a score function, $\mathcal{V} : \mathbf{W}'_t \longrightarrow \mathbb{R}$.

If the subset $\mathbf{W}'_t$ has n parameters, then only those n parameters undergo magnitude changes. After the parameters are updated appropriately, the modified network performs a forward pass on the sample s_i and recalculates the sample loss $\mathcal{L}(y_i, N(\mathbf{W}'_t, s_i))$. A lower sample loss indicates a healthier subset. The score function is given below in Eq. 3.4.

$$\mathcal{V}(\mathbf{W}'_t) = \mathcal{L}(y_k, N(\mathbf{W}'_t, s_i))^{-1}$$

During the 'Replace' action, each parameter in the subset is compared with a new sampled parameter from the corresponding probability distribution $P_t(\mathbf{W})$. If $\mathcal{V}(\mathbf{W}'_t \cup \{w_{new}\}) > \mathcal{V}(\mathbf{W}'_t \setminus \{w_{old}\})$, then w_{old} is replaced by w_{new}. Here w_{new} is a different parameter positioned at a different index from w_{old}. The probability distributing $P_t(\mathbf{W})$ is updated via:

$$P_t(\mathbf{W}) = \left\lceil max\left(0,\ P_{t-1}(\mathbf{W}) - \left|\frac{\partial \mathcal{L}(y_i, N(\mathbf{W}'_{t-1}, s_i))}{\partial w_j}\right|\right)\right\rceil_{j=0}^{d}$$

At the final iteration, PSS will output a parameter subset of a size smaller than the entire parameter space. Therefore, the 'Idealized Subset Selector' can be applied to the selected parameter subset to re-verify the contribution of each parameter. The reason for doing this is to get rid of all the parameters that do not contribute to correcting the sample accuracy. To avoid testing if the sample gets predicted with every combination of the parameters, the first part of PSS takes care of selecting parameters that reduce the sample loss, and the 'Idealized Subset Selector' searches for an even smaller subset that only satisfies the correct sample prediction constraint. Therefore, the 'Idealized Subset Selector' is executed only if the sample is correctly predicted in the first part.

3.4.1 Sensitivity to Hyper-parameters

The hyper-parameters to our approach are as follows: (1) n: the initial subset size, (2) T: total iterations, and (3) x: number of parameters to sample upon

choosing action 'Expand'. The choice of above mentioned hyper-parameters can have an effect on the final solution subset. For example, if the all of them are low in value, such as $\{n, T, x\} = \{5, 5, 5\}$, then a solution might not be obtained for the misclassification. This is especially true when sparsity is introduced in form of magnitude changes to the selected parameters. Additional sparsity in the number of parameters (by limiting x and n) can cause our approach to fail to correct the misclassifications.

Chapter 4

Experiments

In the last chapter, we introduced our approach: Budgeted Gradient Descent (BGD). In this chapter, we evaluate the usefulness and efficiency of our approach. First we introduce the two settings for conductive our experiments in Section 4.1. Then in Section 4.2, we discuss the architectures and dataset we apply BGD on, and define the metrics that evaluate BGD. In the following sections, we record the results for five different experiments, and note our inferences from the individual experiments. The experiments are enumerated below:

- Experiment 1: We compare the second-order variant of our approach, 'BGD', with baseline approaches used alternatively in the literature.
- Experiment 2: We introduce a first-order variant of our approach, 'BGD-I', and explore its benefits on the network's generalizability.
- Experiment 3: We introduce a variant of BGD called 'BGD-C' to investigate if avoiding high gradient parameters, where the gradients are computed on the entire training set, leads to avoiding a loss in generalizability.
- Experiment 4: We investigate the loss of generalizability in networks in a *non-reloading* setting and compare our approach and its variants with other baseline approaches.

- Experiment 5: We investigate the sensitivities and robustness of our approach.

4.1 Loss in Generalizability

Generalizability is a network's ability to generalize to unseen data. Any changes to the network should not cause detriment in its generalizability. However, a network pre-trained on one distribution may lose its past training performance if re-trained on a different distribution. To learn from misclassified examples during testing, the network retrains on samples that are potentially outliers to the training distribution. Therefore, to properly assess the network's generalizability, it is important to evaluate its performance not only on the testing data but also on the original training data. The latter helps us understand how much of the network's past learning is retained.

While the loss in generalizability might be trivial when learning from one single example, this loss could accumulate as the network is forced to learn from multiple misclassified samples. To measure the impact on generalizability in both scenarios, we introduce two experimental settings:

1. *Reloading* setting: In the *reloading* setting, the initial network, N_0, is reset before each misclassified sample is learned, meaning that changes made for the sample s_i are not retained when learning the sample s_{i+1}. This allows us to test whether our method is effective regardless of the difficulty of the sample.

2. *Non-reloading* setting: In the *non-reloading* setting, we assume the modified network N_i after learning sample s_i becomes the starting point for learning sample s_{i+1}. This setup is used to investigate the long-term effects of cumulative changes on the network's generalizability.

In the first five experiments, we focus solely on the reloading setting to avoid redundancy. Once we have introduced all variants of our approach, we then explore how the network's ability to retain generalizability is affected when multiple changes are made.

4.2 Networks and Metrics

We test our approach for two pre-trained networks: (A) AlexNet [20], and (B) a Multilayer Perceptron (MLP). The MLP consists of three fully connected layers, each followed by a Rectified Linear Unit (ReLU) activation function. Both networks are first trained on a relatively small number of epochs, to increase the number of misclassifications. By introducing misclassifications we hope to emulate a real-life scenario where a deployed model misclassifies certain target samples. The AlexNet is trained for 40 epochs and has 370 misclassified samples. The MLP is trained for 10 epochs on the MNIST dataset and has 947 misclassifications in total.

We evaluate our approach against baselines using the following metrics:

1. **P**ercentage **O**f **C**orrections (POC): POC, as defined in Eq 1, represents the percentage of the total incorrectly predicted samples that were correctly predicted after applying an approach. As our primary focus is to correct misclassifications, each approach is evaluated on POC, where a higher POC value indicates a better approach. However, POC alone is not a sufficient measure of an approach, as it may not account for issues like sample memorization, where the network learns specific samples without generalizing to other training or testing examples. It is important to note that a POC of 100% does not imply that the network correctly predicts 100% of testing samples; it only means that the approach successfully modified the network N_0 independently to correctly predict all initially misclassified samples. The POC for the reloading setting is shown below, and corresponds to the entire set of misclassified samples.

$$\textbf{POC} = \frac{\text{Correct}}{\text{Incorrect } + \text{Correct}} * 100\%$$

2. **P**ercentage **O**f **P**arameters **A**ltered (POPA): POPA represents the fraction of parameters altered by an approach to achieve correct classification. As we aim to reduce the number of parameters altered, we prefer an approach with a smaller POPA value. This metric also provides insights

into how generalizability is affected by parameter changes. For example, it helps answer questions like whether altering a smaller percentage of parameters reduces the likelihood of memorizing difficult samples more effectively than altering a larger percentage. Our POPA metric is defined by the Eq 2, where $\mathbf{W}'^{(i)}_{\mathbf{final}}$ is the final parameter subset, containing parameters to change for the correct prediction of sample s_i, and the total number of network parameters is $|\mathbf{W}|$.

$$\mathbf{POPA} = \frac{\left|\mathbf{W}'^{(i)}_{\mathbf{final}}\right|}{|\mathbf{W}|} * 100\%$$

3. Change in training Loss (ΔTL): ΔTL quantifies the loss in learned representations by measuring the change in training loss after an approach has been applied. Although our approach is training data-free, we use this metric to evaluate different baseline approaches. If the training loss for the initial network, N_0, is denoted by $\mathcal{L}_0(D_y, N_0(\mathbf{W}_0, D_x))$, and the training loss, after the network N_0 gets modified for sample s_i, is:

$$\Delta TL_i = \mathcal{L}_i(D_y, N'(\mathbf{W}'^{(i)}_{\mathbf{final}}, D_x)) - \mathcal{L}_0(D_y, N_0(\mathbf{W}_0, D_x))$$

4. Change in testing accuracy (ΔTA): ΔTA is calculated similarly to ΔTL. A higher decrease in testing accuracy (TA) suggests a sharper decline in the network's generalizability. ΔTA is given by the following equation, where TA_0 is the training accuracy of the original network.

$$\Delta TA_i = TA_i - TA_0$$

The set of misclassified examples consists of those initially incorrectly predicted by the network N_0. The metrics POPA, ΔTL, and ΔTA are averaged over all misclassified samples in the results, while POC does not require averaging as it indicates the percentage of samples correctly predicted by an approach out of the entire set.

4.3 Baselines

Below is the list of baselines we consider for understanding the efficiency of our algorithm:

1. Re-train: In this baseline, all the parameters of the network are retrained via backpropagation. This is identical to the full-network fine-tuning case, and we include it as it is the most common approach for transfer learning [17]. We expect an increase in sample memorization due to the greater sample difficulty, resulting in an increase in ΔTL and ΔTA.

2. Fine-tune: In this baseline, only parameters in the penultimate layer of the network are altered. Since retraining all parameters may disrupt learned features, fine-tuning only updates the penultimate layer of the networks. Therefore, we expect less sample memorization than in the case of re-training.

3. L1-Regularizer: In this baseline, a penalty term, comprising of the parameter magnitudes is added to the loss function. The L1-Regularizer is a traditional sparse regularizer typically used for feature selection.

4. Top-k: This approach was introduced by [33], as a form of sparsified gradients. It can also be expressed as a form of pruning $|\mathbf{W}| - k$ parameters ($|\mathbf{W}|$ being the total parameters), i.e., performing a gradient update on only k network parameters. Although most pruning algorithms use iterations to optimize the unpruned parameters, in this baseline, we refrain from doing so as we already have a pretrained network. Sun et al. in [33] propose 'meProp' that alters only 1–4% of the total parameter space during training. However, we only focus on correcting the misclassifications; therefore, we make minor adjustments to the approach to suit our problem.

   Given the pre-trained network N_0, we calculate the parameters' gradients on the sample (s_i) loss, and do a first-order gradient update step for k parameters. This step is kept the same as 'meProp'. However,

the authors of 'meProp' update only 1–4% of the total parameter space. We, on the other hand, update p% of the network parameters, where $p \in (0, 100]$. In other words, we keep increasing the number of parameters altered until the sample s_i is correctly predicted. This modification to the algorithm serves two purposes: (a) it tracks the metric POPA, and (b) it ensures the sample is correctly predicted. Ensuring the sample is correctly predicted (i) amplifies the usefulness of 'meProp' in our problem setting, (ii) provides other pathways to correct predictions, and (iii) justifies the stochasticity in our approach.

4.4 Experiment 1: Updating high gradient parameters with aggressive updates

In this experiment, we investigate how our approach performs relative to relevant baselines.

In this section, we compare different baselines with our approach, BGD.

Table 4.1, 4.2 and Figure 4.1 show the results for the *reloading setting*, i.e., the initial network is identical for all misclassified samples. In the listed tables, we compare BGD with aforementioned baselines and record values for metrics defined in Section 4.2. Observations are recorded after averaging results from experiments done using three distinct seeds. While rows in the tables record their corresponding standard deviations, the missing deviations from the metric 'POPA' for the baselines, re-training and fine-tuning, are because the number of parameters changed are pre-determined and therefore fixed. For re-training, as we alter all network parameters, 'POPA' is fixed at 100%. On the other hand, we only fine-tune the penultimate layer of each network, therefore 'POPA' is fixed at 17.93 for the AlexNet network and at 99.85 for the MLP network.

The *reloading setting* primarily examines if each approach is able to correctly classify a sample, regardless of its difficulty. Both BGD and the L1-Regularizer successfully classify 100% of misclassified samples in every seed. This is followed by the Top-k approach. Compared to the Top-k approach,

BGD alters a smaller fraction of the network parameters (denoted by the metrics 'POPA'). Although both are guided by the gradient magnitudes, BGD finds a smaller solution set for the misclassifications. Moreover, our approach is able to find the smallest number of parameters to change, relative to the chosen baselines, for solving the misclassifications.

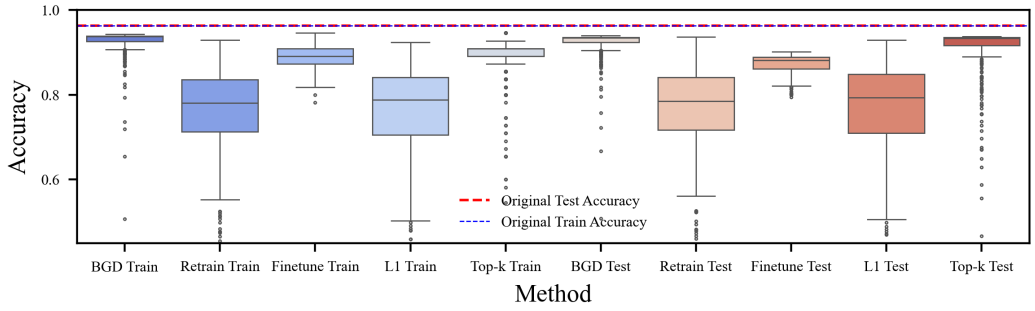
For both the networks, we observe a negative (average) change in training loss. While the other baselines increase the training loss value to correct the misclassification, our approach reduces the loss compared to the original network. However, the decrease in training loss does not correlate with an increase in testing accuracy. On the contrary, our approach decreases the testing accuracy by an order of 10^{-2} for both the networks.

The decrease in testing accuracies suggests overfitting to the training examples. However, our approach assumes the unavailability of training examples, and fine-tunes the network based on the misclassified sample alone. As this sample is not part of the training set, the decrease in training loss can be attributed to the network learning features from the misclassified sample, that are similar to those in the majority of the training examples. Therefore, BGD relies on these familiar features to correct the classification. The observed decrease in training loss indicates that unique features of the misclassified sample are not being learned; instead, the network relies on features already learned during training. Consequently, sample memorization is avoided.

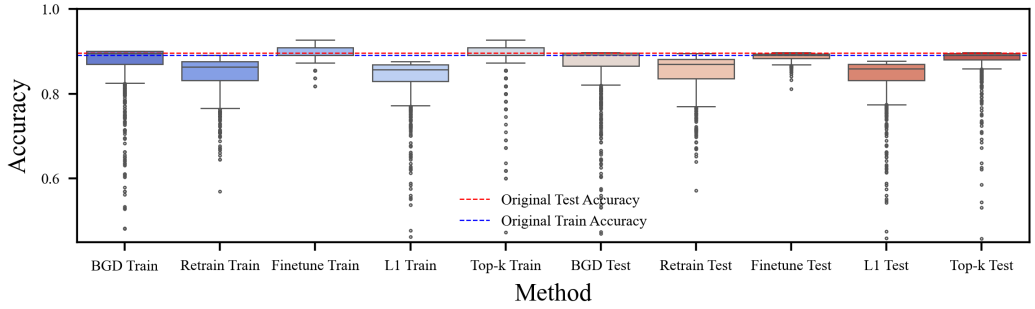
Along with the drop in training loss, we also observe a decrease in the training accuracy (as shown in Figure 4.1). Although this drop is in order of 10^{-2}, it may result from the network's overemphasis on specific features rather than distributing focus across diverse features (learned during training). This overemphasis allows the network to correctly predict the misclassified sample.

As the number of parameters altered in BGD is relatively lower, this overemphasis on features, relevant to the misclassified sample, only pertains to a few features. In contrast, approaches that modify a larger fraction of parameters, such as re-training and fine-tuning, increase the training loss, leading to an expected decrease in training accuracy as well as the testing accuracy. However, the average change in training/testing accuracies for fine-tuning are

lower than those observed for BGD. A larger ΔTA in BGD can be attributed to the presence of outliers, as observed in Figure 4.1. We note that our claims only extend to the models we test with, however we would anticipate generalization to similar models.



(a) AlexNet



(b) MLP

Figure 4.1: Comparing testing and training accuracies of BGD with baselines, namely, 'Re-training', 'Fine-tuning', 'Top-k', 'L1-Regularizer'. The highlighted entries in each row signify the best observation. The box plots provide a distribution of the training/testing accuracies after each misclassified sample is correctly predicted. The original accuracies have been dotted.

4.5 Experiment 2: Dependency on the Magnitude of Change

Misclassified samples have more associated loss values, and therefore, re-training the network on those particular samples impact the network negatively (as seen in Section 4.3). In our approach we perform second-order updates to selected parameters to avoid increasing the number of changes to the network. However, these updates are based on the gradients calculated on a single sample.

	BGD	**Re-training**	**Fine-tuning**	**Top-k**	**L1-Regularizer**
POC	**100** ± 0	93.33 ± 2.1	70 ± 12.1	98.65 ± 0.17	**100** ± 0
POPA	**1.9e-05 ± 1.5e-05**	100	17.93	5.8e-03 ± 0.01	100 ± 0
Δ TL	**-3.3e-03 ± 2.5e-04**	9.1e-03 ± 7.7e-03	2.4e-03 ± 1.3e-03	3.7e-03 ± 8.5e-3	5e-3 ± 4.3e-03
Δ TA	0.09 ± 0.07	0.21 ± 0.13	0.07 ± 0.02	**0.06 ± 0.07**	0.12 ± 0.05

Table 4.1: Comparing BGD to the baselines: 'Re-training', 'Fine-tuning', 'Top-k', 'L1-Regularizer' for the network AlexNet. The highlighted entries in each row signify the best observation. The box plots provide a distribution of the training/testing accuracies after each misclassified sample is correctly predicted. The original accuracies have been dotted.

	BGD	**Re-training**	**Fine-tuning**	**Top-k**	**L1-Regularizer**
POC	**100 ± 0**	100 ± 0.01	99.67 ± 0.02	99.47 ± 0.10	**100 ± 0**
POPA	**0.02 ± 0.02**	100	99.85	3.52 ± 4.88	100 ± 0
Δ TL	**-6.6e-03 ± 2.5e-04**	1e-03 ± 1.9e-03	1.3e-03 ± 4e-04	2.1e-03 \pm 6.2e-03	1.7e-3 ± 4.3e-03
Δ TA	0.03 ± 0.07	0.04 ± 0.04	**0.009 ± 0.01**	0.02 ± 0.05	0.06 ± 0.06

Table 4.2: Comparing BGD to the baselines: 'Re-training', 'Fine-tuning', 'Top-k', 'L1-Regularizer' for the network MLP. The highlighted entries in each row signify the best observation.

Therefore, these updates should be limited. This limit can be applied to both the number of parameters updated, and the magnitude of changes introduced to each parameter. The magnitudes parameter changes can be restricted, as in the case of first-order updates. On restricting the magnitude of change, as we observed in the Top-k approach, that more parameters need to be changed. On the other hand, BGD (the second-order variant is called BGD-II) makes aggressive second-order updates to a smaller number of parameters. To investigate which option reduces generalization loss in networks, we investigate the following question: Should we update a large number of parameters by a small magnitude, or should we update a smaller number of parameters more aggressively?

Although the Top-k approach makes first-order changes to parameters until the misclassification is solved, the number of changes are relatively higher. We introduce a variant of our algorithm that makes more precise changes to selected parameters, while also aiming to minimizing the number of such changes.

	BGD-II	**BGD-I**	**Top-k**
POC	**100 ± 0**	61.35 ± 0.12	95.1 ± 7.1e-03
POPA	**1.1e-05 ± 1.3e-05**	1.1e-04± 1.7e-04	0.01 ± 0.01
Δ TL	-3.3e-03 ± 1.8e-05	**-3.4e-3 ± 1.6e-05**	7.1e-03 ± 0.01
Δ TA	0.03 ± 0.01	**0.03 ± 4.1e-03**	0.091 ± 0.10

Table 4.3: Comparing the two variants BGD-I and BGD-II for the network AlexNet.

	BGD-II	**BGD-I**	**Top-k**
POC	**100 ± 0**	95.88 ± 0.05	99.77 ± 1.4e0-5
POPA	**0.02 ± 0.02**	0.04± 0.07	3.23 ± 4.57
Δ TL	-6.7e-3 ± 2.5e-05	**-6.7e-3 ± 1.6e-05**	2e-03 ± 6.2e-03
Δ TA	0.03 ± 0.06	**0.02 ± 0.03**	0.02 ± 0.05

Table 4.4: Comparing the two variants BGD-I and BGD-II for the network MLP.

4.5.1 BGD-I: First order variant of BGD

The structure of our optimization algorithm (see 2) remains the same. The only change is made to how the chosen parameters are updated. Recall that the score function calculates the inverse of the sample loss, $\mathcal{L}(y_i, (N(s_i, \mathbf{W}'_t))$. To calculate the loss for s_i, second-order updates are made to the network parameters in the subset $\mathbf{W}'_t = [w_i, w_2, ..., w_n]$. However, to prevent aggressive updates, we alter our selected parameters by a fraction of the calculated gradient. Specifically, the parameters change by a gradient descent step as defined in Eq. **??**. This change in the algorithm is expected to result in an increase in the number of changed parameters in order to correctly classify misclassified samples.

The results to different metrics defined in Section 4.2 can be found in Tables 4.3 and 4.4. We find that this variant of our approach only corrects 61.35% of all misclassifications in AlexNet, and 95.88% of them in MLP. The disparity between the number of correct predictions in BGD-I and Top-k suggest that more changed parameters were required to solve the misclassifications, as both of them are guided by the gradients calculated on the samples. Recall that

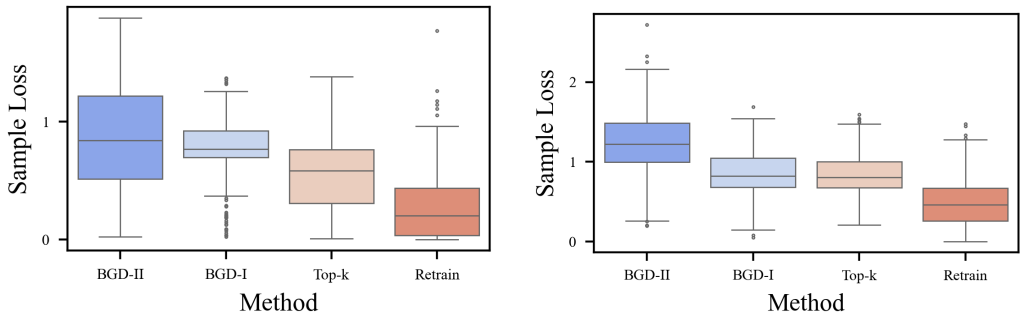
in each iteration of our algorithm, we can dynamically increase the parameter subset. We fix the hyper-parameter that decides the number of changed parameters to be added to the parameter subset to be 10 for this and all future experiments. Therefore, either the number of iterations of our approach needs to be higher, or the number of changed parameters by which the subset expands needs to be increased to find the solutions to all the misclassified samples. We test the rest of the metrics for all approaches, namely, BGD and Top-k for only those 61.35% and 95.88% of the misclassified samples in Table 4.3. Therefore, the statistics mentioned for Top-k and BGD (apart from POC) is for the samples correctly predicted by Top-k out of those 61.35%. By doing so, we better understand the contribution of each approach.

We find that for all the samples that were correctly predicted by all three approaches, BGD-I has the least decrease in testing accuracy. This is also accompanied by the highest drop in training accuracy, which suggests that the parameters retain a fraction of the learned representation, thereby losing fewer testing samples to misclassifications. The Top-k approach operates similarly, with the only difference being that it changes more parameters. Therefore, it can be concluded that even though the magnitude of change is smaller, as the number of changed parameters increases, the network starts degrading.

On the other hand, second-order updates in BGD-II has comparable performance with BGD-I and Top-k. (We call the second-order variant of BGD as BGD-II and not BGD to enhance clarity.) Additionally, the change in testing accuracy is similar in all three approaches, with the only difference between Top-k and BGD-I, BGD-II being the difference in training loss. While training loss increases in Top-k, it decreases in the other two approaches, which suggests that the only outcome of making fewer changes is a drop in training loss. Additionally, this drop in training loss can be attributed to a strong reliance on the training features in BGD-II and BGD-I, as opposed to overwriting the features with the sample's unique features, as in Top-k.

To better support this claim, we visualize the sample losses, after the sample was correctly predicted. A lower sample loss is associated with an increase in sample overfitting. To understand the trend of how updating more pa-

rameters leads to lower sample loss, we also plot the sample losses for the Re-training baseline in Figure 4.2. For both the architectures, BGD-II has the least sample loss, suggesting that the network is not overfitting on that particular sample. On the opposite end is Re-training, exhibiting behaviour we want to avoid. For example, for the AlexNet network, Figure X shows that the Top-k has sample losses close to 0, which is supported by Table 4.3 showing the highest increase in testing accuracy. BGD-I strikes a balance between the two, Top-k, and BGD-II, and therefore has the least increase in testing accuracy.



(a) AlexNet (b) MLP

Figure 4.2: Sample losses after the sample is correctly predicted.

4.6 Experiment 3: Avoiding sensitive parameters to the overall training set

In this section, we first identify disadvantages to our approach, and then provide a work-around. We run through some experiments to identify the benefits of said work-around.

In Section 4.1, we discuss that an approach is beneficial when the network's generalizability is preserved, along with its ability to learn a new sample. In our approach, we minimize the number of updated parameters to preserve said generalizability, under the assumption that training data is unavailable at the time of application. However, to reduce the number of parameters altered, we see that the amount of updates to those parameters need to be more aggressive.

As the updates are based on the gradients calculated on the single misclassified sample, it can potentially harm the network's generalizability.

To bypass this outcome, in this section, we introduce a second variant of our approach. In this variant, we still assume a lack of the training data. However, to have an estimate of the overall loss landscape, we assume the availability of the expected parameter gradients for the training data. Empirically, this gradient vector calculated on the training data, $\bar{\mathbb{G}}$ is defined as follows:

$$\bar{g}_i = \frac{1}{|D|} \sum_{j}^{|D|} \frac{\partial \mathcal{L}(D_y, (N(D_x, \mathbf{W}))}{\partial w_i}, \quad \forall \bar{g}_i \in \bar{\mathbb{G}}$$

We hypothesize that the impact to the network's generalizability can be reduced if we can avoid highly sensitive parameters to the training loss. To test this hypothesis, we use, $\bar{\mathbb{G}}$ given in Eq 4.6, to estimate the loss landscape. As Chen et al. in [10], conclude that pruning the most sensitive parameters improves generalization, we aim to update the parameters with not only higher sensitivity to the sample loss, but also the ones with lower sensitivity to the training loss.

First, we estimate the curvature of the training loss from the gradient vector $\bar{\mathbb{G}}$ in Subsection 4.6.1, because we aim to avoid updating parameters with higher training loss curvatures. Once we have these estimates, we determine a threshold for the curvature. If a parameter has a curvature steeper than the determined threshold, we avoid updating the parameter. We do this because updating parameters with a higher training loss curvature has the potential to disrupt learned representations.

4.6.1 Estimating the curvature for the training loss

We use Taylor's expansion to estimate the curvature of the training loss landscape. Updating parameter w_j results in changes to the loss function, denoted by $\Delta\mathcal{L}_{w_j}$, defined in Eq. 4.6.1. Here $\mathbf{W}'$ represents the updated weight vector where the j-th element is $w_j + \Delta w_j$.

$$\Delta\mathcal{L}_{w_j} = \mathcal{L}(D_y, (N(D_x, \mathbf{W}')) - \mathcal{L}(D_y, (N(D_x, \mathbf{W}))$$
$$\simeq \Delta w_j \left(\frac{\partial \mathcal{L}(D_y, (N(D_x, \mathbf{W}))}{\partial w_j} \right) = \Delta w_j \circ \bar{\mathbb{G}}$$

The two factors estimating $\Delta\mathcal{L}_{w_j}$ are (a) changes in parameter magnitude, and (b) the parameter's gradient calculated on the training loss, given by $\bar{\mathbb{G}}$. Since, we assume that $\bar{\mathbb{G}}$ is already available to us, the only factor we need to determine is Δw_j. Recall, in our approach, to limit the number of changed parameters, we provide second order updates to the parameters, given by: $\Delta w_j \simeq -\mathrm{H}^{-1}\nabla_{w_j}\mathcal{L}(\mathbf{W}, N(s_i))$. Therefore, the curvature can be estimated by the following:

$$\Delta\mathcal{L}_{\mathbf{W}} = -\mathrm{H}^{-1}\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W}, N(s_i)) \circ \bar{\mathbb{G}}$$

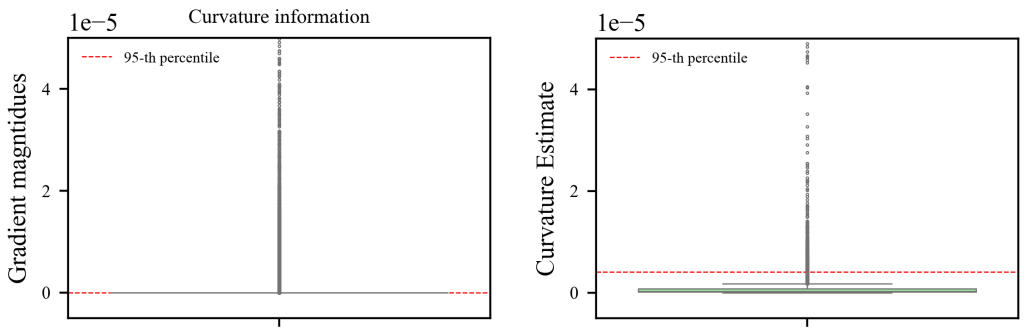
The change in the loss value should not be detrimental so that the past learned features from the training data are preserved. Therefore, it is necessary to establish a threshold, τ, to identify changes that are detrimental to the network. We define a parameter, w_j, as a *harmful* parameter if $\Delta\mathcal{L}_{w_j} > \tau$. In other words, if second-order updates to w_j, result in an increased loss value greater than τ, we do not change the parameter at all. In the following Section 4.6.2, we elaborate how we determined the threshold τ.

4.6.2 Determining the threshold τ

For second-order updates to parameters, we use the Empirical Fisher matrix (See Section 2.1) in place of the Hessian to compute $\Delta\mathcal{L}_{w_j}$.

$$\Delta\mathcal{L}_{\mathbf{W}} = -\frac{\nabla_{\mathbf{W}}\mathcal{L}}{(\nabla_{\mathbf{W}}\mathcal{L})\ (\nabla_{\mathbf{W}}\mathcal{L})^T} \circ \bar{\mathbb{G}}$$

Before determining the threshold, we visualize $\Delta\mathcal{L}_{\mathbf{W}}$ in Figure 4.3. For both networks, the curvature information is very low, therefore, we select the curvature estimate threshold based on a high percentile of gradient magnitude. For our experiments, we set the value of τ to be 95 by visual analysis, i.e., if the curvature estimate is above the 95^{th} percentile, then we resample to select a different parameter.



(a) AlexNet (b) MLP

Figure 4.3: $\Delta\mathcal{L}_{\mathbf{W}}$ for both networks.

4.6.3 Results

First, we analyze whether a solution to misclassifications can be achieved by avoiding parameters with a high curvature estimate. Next, we investigate if this approach can increase/preserve better generalizability in the networks. We name this variant of our approach 'BGD-C' (C for curvature) to separate it from the previous variants.

The results from the previously defined metrics can be found in Table 4.5 and 4.6, and Figure 4.4. BGD and BGD-C appear to be minutely different since BGD-C selects different parameters only if the previously sampled parameters had a training loss curvature estimate greater than the 95-th percentile. Among the important differences is the observation of BGD-C for the MLP network not being able to correctly classify all of the misclassified samples. This suggests that for 2.01% of the misclassified samples, BGD-C was not able to find an alternative pathway to their correct predictions. In other words, the parameters selected by BGD had a higher curvature, that could potentially harm the training performance. Since the number of parameters with high training loss curvature are relatively small, there is little chance for BGD-C to replace them with lower training loss curvature parameters.

Figure 4.4 provides deeper insights into the distribution of training and testing accuracies. For the AlexNet network, BGD-C finds low curvature parameters; however, this process increases the number of changed parameters, thus reducing the training/ testing accuracies. The change in these accuracies

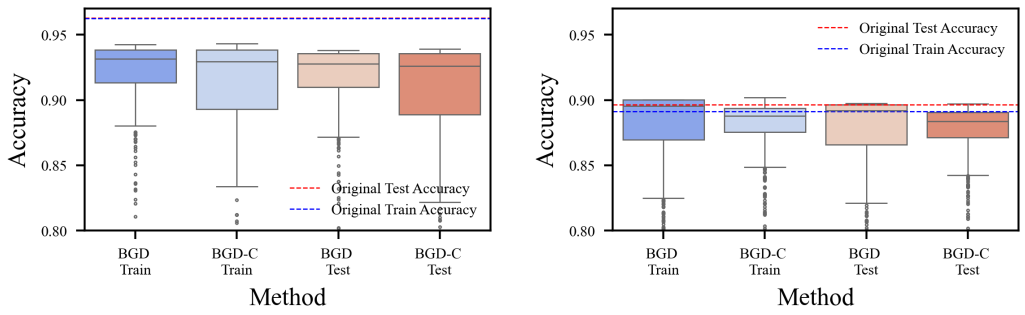
	BGD	BGD-C
POC	**100 ± 0**	**100 ± 0**
POPA	**1.9e-05 ± 1.5e-05**	2.1e-05 ± 1.5e-05
Δ TL	**-3.3e-03 ± 2.5e-04**	-2.5e-3 ± 0.01
Δ TA	**0.09 ± 0.07**	0.09 ± 0.14

Table 4.5: Comparing BGD to the new variant, BGD-C for the AlexNet network. The highlighted entries in each row signify the best observation.

	BGD	BGD-C
POC	**100 ± 0**	97.99 ± 1.3
POPA	**0.02 ± 0.02**	0.09 ± 0.13
Δ TL	-6.6e-03 ± 2.5e-04	**-6.7e-03 ± 2.5e-03**
Δ TA	0.03 ± 0.07	**0.02 ± 0.02**

Table 4.6: Comparing BGD to the new variant, BGD-C for the MLP network. The highlighted entries in each row signify the best observation.

is also influenced by the specific sample and the locations of high gradient parameters (gradients computed on the sample). If a parameter has a high training loss curvature estimate, other parameters are sampled instead. Our analysis indicates that avoiding certain high gradient parameters may require modifying more parameters to correctly classify the sample.



(a) AlexNet (b) MLP

Figure 4.4: Comparing training and testing accuracies for BGD, and BGD-C for AlexNet and MLP networks.

4.7 Experiment 4: The *Non-reloading* Setting

In this experiment, we analyze the effect each approach has on the generalizability of both networks, given the *non-reloading* setting. Recall, that in this setting, misclassified samples are learned one after the other, sequentially. Therefore, although the initial network is the same, N_0, the modified network N_i after learning s_i, is the input network for learning sample s_{i+1}. This is different from the reloading setting, where N_0 is the input network for all samples, including s_{i+1}.

In order to reduce redundancy, we compare all versions of our approach with baseline approaches introduced in Section 4.3, namely, 'Re-training', 'Partial fine-tuning', 'Top-k', and 'L1-Regularization'. The versions of our approach to be compared with the said baselines are 'BGD-II', 'BGD-I', and 'BGD-C'.

The metrics used to compare the approaches are the same as before, as defined in Section 4.2. In the *non-reloading* setting, POC still represents the fraction of learned samples. However, the network, N_i after learning sample s_i may or may not remember sample s_{j-i}. Therefore, even though N_{i-1} is able to correctly predict s_{i-1}, the modified network N_i may or may not correctly predict s_{i-1}. Instead of recording the metrics for the final network N_m (m is the total number of misclassified samples), we record the metrics as an average over the statistics produced by each network N_i. The metrics, POC and POPA are redefined as:

$$\textbf{POC} = \left(\frac{1}{m} \sum_{i=1}^{m} \text{Correct}_{N_i} \right) * 100\%$$

$$\textbf{POPA} = \left(\frac{1}{|\mathbf{W}|} \sum_{i=1}^{m} \mathbf{W}'^{(i)}_{\textbf{final}} \right) * 100\%$$

Here, Correct_{N_i} is 1 if the network N_i correctly predicts s_i, otherwise it is 0. $\mathbf{W}'^{(i)}_{\textbf{final}}$ is the final subset containing parameters that correctly predict s_i. If network N_i is unable to correctly predict s_i, then the subset is empty. The total number of network parameters is $|\mathbf{W}|$. The metric values for this experiment are recorded in Tables 4.7 and 4.8. Figures 4.5, 4.6, 4.7 show the results for

the AlexNet network. Figures 4.8, 4.9, and 4.10 show the results for the MLP network. The values have been recorded after running the experiment for three distinct seeds and then averaging the statistics.

Instead of recording the changes in training loss and testing accuracy in the table, we directly visualize these metrics in the aforementioned figures. The figures show the results over training accuracy, test accuracy, and training loss, respectively. As the samples are incrementally learned, the network loses its past representations. We plot the network behaviour for the first 100 misclassified samples as a subfigure to get a closer look at what happens before the network forgets catastrophically. In said experiments, the order of the samples was kept consistent, meaning that the sample at the i-th iteration is the same across all seeds for a network. Since our primary objective is to evaluate how comparatively well each approach preserves generalizability, we did not experiment with altering the order of the samples.

	BGD-II	**BGD-C**	**BGD-I**	**Re-training**	**Fine-tuning**	**Top-k**	**L1-Regularizer**
POC	**88.91 ± 0.12**	54.59 ± 10.33	67.29 ± 1.7	27.43 ± 1.66	44.59 ± 1.33	84.32 ± 3.77	46.39 ± 0.21
POPA	1e-04 ± 1.3e-04	**6.19e-05 ± 1.4e-04**	8.5 e-05 ± 1.6e-04	100	17.93	5.9e-03 ± 0.01	100 ± 1.7

Table 4.7: Comparing BGD and its variants to the baseline approaches for the AlexNet network.

	BGD-II	**BGD-C**	**BGD-I**	**Re-training**	**Fine-tuning**	**Top-k**	**L1-Regularizer**
POC	25.87 ± 0	66.10 ± 0	77.64 ± 2.03	99.69± 0.01	25.87 ± 0	**77.71 ± 0**	67.68 ± 0
POPA	**0.02 ± 0.11**	0.11 ± 0.19	0.04 ± 0.09	100	99.85	12.07 ± 23.7	100 ± 0

Table 4.8: Comparing BGD and its variants to the baseline approaches for the MLP network.

Reading the figures: the x-axis represents the iterations; in each iteration, one misclassified sample, s_i, is learned. Hence, the modified network, N_i, becomes the input network for the subsequent iteration or sample (in this context, these terms are interchangeable). The AlexNet network has 370 misclassified samples, and the MLP network has 947 misclassifications in total. As the iterations progress, N_i learns each sample, and incrementally diverges from the original network. The red dots along the lines correspond to incorrect classification of a sample. If a misclassification has occurred, the network

does not change, and therefore, horizontal lines can be seen in parts of the graphs. This process tests the long-term impact of each approach on the network's generalizability. At iteration i, all approaches modify the network by attempting to learn the same sample s_i. Therefore, the dips in the line graphs (for testing/training accuracies) that happen at the same iteration, suggesting that learning a particular sample s_i may be difficult, causing the network to forget some training representations. By analyzing the statistics, we draw conclusions about each approach's effects on network generalizability. We divide the observations and inferences into their own subgroups, and discuss them for each of the baselines and variants. We discuss these in an order to allow for comparative analysis rather than from best performing to least performing approach, or other similar orderings.

BGD-II. Recall that BGD-II makes second-order updates to fewer network parameters. This is supported by the observation that BGD-II has the least POPA metric value among all other approaches. The network iteratively learns from misclassified samples by encouraging aggressive sparsity in BGD-II. This results in only a small number of parameters encoding the features learned from the misclassified sample. For the AlexNet network, which has a larger architecture, BGD-II initially performs well, maintaining most of the training and testing performance for the first 100 misclassified samples. However, as the number of changes accumulates, a drastic increase in training loss can be observed, particularly compared to other baselines. Despite the increase in training loss, the training and testing accuracies do not decline as significantly as in other baselines, indicating that the network is making substantial errors on only a part of the training data. This outcome is intuitively understandable: BGD-II constrains updates to a few parameters, but those parameters are updated aggressively, leading to overfitting on the features of the misclassified samples. Since these samples are likely outliers relative to the training data, the newly learned features do not generalize well. As a result, part of the network overfits on these samples, driving up the training loss, while the rest of the network retains generalizable features, allowing it to correctly predict a

portion of the training and testing sets. These observations indicate that while updating a minimal set of parameters may initially help correct misclassified samples, it can ultimately harm the network's generalizability. As the network is exposed to more samples that are potential outliers, the reliance on a few parameters to correctly predict the sample increases, leading to a reduction in overall generalizability. Similarly, in the MLP network, which has fewer parameters than AlexNet, this degradation occurs earlier in the iterations.

BGD-C. BGD-C was introduced to tackle the above disadvantage of BGD-II. As the disadvantage arises due to the aggressive updates to parameters, such updates would be avoided in BGD-C if the training sensitivities of those parameters were higher as well. For both networks, we observe in Figures 4.7 and 4.10, the detrimental increase in training loss by the BGD-II approach could be successfully avoided by BGD-C. We additionally observe some opposite effects in the two networks. On comparing BGD-II and BGD-C, we observe that when applying BGD-C on the MLP network that the number of changed parameters (value for metric POPA), training and testing accuracies, and the number of corrections (value for metric POC) are higher than those for BGD-II. We had observed previously that MLP has fewer parameters than AlexNet, and therefore, changing even a small number of parameters aggressively degraded the performance of most training and testing examples. But this degradation can be avoided if more parameters that have low training loss sensitivities are updated instead. It also suggests that the training loss landscape of the MLP network is mostly flat. This claim is supported by the higher number of parameters updated without loss in training performance. The opposite effect is observed for the AlexNet network. POPA, POC, training and testing accuracies drop when BGD-C is applied compared to when BGD-II is applied. Although the training loss decreases, the training and testing accuracies also decrease, suggesting that by avoiding sensitive parameters, we do not guarantee an increase in training and testing accuracies. This was also observed in Experiment 1, where we could not find any relationship between decreasing training loss and increasing testing accuracy. Additionally,

we observe that by avoiding sensitive parameters, BGD-C finds fewer solutions for misclassifications than BGD-II. As fewer samples were correctly predicted, fewer changes were made to the network.

BGD-I. We find that for both architectures, variant 'BGD-I' outperforms all other variants and baselines. Recall that 'BGD-I' provides first order updates to selective parameters to correct misclassifications. The magnitude changes to parameters in 'BGD-I' were comparatively smaller than that in 'BGD-II' and therefore, we had observed that more parameters needed to be updated to learn the misclassified samples (see Sec 4.5). We had similar findings in these experiments. This suggests that the loss in generalizability can be attributed to the amount of change rather than sparsity. Both BGD-II and BGD-I update only a subset of the parameters that correspond to correcting misclassifications, and thereby they both aim to modify the network sparsely. BGD-II may lose generalizability more quickly than BGD-I. This also suggests that to preserve generalizability, one should not aim to limit the number of changed parameters strongly by increasing the magnitude of updates. Rather, one should aim to sparsify updates by obtaining a balance between the number of parameters to update, and the magnitude of updates to them. However, this conclusion is applicable only if the updates are based on the gradients calculated on the misclassified sample alone.

L1-Regularizer. We observe that the L1-Regularizer correctly predicts approximately 46.39% of misclassified samples in the AlexNet network and 67.68% of them in MLP network. However, we also observe that it produces low training and testing accuracies in both architectures, while having comparable training loss with other baselines. Although the training loss increases from the initial model, it remains stable after some number of iterations. The increased loss and decreased training and testing accuracies suggest that the L1-regularier overfits on the misclassified samples. The stability in the training loss across samples might suggest that the network makes small errors for most of the training data. Recall that the formulation of the L1-Regularizer

forces the network to make small updates to its parameters. Therefore, although most of the parameters are changed (See metric POPA in Table 4.7), they do not deviate substantially from their initial values. However, doing so continuously fails to lead to correct predictions for most of the training and testing data.

Retrain. We see a similar behaviour as L1-Regularizer in Retrain. In both architectures, Retrain has a relatively low but stable increase in the training loss across iterations. Additionally, in both architectures, it fails to correctly classify any misclassified sample after a certain iteration (marked by red dots on the graphs). After the drastic drop in performance in the first few iterations, there is a range of iterations during which the training and testing accuracies seem to gradually increase by small magnitudes. However, this halts when no misclassified samples are correctly predicted, and the network stops changing. Recall that the Retrain algorithm only performs one gradient descent step on all parameters, where the gradient is computed on a single misclassified sample. Therefore, one gradient descent step is insufficient for the network to continuously learn more misclassified samples. The rise in the training and testing accuracy after the drastic drop in performance might be the network relearning old features that are similar to the training distribution and the misclassified samples. However, the accuracies plateau as the network is no longer able to learn from the new samples. This suggests that the network might be overfitted on a few samples.

Fine-tune. Recall that fine-tuning provides first-order updates to only the penultimate layer in both architectures. Unlike Top-k and BGD (and its variants), the fine-tuned parameters are not dependent on the gradients. Note that the penultimate layer has the second-highest gradient magnitudes in both AlexNet and the MLP network (see Figure 4.13)(a) and (b). Fine-tuning the penultimate layer corresponds to changing 17.93% of the AlexNet network, and 99.85% of the MLP network. For the AlexNet network, the number of changed parameters is relatively low, but the gradients of the penultimate

layer are high. On the other hand, for the MLP network, the number of parameters is almost 100%, but the gradient magnitudes are low. Note that the preservation of the training and testing performances may be caused by the relatively higher gradient magnitudes in the penultimate layer. However, we do not verify the contribution of the magnitudes of the gradients to the network's generalizability preservation, and merely note the observation. Such verification could be achieved by updating all parameters in individual layers. Instead, we do random parameter selection into subsets, and find that for a fixed manually decided number of iterations (=20), none of the misclassified samples could be correctly predicted. Therefore, while the gradient magnitude might play a role in fine-tuning, we cannot make a consistent conclusion given the lack of ablation studies. Additionally, we observe that this approach, like Retrain, fails to correctly predict most samples after learning from a few initial samples.

Top-k. We observe that in both architectures, the Top-k approach starts strongly, but degrades the network heavily in the first 50 iterations. This is indicated by the training and testing accuracies, while the training loss is similar to that of Retrain and L1-Regularizer. This is in contrast to the entries by Top-k for POC in Table 4.7 and 4.8, where the total correct predictions of all misclassified samples is 84.32% for the AlexNet network and 77.71% for the MLP network. This discrepancy may be a result of sample memorization. Although both networks alter a small fraction of network parameters (relative to L1-Regularization, Retrain, and Fine-tune), sample bias is observed to be greater, i.e., although the number of correctly predicted samples can increase, the training and testing accuracy degrades down to approximately 0.20 in both architectures. Like in L1-Regularization, Retrain, and Fine-tune, Top-k makes first-order updates to a relatively smaller fraction of parameters. Apart from fine-tuning, Top-k updates a small subset of the parameters updated in Retraining and L1-Regularization by equal magnitudes. As the only difference for Top-k lies in changing a partial network, we can infer that this approach causes localized overfitting, i.e., a part of the network is biased to

the sample, and the other parts are generalizable. Therefore, to completely lose generalizability, the network needs to continue to learn from other misclassified samples. In the process of learning, generalizability fades, although it fades slower than in other approaches that modifies most of the network parameters. Additionally, since we also observe correct predictions of said misclassified samples, a conclusion about localized overfitting, causing correct classifications can be drawn. These observations also justify the use of sparse gradients instead of updating all k highest gradient parameters.

4.8 Experiment 5: Investigating Sensitivities of BGD

In this experiment, we test the robustness to changes in assumptions to our approach. Firstly, in our approach, the parameters are sampled based on the gradient computed on the misclassified sample. This approach is adopted on the assumption that upon updating high gradient parameters, the sample loss decreases and leads to correcting the misclassification. In subsection 4.8.1, we investigate this assumption and additionally test the effect of updating high gradient parameters on overall network generalizability. Secondly, our approach uses gradients as a prior distribution to select parameters for changes, i.e., the higher the sensitivity of a parameter, the more likely it is to be included in the subset. In subsection 4.8.2, we investigate the performance of our approach when subjected to changes in the prior distribution of gradients.

4.8.1 Assumption 1

In our approach, BGD, we guide our parameter selection method using the gradients calculated for the misclassified sample. Chen et al. in [10] show that removing parameters with exceptionally large sensitivities can improve performance for a pruned network[1]. Although, the authors do this for the entire

[1]This conclusion given by Chen et al. in [10] is applicable for pruned networks. In a pruned network, only a fraction of the parameters are pruned, and the remaining parameters are fine-tuned iteratively. In comparison, we select a subset of parameters for updating the network instead of selecting a subset of parameters for pruning.

training dataset, we focus on only a single sample, that may be outside the training data distribution. As BGD, uses gradient information as a *priori* to our approach, it is necessary to answer the question of whether avoiding high sensitivity parameters, where the sensitivities depend on the sample alone, is indeed beneficial. The reason for answering this question is also related to the network's generalizability. Intuitively, as we are providing second-order updates (in BGD-II) to parameters based on the gradient calculated on the misclassified sample alone, we ought to negatively impact the network's generalizability. If avoiding highly sensitive parameters can indeed preserve past learning, using gradients might be harmful to our approach.

Set-up

As updates are calculated based on the gradients computed on the sample loss, we look more closely at these gradients. We investigate whether updating parameters with high gradients are indeed detrimental to the network, especially when the gradients are calculated on a misclassified sample alone. To do this, we provide independent second-order updates to each of the top-100 gradient parameters, updating them one at a time. We aim to analyze if updates to the k^{th} highest sensitive parameter is better for the network, than updates to the $k - x^{th}$ highest sensitive parameter. After each update to the k^{th} highest sensitive parameter, the modified network is evaluated by checking the prediction of the previously misclassified sample. If the update corrects the classification, we then measure the training loss, and both training and testing accuracy. The intuition behind this approach is to identify which parameters, when updated, cause the least deviation from the main network's performance while still correcting the misclassification.

Gradients, $\nabla_{\mathbf{W}}\mathcal{L} = \left[\frac{\partial\mathcal{L}(\mathbf{W},N(s_i))}{\partial w_1}, \frac{\partial\mathcal{L}(\mathbf{W},N(s_i))}{\partial w_2}, ..., \frac{\partial\mathcal{L}(\mathbf{W},N(s_i))}{\partial w_n}\right]$, are calculated for the sample loss with respect to all parameters. The gradient vector is then sorted in descending order, where the top-k^{th} gradient parameter denotes the k highest gradient magnitude.

Results

To observe an overall trend in the evaluation metrics, we plot the average training loss, and both average training and testing accuracy (metrics are calculated on all the samples and averaged) in Figure 4.11 (for the AlexNet network) and Figure 4.12 (for the MLP network). The figures plot the resultant performance for the networks upon changing the top-1 gradient parameter (in descending order of the gradient) to the top-100 gradient parameter. Performance is only recorded for the samples that were correctly predicted after updating the top-k^{th} gradient parameter. For example, in Figure 4.11(b), the training accuracy is the highest for the top-1 gradient parameter: meaning that among all the samples that were correctly predicted (by altering the top-k^{th} gradient parameter), upon updating the highest gradient parameter, we observe the highest training accuracy. Therefore, although the samples might get correctly predicted by updating other top-k^{th} gradient parameters, altering the highest gradient parameter retains the training accuracy to the maximum for the AlexNet network. However, this observation is not generalizable across different architectures. To better analyze the top-k gradients, we plot the best three performances in Tables 4.9 and 4.10.

In the Tables 4.9 and 4.10, the highest-level columns indicate information for the gradient rank that led to the best (1st), second best (2nd), and third best (3rd) values for the row attributes, training loss, training accuracy, and testing accuracy. Therefore, the best training loss is the minimum training loss observed, and the best training and testing accuracies are the highest training and testing accuracies observed. 'Rank' indicates the gradient rank. For example, a rank of 1 indicates it is the most sensitive parameter (based on the gradients calculated for a sample). POC represents the number of samples that are correctly predicted if the gradient parameter at Rank k is updated. For example, on updating the highest gradient parameter in the MLP network, the training accuracy is maximum for 31.89% of samples. These samples were previously misclassified but got correctly predicted on updating the highest gradient paramter (Recall that we apply second-order updates to

the parameter).

In the aforementioned tables, we do not observe any rank advantage of the gradients, i.e., we do not find a generalizable range of ranks within which the network is bound to have benefits in terms of preserving or increasing generalizability. In the Table 4.9, the 86^{th} highest gradient parameter has the lowest training loss for 6.75% of the misclassified samples. However, on moving down the column for the best observations (the column 1st in Table 4.9), we see that updating the 86^{th} highest gradient parameter did not yield either of the highest training accuracy or the highest testing accuracy. The discrepancy between the three distinct statistics suggest a lack of deterministic relation between them, i.e., the lowest training loss does not necessitate the highest training or testing accuracy. Therefore, from the tables, we observe there is a lack of a clear relationship for the three performance metrics. Two inferences can be drawn from these observations: (a) An excessively small sample loss does not guarantee a correct prediction of the sample, and (b) making top-k gradient updates is not the shortest way to achieve a correct prediction.

One might be tempted to think that tweaking the highest gradient parameter would give the lowest (and hence, the best) training loss. However, that is not the case for the two following reasons: (a) Training loss is hardly related to the gradient calculated on the misclassified sample. Adjusting the parameter with the highest sensitivity to the sample loss might or might not decrease the training loss. (b) Although the sample loss is the lowest after adjusting the highest gradient parameter, the values in the tables are considered only if the sample is correctly predicted. Therefore, a low sample and training loss does not necessarily mean a correct sample prediction.

Among the top-100 parameters, given the condition that only a single parameter needs to be updated to get a correct prediction, we do not find a clear relationship between updating the highest gradient elements and the generalizability of the network. This conclusion is however, only dependent on updating (using second-order updates) a single parameter. It is unknown what happens when more than one parameter is updated. We do not perform those experiments as combining x parameters deterministically, where $x \in$

	1st		2nd		3rd	
	Rank	POC	Rank	POC	Rank	POC
Training Loss	86	6.75	92	6.75	58	6.21
Training Accuracy	9	19.72	3	19.45	11	19.45
Testing Accuracy	90	18.91	39	13.51	74	13.51

Table 4.9: Comparing the parameters with minimum training loss, maximum training accuracy, and maximum testing accuracy for the AlexNet network

$[1, |\mathbf{W}|]$, to analyze a relationship between those parameters and the network's generalizability is computationally infeasible. Therefore, we cannot conclude if this shorter way of only updating one parameter is better for the network when compared to updating a larger subset of parameters.

	1st		2nd		3rd	
	Rank	POC	Rank	POC	Rank	POC
Training Loss	59	37.59	20	37.06	1	5.59
Training Accuracy	1	31.89	59	13.19	20	10.13
Testing Accuracy	1	17.95	19	5.38	40	4.54

Table 4.10: Comparing the parameters with minimum training loss, maximum training accuracy, and maximum testing accuracy for the MLP network

4.8.2 Assumption 2

Our approach is guided by gradients to sample parameters to update. Here, we investigate the effects of changes to the gradient distribution. An additional motivation is to provide our algorithm with equal opportunity for selecting parameters of each layer of the network. As shown in Figure 4.13, the layer-wise average gradient magnitude varies in each layer in different architectures. In the Figure, the absolute values of the gradients (calculated for a single misclassified sample) is summed over all the parameters in a particular layer. We repeated this process for all the samples, and plot the average gradient magnitudes in the Figure. This gives us an estimate of which layer parameters are more likely to be updated in our approach. The first fully connected layer, 'lin1', has the highest gradient magnitudes of all the layers for all misclassified

samples. This means that BGD is most likely to sample gradients from 'lin1' layer, without considering any other layer. In the Top-k approach, however, this happens deterministically, and therefore, the majority of the elements updated belong to the 'lin1' layer. Zhou et al. in [45] show that when networks are aggressively pruned, almost all parameters in certain layers get pruned. As altering the fewest possible parameters is one of our primary goals, we aim to provide equal opportunity to parameters in each layer. Specifically, we normalize by dividing the gradient magnitudes of each layer by the total gradient magnitude of that layer. If a layer has l parameters, then all parameters in the layer are normalized by the following:

$$P(\mathbf{W}) = \left\lceil \frac{\left|\frac{\partial \mathcal{L}}{\partial w_i}\right|}{\sum_j^l \left|\frac{\partial \mathcal{L}}{\partial w_j}\right|} \right\rceil_{i=0}^{d}$$

The gradients after normalization are visualized in Figure 4.13(c) & (d). Normalization preserves the gradient distribution among parameters within each layer while ensuring the proportional contribution of each parameter within its respective layer. The distribution of gradient magnitudes is therefore standardized across layers. In a layer, the gradients retain their individual distributions relative to other parameters in that layer. Thus, while our algorithm samples gradients uniformly across all layers, high-gradient parameters within a layer are more likely to be sampled.

Prioritizing parameters with high gradients (relative to other parameters in the same layer) can, however, lead to updates that do not reduce the sample loss as effectively as those from non-normalized gradients. This is because the sampled parameters are more likely to have relatively lower gradients upon normalization. Therefore, normalization potentially misleads our algorithm and may cause failure to solve misclassifications.

Apart from investigating the robustness of our approach via changes to the prior probabilities distribution, normalization enables us to gain insights about altering relatively lower gradient parameters. We investigate these effects by comparing the results of our approach to a non-normalized variant. In figure 4.14, we compare the distribution of accuracies for both testing and training

	BGD	**BGD-Norm**
POC	**100 ± 0**	**100 ± 0**
POPA	**1.9e-05 ± 1.5e-05**	**1.9e-05 ± 1.5e-05**
Δ TL	-3.3e-03 ± 2.5e-05	**-3.3e-3 ± 5.3e-04**
Δ TA	**0.09 ± 0.07**	0.11 ± 0.08

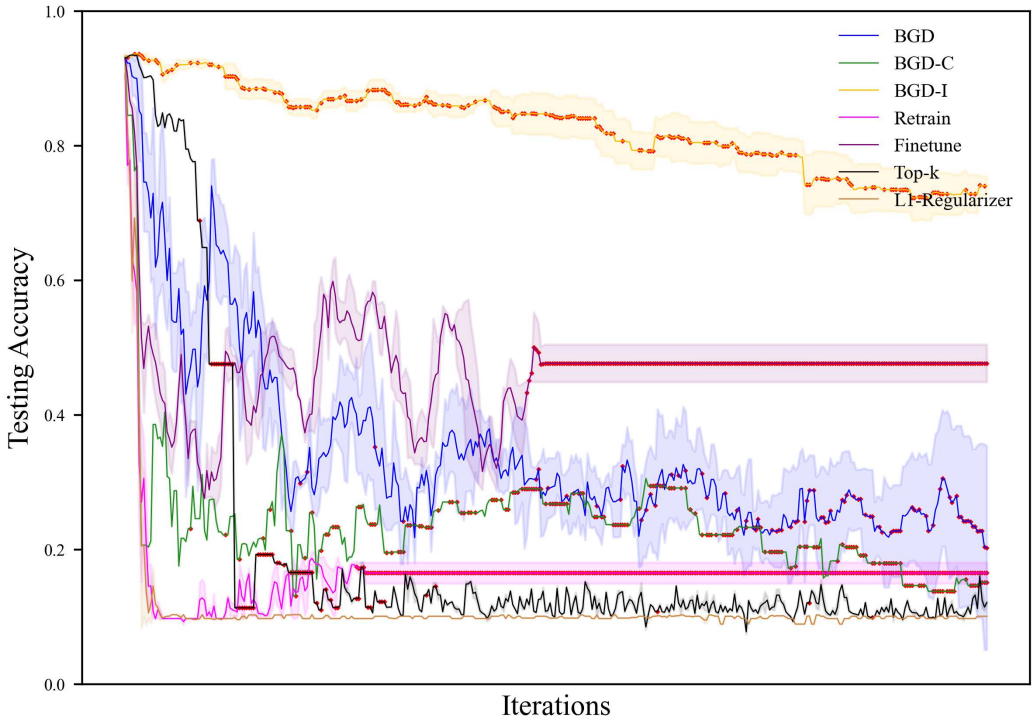
Table 4.11: Comparing BGD with BGD-Norm for the AlexNet architecture

	BGD	**BGD-Norm**
POC	**100 ± 0**	**100 ± 0**
POPA	**0.02 ± 0.02**	**0.02± 0.02**
Δ TL	-3.3e-03 ± 1.8e-05	**-6.7e-3 ± 4.8e-05**
Δ TA	**0.03± 0.01**	0.04 ± 0.05

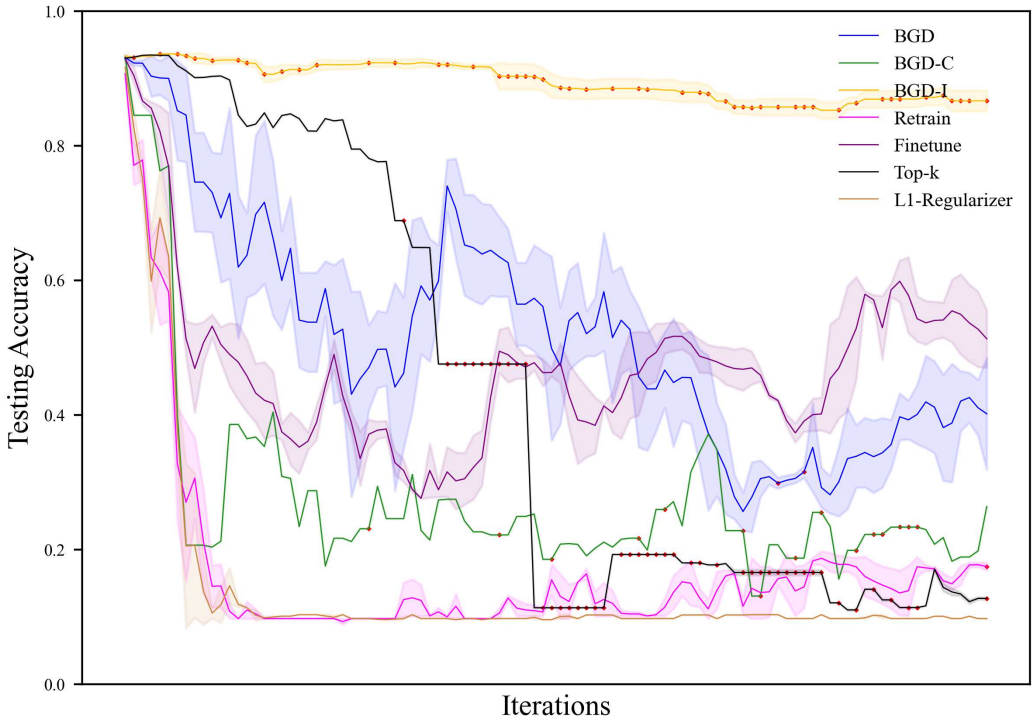
Table 4.12: Comparing BGD with BGD-Norm for the MLP architecture

dataset. Table 4.11 & 4.12 record the values of each metrics defined in Section 4.2. We observe that although normalization has the potential to obstruct our method, BGD-Norm succeeds in correctly classified 100% of the misclassifed samples for both networks.

In earlier experiments, we observed decreases in both training loss and training/testing accuracy, despite not fine-tuning on training examples. Similar behavior was observed with BGD-Norm. The decrease in training loss was more pronounced in BGD-Norm compared to BGD. Since the only difference between the two methods is that BGD-Norm is more likely to sample relatively lower gradient parameters (compared to BGD), we can conclude that updating low gradient parameters promotes over-reliance on certain features. Although these features are not uniquely extracted from the sample, the over-reliance (on fewer features) causes the network to disrupt a few training and testing examples, hence the decrease in accuracies.

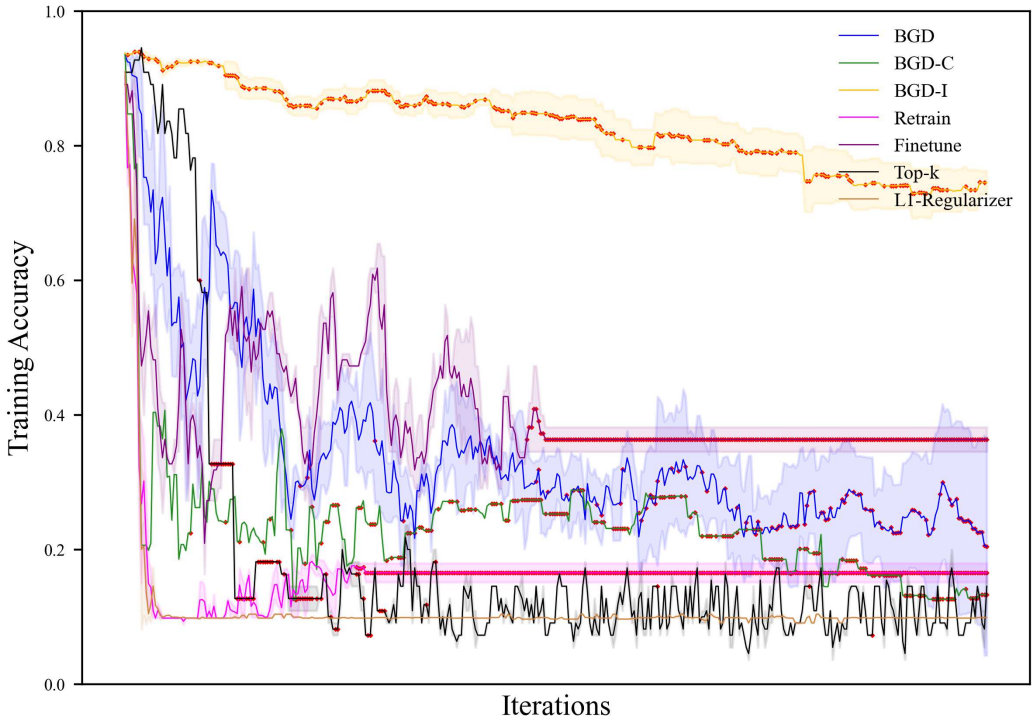


(a) All misclassified samples

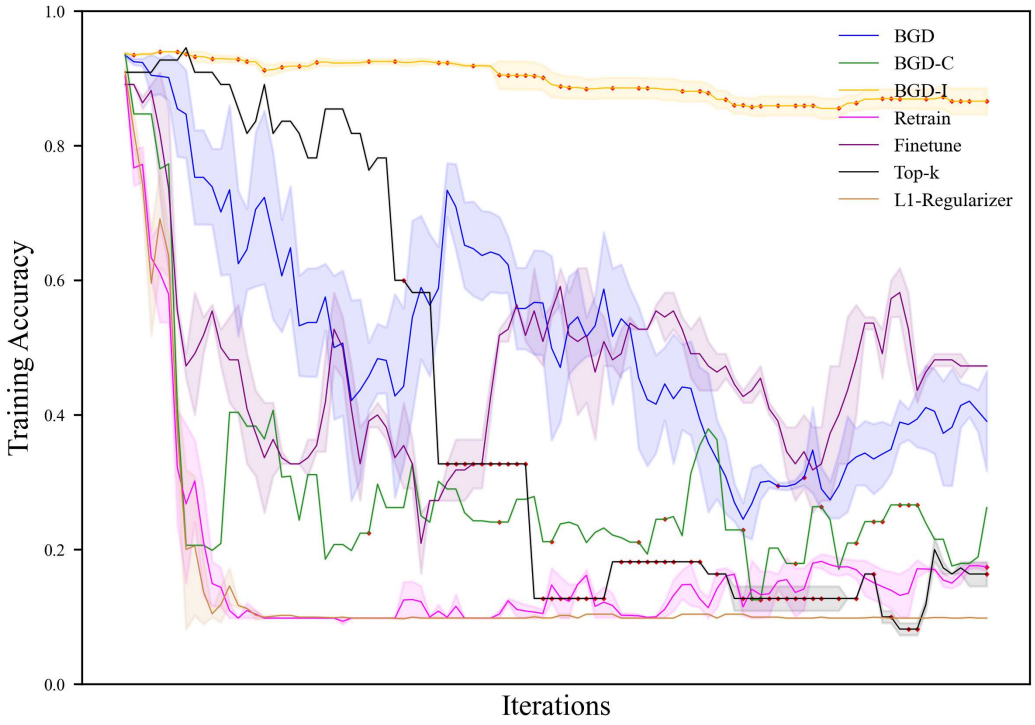


(b) First 100 misclassified sample

Figure 4.5: Comparing the testing accuracies for BGD, BGD-C, and different baseline approaches for the AlexNet network.

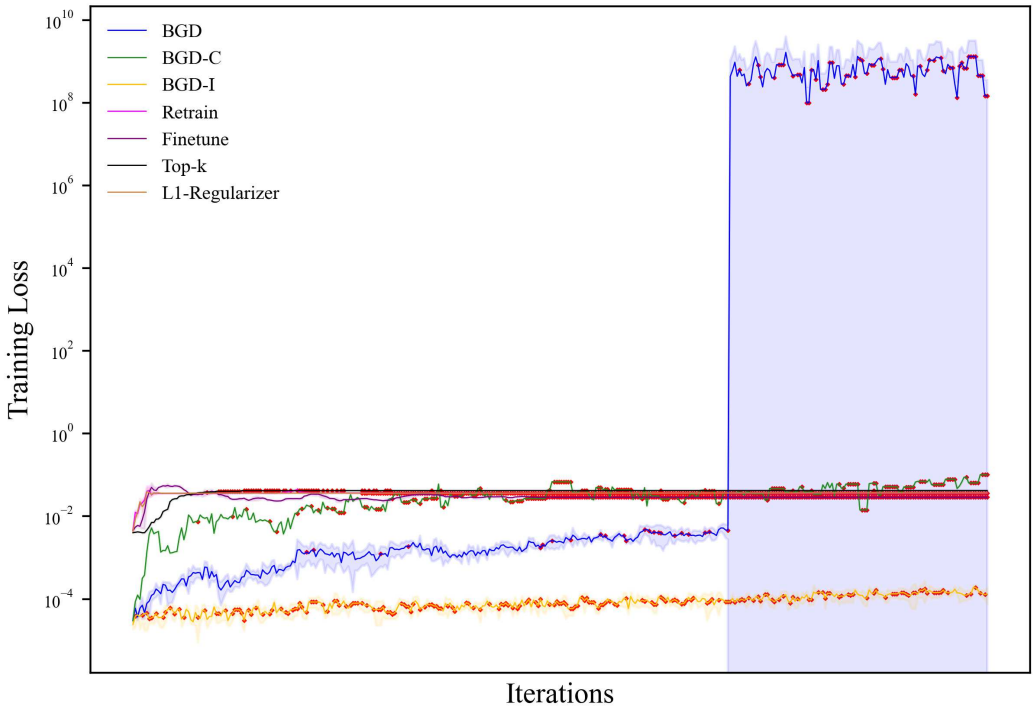


(a) All misclassified samples

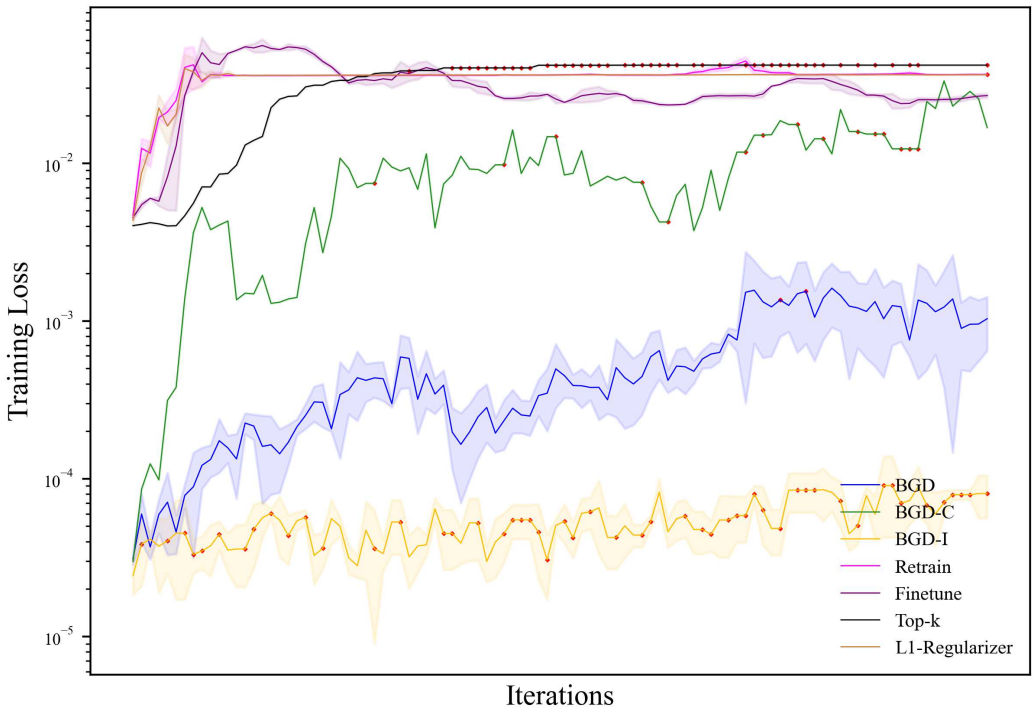


(b) First 100 misclassified sample

Figure 4.6: Comparing the training accuracies for BGD, BGD-C, and different baseline approaches for the AlexNet network.

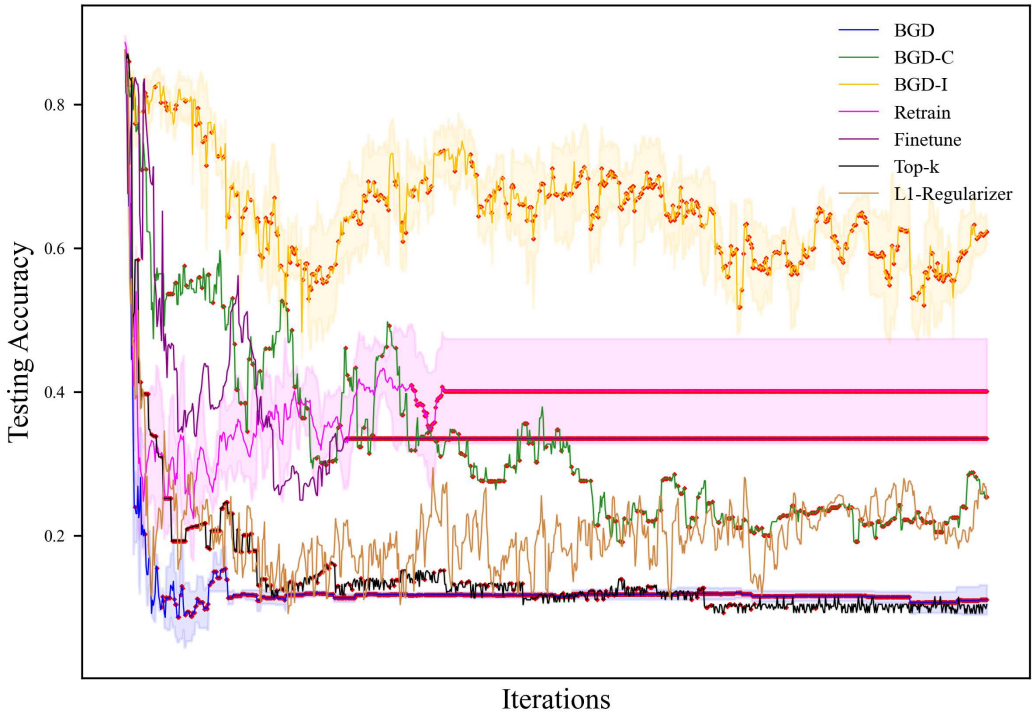


(a) All misclassified samples

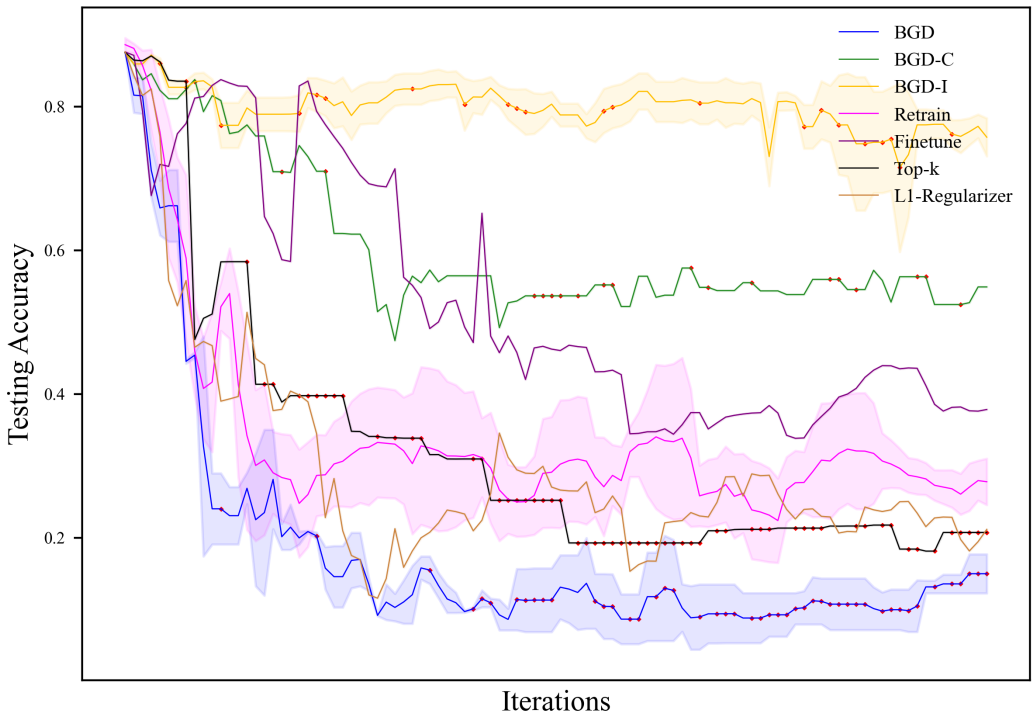


(b) First 100 misclassified sample

Figure 4.7: Comparing the training loss for BGD, BGD-I, BGD-C, and different baseline approaches for the AlexNet network.

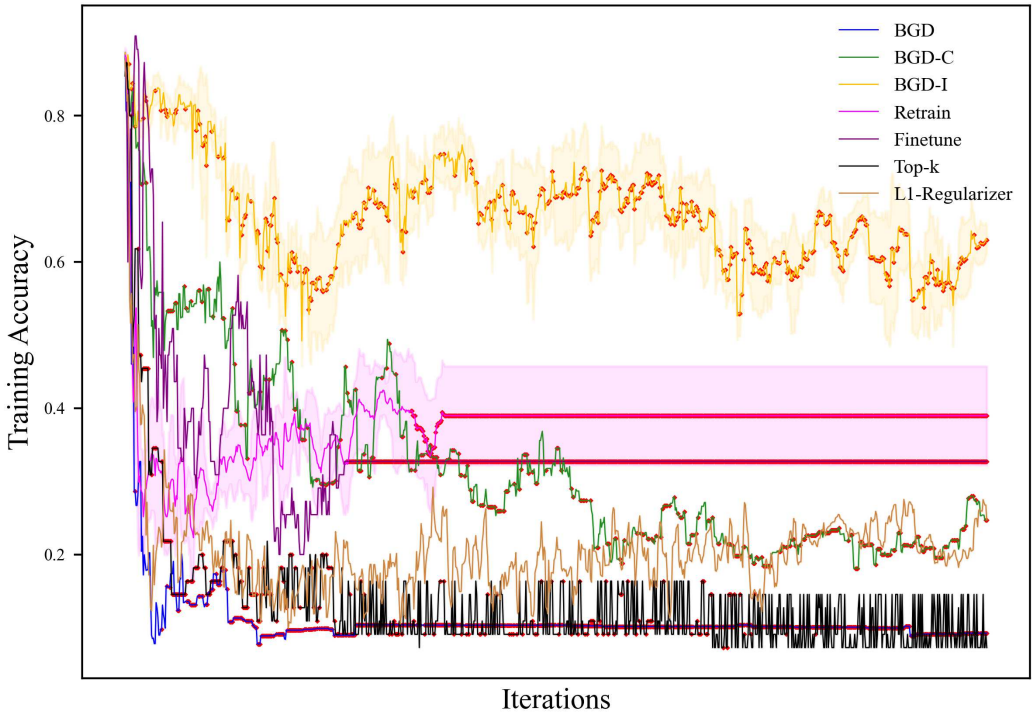


(a) All misclassified samples

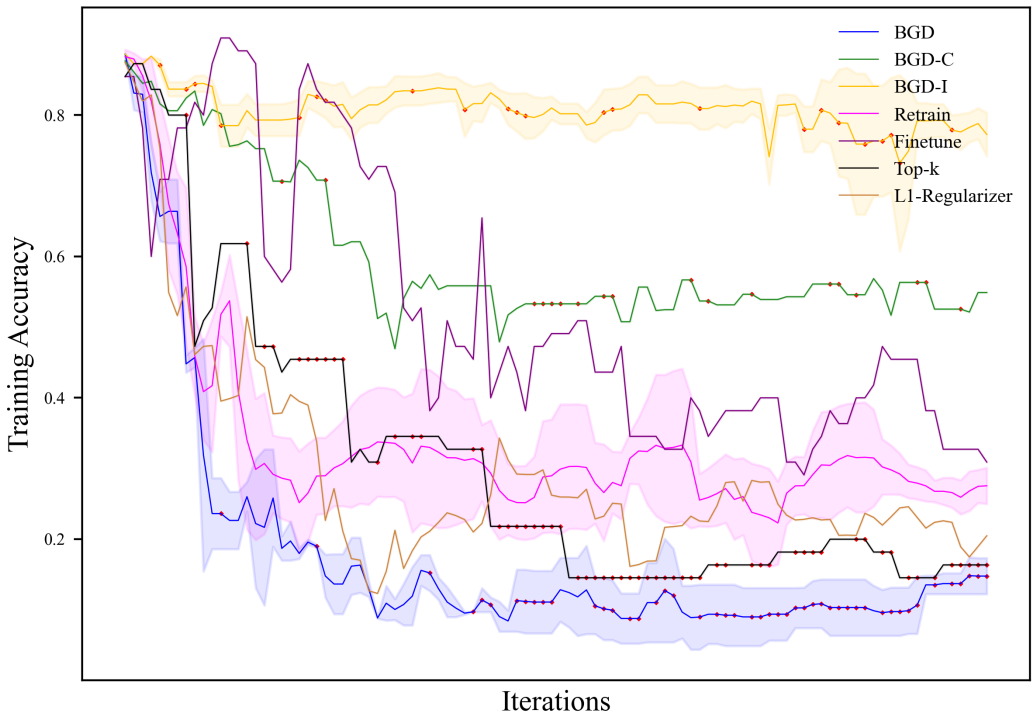


(b) First 100 misclassified sample

Figure 4.8: Comparing the testing accuracies for BGD, BGD-C, and different baseline approaches for the MLP network.

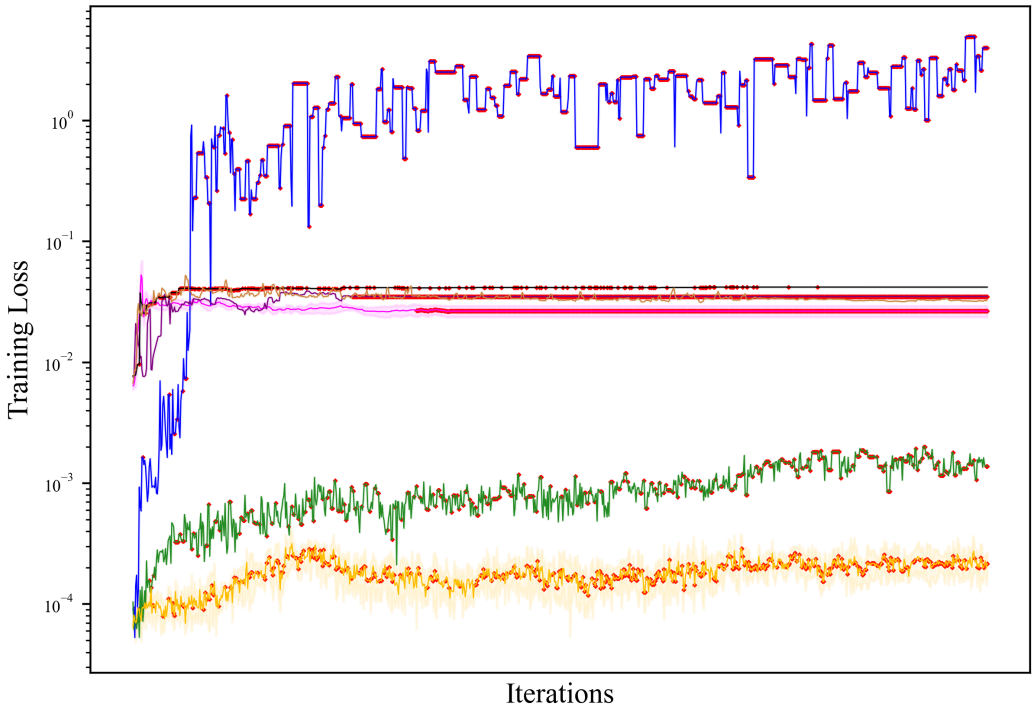


(a) All misclassified samples

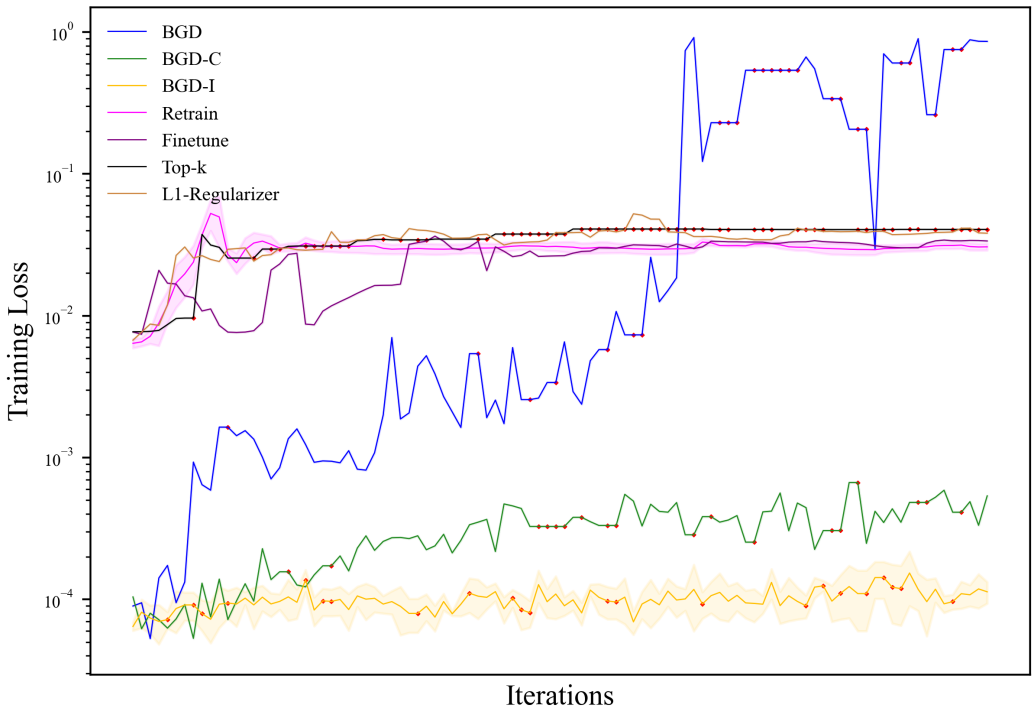


(b) First 100 misclassified sample

Figure 4.9: Comparing the training accuracies for BGD, BGD-C, and different baseline approaches for the MLP network.

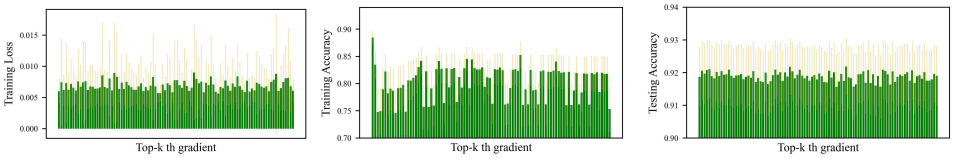


(a) All misclassified samples



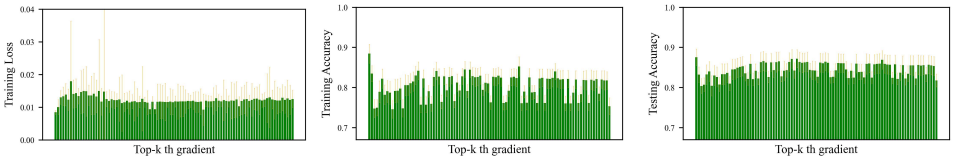
(b) First 100 misclassified sample

Figure 4.10: Comparing the training loss for BGD, BGD-I, BGD-C, and different baseline approaches for the MLP network.



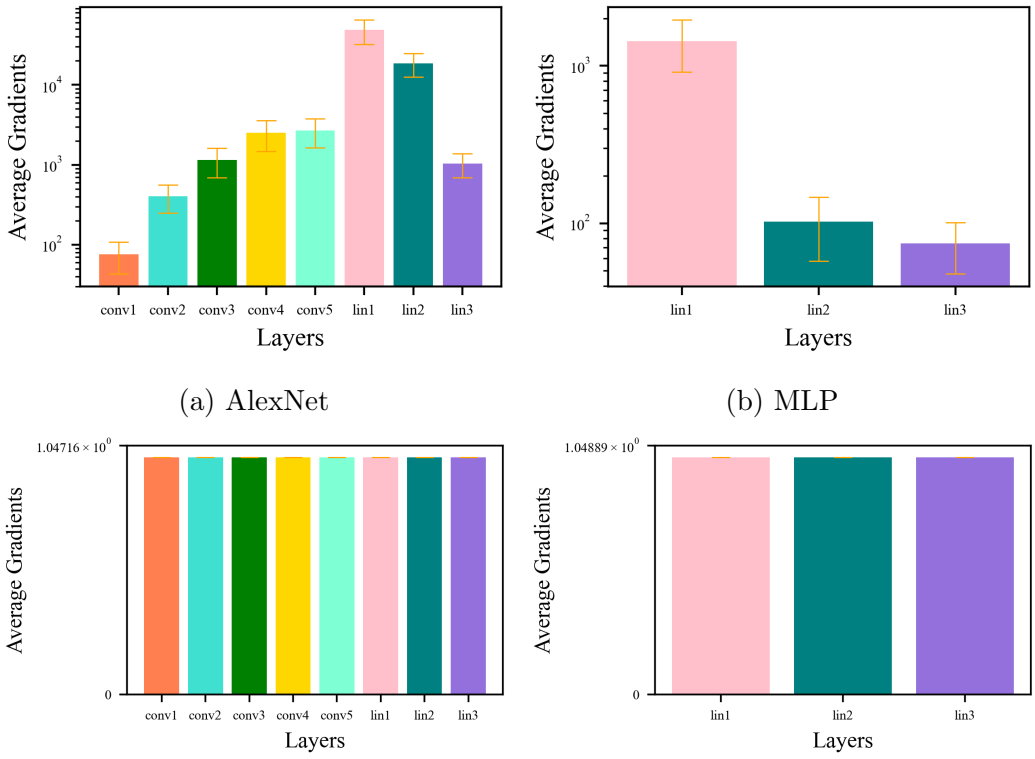
(a) (b) (c)

Figure 4.11: Performance metrics after updating each of the top-k gradient parameters for the AlexNet network.



(a) (b) (c)

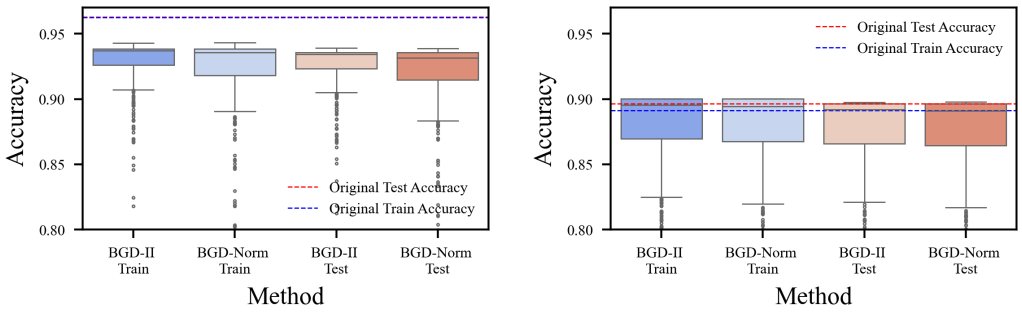
Figure 4.12: Performance metrics after updating each of the top-k gradient parameters for the MLP network.



(a) AlexNet (b) MLP

(c) AlexNet (d) MLP

Figure 4.13: Layer-wise variance in gradient magnitudes before (first row) and after (second row) normalization. For both architectures, the gradient magnitude is summed for a layer l. This is averaging across all the misclassified samples for visualization purposes.



(a) AlexNet (b) MLP

Figure 4.14: Comparing testing and training accuracies for both architectures. Here, 'BGD-II' is our approach, introduced in Chapter 3, is compared with the normalized variant of our approach, 'BGD-Norm'.

Chapter 5

Conclusion

In this thesis, we propose a sparse regularizer called Budgeted Gradient Descent (BGD), that aims to modify the network such that previous misclassifications are corrected. Like SGD, the network learns from each misclassified sample sequentially. Apart from learning from misclassifications, our secondary focus lies in preserving the pre-trained network's generalizability. Our approach is blind to the training data, and therefore does not require its presence after deployment. Given the sparse nature of our algorithm, it also provides us with an opportunity to understand how much sparsity is beneficial for preserving network generalizability.

5.1 Takeaways

From experiments with the AlexNet and an MLP network on the MNIST dataset, we gather the following conclusions: (For the below list, the mentioned gradients are computed on a single misclassified sample.)

1. The first-order variant of our approach, BGD-I, uses the first-order update rule while the second-order variant, BGD-II, uses the second-order update rule. We observe that while BGD-II provides aggressive updates to parameters, fewer parameters need to be updated to solve the misclassifications. Comparatively, BGD-I updates a larger fraction of parameters, while the magnitude of these updates are relatively low. Therefore, if one aims to alter the fewest network parameters possible, one might

prefer BGD-II. On the other hand, one might choose BGD-I, to better preserve generalizability.

2. Another variant of our approach, BGD-C, avoids updating the parameters that produce very high training loss. While achieving this goal may lead to increase in generalization preservation, it solves less misclassifications, and does not guarantee an increase in training and testing accuracies.

3. On comparing all the variants of our approaches with various baseline approaches, namely, Random Sampling, Retraining, Finetuning, Top-k, and L1-Regularization, we find that our approaches BGD-I and BGD-II can successfully solve most misclassifications. Moreover, when tested for generalizability in the long term (by sequentially updating the network on samples), we find that most approaches that alter the majority of the network lead to it losing its generalizability.

4. Our approach is guided by the gradient calculated on the misclassified sample. Upon systematically updating parameters, one at a time, from the top-100 magnitude-wise gradient parameters, we observe that although updating parameters with relatively higher gradient magnitudes help with lowering sample loss, and correct classifications of misclassified samples, there is no clear relationship between the rankings of these parameters (magntitude-wise), and correct predictions of misclassified samples in these top parameters.

5. We observe that sparsity plays a role in preventing sample memorization and preserving network's generalizability. The Top-k approach, for example, updates k network parameters. Even though it is a small fraction of parameters, a misclassified sample might not need to be updated by all top k parameters. There might exist a solution that comprises a subset of those top k parameters. Introducing such sparsity can decrease sample memorization and loss of generalizability by limiting localized overfitting.

5.2 Future directions

We assume that the misclassifications can be solved by the gradients of the loss calculated on the single sample. However, in pruning literature there exist other utility functions and criteria. One such criterion for sampling parameters can be the activation values of each neuron. A future study, guided by said criterion, may investigate the relationship among correcting misclassifications, overall loss, and loss/increase in generalizability.

Among the variants of our proposed method is BGD-II that provides second-order updates to selective parameters. As these updates are aggressive, very few parameters need to be updated. However, we observe that a lack of additional regularization and training data ensuring preservation of generalizability can be challenging. Therefore, the second order variant, BGD-II, fails to preserve generalizability in the long run. However, we compute these second order updates based on the Empirical Fisher approximation. As it provides only an approximation to the Hessian matrix, the second-order updates may not be accurate, thereby degrading our approach. To investigate the dependence of our approach, BGD-II needs to be evaluated after applying other alternative Hessian approximation techniques, such as, Kronecker-Factored Approximate Curvature (KFAC) [24].

References

[1] M. Abramowitz and I. A. Stegun, *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. US Government printing office, 1968, vol. 55.

[2] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *arXiv preprint arXiv:1704.05021*, 2017.

[3] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The convergence of sparsified gradient methods," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[4] R. Aljundi, "Continual learning in neural networks," *arXiv preprint arXiv:1910.02718*, 2019.

[5] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural computation*, vol. 10, no. 2, pp. 251–276, 1998.

[6] F. Bach, "Breaking the curse of dimensionality with convex neural networks," *Journal of Machine Learning Research*, vol. 18, no. 19, pp. 1–53, 2017.

[7] T. Becker, S. H. Babey, R. Dorsey, and N. A. Ponce, "Data disaggregation with american indian/alaska native population data," *Population Research and Policy Review*, vol. 40, no. 1, pp. 103–125, 2021.

[8] A. S. Bosman, A. Engelbrecht, and M. Helbig, "Empirical loss landscape analysis of neural network activation functions," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, 2023, pp. 2029–2037.

[9] H.-S. Chang, E. Learned-Miller, and A. McCallum, "Active bias: Training more accurate neural networks by emphasizing high variance samples," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[10] C. Chen, Z. Fu, K. Liu, Z. Chen, M. Tao, and J. Ye, "Optimal parameter and neuron pruning for out-of-distribution detection," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[11] S. Dohare, R. S. Sutton, and A. R. Mahmood, "Continual backprop: Stochastic gradient descent with persistent randomness," *arXiv preprint arXiv:2108.06325*, 2021.

[12] V. Feldman and C. Zhang, "What neural networks memorize and why: Discovering the long tail via influence estimation," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2881–2891, 2020.

[13] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," *Advances in neural information processing systems*, vol. 5, 1992.

[14] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE international conference on neural networks*, IEEE, 1993, pp. 293–299.

[15] J. He, R. Mao, Z. Shao, and F. Zhu, "Incremental learning in online scenario," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 13 926–13 935.

[16] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," *Journal of Machine Learning Research*, vol. 22, no. 241, pp. 1–124, 2021.

[17] S. C. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online learning: A comprehensive survey," *Neurocomputing*, vol. 459, pp. 249–289, 2021.

[18] D. Kim and B. Han, "On the stability-plasticity dilemma of class-incremental learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 20 196–20 204.

[19] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," *Advances in neural information processing systems*, vol. 28, 2015.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[21] T. LaBonte, V. Muthukumar, and A. Kumar, "Towards last-layer retraining for group robustness with fewer annotations," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[22] Z. Liu, Z. Xu, J. Jin, Z. Shen, and T. Darrell, "Dropout reduces underfitting," in *International Conference on Machine Learning*, PMLR, 2023, pp. 22 233–22 248.

[23] R. Ma, J. Miao, L. Niu, and P. Zhang, "Transformed 1 regularization for learning sparse deep neural networks," *Neural Networks*, vol. 119, pp. 286–298, 2019.

[24] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," in *International conference on machine learning*, PMLR, 2015, pp. 2408–2417.

[25] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 264–11 272.

[26] M. Pezeshki, O. Kaba, Y. Bengio, A. C. Courville, D. Precup, and G. Lajoie, "Gradient starvation: A learning proclivity in neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1256–1272, 2021.

[27] X. Qiu and R. Miikkulainen, "Detecting misclassification errors in neural networks with a gaussian process model," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 8017–8027.

[28] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[29] A. Saha, A. Subramanya, and H. Pirsiavash, "Hidden trigger backdoor attacks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 11 957–11 965.

[30] S. Salman and X. Liu, "Overfitting mechanism and avoidance in deep neural networks," *arXiv preprint arXiv:1901.06566*, 2019.

[31] M. Schmidt, G. Fung, and R. Rosales, "Fast optimization methods for l1 regularization: A comparative study and two new approaches," in *Machine Learning: ECML 2007: 18th European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007. Proceedings 18*, Springer, 2007, pp. 286–297.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[33] X. Sun, X. Ren, S. Ma, and H. Wang, "Meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting," in *International Conference on Machine Learning*, PMLR, 2017, pp. 3299–3308.

[34] A. Tang, P. Quan, L. Niu, and Y. Shi, "A survey for sparse regularization based compression methods," *Annals of Data Science*, vol. 9, no. 4, pp. 695–722, 2022.

[35] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International conference on machine learning*, PMLR, 2013, pp. 1058–1066.

[36] C. Wang, R. Grosse, S. Fidler, and G. Zhang, "Eigendamage: Structured pruning in the kronecker-factored eigenbasis," in *International conference on machine learning*, PMLR, 2019, pp. 6566–6575.

[37] M. Wicker, X. Huang, and M. Kwiatkowska, "Feature-guided black-box safety testing of deep neural networks," in *Tools and Algorithms for the Construction and Analysis of Systems: 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I 24*, Springer, 2018, pp. 408–426.

[38] Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M. Mahoney, "Adahessian: An adaptive second order optimizer for machine learning," in *proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 10 665–10 673.

[39] S.-K. Yeom, P. Seegerer, S. Lapuschkin, *et al.*, "Pruning by explaining: A novel criterion for deep neural network pruning," *Pattern Recognition*, vol. 115, p. 107 899, 2021.

[40] J. Yoon and S. J. Hwang, "Combined group and exclusive sparsity for deep neural networks," in *International Conference on Machine Learning*, PMLR, 2017, pp. 3958–3966.

[41] S. Yu, Z. Yao, A. Gholami, *et al.*, "Hessian-aware pruning and optimal neural implant," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 3880–3891.

[42] M. B. Zafar, I. Valera, M. Gomez-Rodriguez, and K. P. Gummadi, "Fairness constraints: A flexible approach for fair classification," *Journal of Machine Learning Research*, vol. 20, no. 75, pp. 1–42, 2019.

[43] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[44] G. Zhang, J. Martens, and R. B. Grosse, "Fast convergence of natural gradient descent for over-parameterized neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[45] X. Zhou, W. Zhang, H. Xu, and T. Zhang, "Effective sparsification of neural networks with global sparsity constraint," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3599–3608.

[46] F. Zhuang, Z. Qi, K. Duan, *et al.*, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.