LLBOOST: LAST LAYER PERTURBATION TO BOOST PRE-TRAINED NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

While deep networks have produced state-of-the-art results in several domains from image classification to machine translation, hyper-parameter selection remains a significant computational bottleneck. In order to produce the best possible model, practitioners often search across random seeds or use ensemble methods. As models get larger, any method to improve neural network performance that involves re-training becomes intractable. For example, computing the training accuracy of FixResNext-101 (829 million parameters) on ImageNet takes roughly 1 day when using 1 GPU.

In this work, we present LLBoost, a theoretically-grounded, computationally-efficient method to boost the validation accuracy of pre-trained over-parameterized models without impacting the original training accuracy. LLBoost adjusts the last layer of a neural network by adding a term that is orthogonal to the training feature matrix, which is constructed by applying all layers but the last to the training data. We provide an efficient implementation of LLBoost on the GPU and demonstrate that LLBoost, run using only 1 GPU, improves the test/validation accuracy of pre-trained models on CIFAR10, ImageNet32, and ImageNet. In the over-parameterized linear regression setting, we prove that LLBoost reduces the generalization error of any interpolating solution with high probability without affecting training error.

1 INTRODUCTION

Over the past decade, deep networks have produced a number of state-of-the-art results including surpassing human performance on the ImageNet classification task (Russakovsky et al., 2015; He et al., 2016). However, tuning hyper-parameters to produce the best possible neural network in a given application remains a computationally expensive task. State-of-the-art results often involve selecting the best model using multiple random seeds (He et al., 2016; Szegedy et al., 2015; Hasanpour et al., 2016; Touvron et al., 2019) or ensembling (Huang et al., 2017), and training even a single model can take several days even when using multiple GPUs. Hence, it is critical to identify computationally efficient approaches to improve the performance of pre-trained models without the need of re-training them.

In this work, we present LLBoost, a theoretically-grounded, computationally-efficient method to boost the validation accuracy of pre-trained, over-parameterized models without impacting the training accuracy. Figure 1 provides an overview of our method as well as the main results. As shown in Figure 1A, our method adjusts the last fully-connected layer of a neural network by selecting the best performing perturbation out of several orthogonal to the training feature matrix, which is constructed by applying all but the last layer to the training data. In Figure 1B, we provide an example showing how our method applied to a model trained under a poorly chosen random seed can boost validation accuracy comparable to that of a model trained under a better seed. Lastly, Figure 1C shows that our method can significantly improve the validation accuracy of pre-trained neural networks on large datasets using a fraction of the training time.

The intuition for our method is based on characterizing the benefit of random initialization in over-parameterized linear regression. In particular, consider a dataset $(X, y) \subset \mathbb{R}^{d\times n} \times \mathbb{R}^{1\times n}$ with $n < d$ for which there exists $w^* \in \mathbb{R}^{1\times d}$ such that $y = w^*X$. In order to estimate w^* from the data, we use gradient descent with learning rate η and initialization $w^{(0)}$ to minimize the squared loss, i.e. to

solve:

$$\arg\min_{w \in \mathbb{R}^d} \frac{1}{2}\|y - wX\|^2.$$

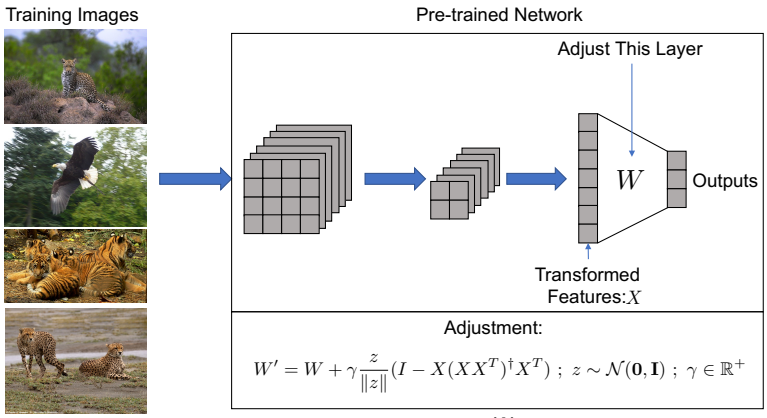
It is well-known (Engl et al., 1996) that gradient descent converges to the following closed-form solution:

$$w_r = w^{(0)}(I - X(X^T X)^\dagger X^T) + yX^\dagger,$$

where $A^\dagger$ is the pseudo-inverse of A (see Appendix A for a proof). In this work, we prove that when $n < d$, sampling $w^{(0)}$ on a hyper-sphere leads to w_r with lower generalization error than the minimum norm solution $yX^\dagger$ with constant probability. Since the last layer of modern deep networks is usually linear, we apply this result from regression to the last layer of networks to arrive at our method, LLBoost.

We end the introduction with a summary of our main contributions:

1. We present LLBoost, a computationally efficient method for boosting the validation accuracy of pre-trained neural networks without affecting training accuracy. Additionally, we provide an efficient implementation of LLBoost on the GPU.
2. We provide a wealth of empirical evidence that LLBoost improves the validation accuracy of pre-trained neural networks and prove that it does not affect training accuracy for over-parameterized models.
3. We provide evidence that LLBoost yields a computational advantage over random seed search for training neural networks.
4. In the over-parameterized linear regression setting, we prove that LLBoost reduces the test error of any interpolating solution with high probability without affecting training error.



(A)

Dataset: 100 Ex. ImageNet32 Model: Pret-trained ResNet-18	
Seed	Val Acc.
200	80%
3	78%
70	85%
25	80%
500	82%
Our Method on Seed 3	**85%**

(B)

Dataset	Model	Train/Val. Acc. (Original)	Train/Val. Acc. (Ours)	Training Time	Correction Time
100 Ex. ImageNet32	ResNet-50	100% / 83%	100% / **89%**	164.07 sec	0.12 sec
CIFAR10	ResNet-18	99.99%/95.05%	99.99%/**95.25%**	1.35 hr	15.36 min
ImageNet	FixResNext-101	94.92% / 86.26%	94.92% / **86.34%**	>1 day/epoch	7.59 hr

(C)

Figure 1: An overview of LLBoost and a demonstration of the benefits of our method in boosting pre-trained models. (A) A breakdown of LLBoost, which improves the performance of pre-trained neural networks by adjusting the last layer of a pre-trained network with a term orthogonal to the training feature matrix, X. (B) LLBoost applied to a model trained on a poorly-chosen seed boosts performance to that of a model trained on a well-chosen seed. (C) LLBoost provides a boost to the pre-trained models at a fraction of the computational cost of training.

2 RELATED WORK

Understanding the benefits of over-parameterization is a recent topic of major interest in machine learning. Belkin et al. (2019a) introduced the double descent curve showing that when increasing model capacity past the interpolation threshold, test error can drop a second time. Loog et al. (2020) noted that this phenomenon had been noticed empirically several decades prior to the work of Belkin et al. (2019a). A line of recent work have provided a theoretical analysis of double descent in linear regression (Hastie et al., 2019; Belkin et al., 2019b; Bartlett et al., 2020; Muthukumar et al., 2020; Bibas et al., 2019; Mitra, 2019). In particular, Hastie et al. (2019); Belkin et al. (2019b) analyzed the generalization error for the minimum norm solution in over-parameterized linear models. These works proved that the minimum norm solution can yield lower generalization error in the over-parameterized regime than in the under-parameterized regime. Our theoretical analysis primarily builds on that of Belkin et al. (2019b), but we analyze the generalization error of random-initialization interpolating solutions instead of the minimum-norm solution.

Various methods, including random seed search and ensembling, are standardly used to improve the performance of machine learning models through training. Bengio (2012) recommends using 5-10 random seeds if computational power is available, and this is typically done in state-of-the-art models; for example, He et al. (2016) considered 5 random seeds for ResNet-110 and Hasanpour et al. (2016) considered at least 2 random seeds on CIFAR10 (Krizhevsky, 2009). Erhan et al. (2010) rigorously analyzed the impact of random seeds on model performance by considering between 50-400 random seeds for models of varying depth on MNIST (LeCun et al., 1998). Figure 1 from their work demonstrates that random seeds can affect the validation accuracy of models by up to a few percentage points on MNIST. Since this comes at a significant computational price, it is critical to identify methods that can obtain such boosts without having to perform random seed search, which is the topic of this paper.

Another popular approach to improving neural network performance is ensembling, for example via bagging (Breiman, 1996) or boosting (Freund & Schapire, 1999). Recently, Huang et al. (2017) presented an approach to ensembling, which involved training a single network and saving the model parameters each time training stopped at a local minimum. Huang et al. (2017) built upon Xie et al. (2013), which introduced horizontal and vertical voting to ensemble a model across training epochs.

LLBoost can be used in combination with all the above approaches to further boost their performance, since it only adjusts the last layer of a pre-trained model without requiring any training.

3 PRELIMINARIES AND METHOD

We now present the preliminaries relevant for formally introducing LLBoost. Since our method is derived from over-parameterized linear regression, we first describe this problem setting.

Noiseless Regression. Let $(X, y) \subset \mathbb{R}^{d\times n} \times \mathbb{R}^{1\times n}$ denote the training dataset where d is the number of dimensions and n is the number of samples. In noiseless linear regression there exists $w^* \in \mathbb{R}^{1\times d}$ such that $y = w^* X$. In order to estimate w^* from the data, we use gradient descent to minimize the squared loss. Formally, we solve:

$$\arg\min_{w\in\mathbb{R}^d} \frac{1}{2}\|y - wX\|^2 \tag{1}$$

using gradient descent with learning rate η, which proceeds according to the updates:

$$w^{(t+1)} = w^{(t)} + \eta(y - w^{(t)}X)X^T.$$

The following well-known theorem (Engl et al., 1996) gives the solution, $w^{(\infty)}$, for the objective (1) given by gradient descent.

Theorem 1. *Let $(X, y) \subset \mathbb{R}^{d\times n} \times \mathbb{R}^{1\times n}$ with X of rank $r < d$, let λ_{max} denote the largest eigenvalue of XX^T, and let $A^\dagger$ denote the pseudo-inverse of a matrix A. Given initialization $w^{(0)} \in \mathbb{R}^d$, gradient descent used to solve:*

$$\arg\min_{w\in\mathbb{R}^d} \frac{1}{2}\|y - wX\|^2$$

with learning rate $\eta < \lambda_{max}^{-1}$ converges to $w^{(\infty)} = w^{(0)}(I - X(X^TX)^\dagger X^T) + yX^\dagger$.

When $w^{(0)} = \mathbf{0}$ (zero initialization), $w^{(\infty)}$ is the minimum ℓ_2-norm solution for solving the linear system $wX = y$. The following property of $w^{(\infty)}$ is used to demonstrate that LLBoost does not impact training error.

Lemma 1. *Let $w^{(\infty)} = w^{(0)}(I - X(X^TX)^\dagger X^T) + yX^\dagger$. Then, $w^{(0)}(I - X(X^TX)^\dagger X^T) \perp yX^\dagger$.*

The proof follows directly by substituting in $X = U\Sigma V^T$ given by the singular value decomposition (SVD). Lemma 1 implies that LLBoost does not affect training predictions since it only ever adds a component orthogonal to the span of the training feature matrix. We primarily focus on the noiseless regression setting, but we note that our theory directly extends to the Gaussian model from Belkin et al. (2019b) and Breiman & Freedman (1983).

We now present a result from random matrix theory, which is at the basis of our results. The proof of the following lemma relies on the rotation invariance of the multivariate Gaussian distribution and is presented in Appendix B.

Lemma 2. *If $X \in \mathbb{R}^{d\times n}$ with columns $x^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{d\times d})$, then the singular vectors of X are uniformly distributed on the unit sphere $\mathcal{S}_{d-1}$. In particular, if $x \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{d\times d})$, then $\frac{x}{\|x\|}$ is uniformly distributed on the unit sphere $\mathcal{S}_{d-1}$.*

Method Overview. Given the above preliminaries, we present LLBoost in Algorithm 1. LLBoost takes in the following inputs: (1) the training feature matrix X; (2) the test feature matrix X_t; (3) the test labels y_t; (4) the last layer weights w; (5) the number of samples to consider T; and (6) the radius of the hyper-sphere to sample from γ. LLBoost begins by constructing the projection matrix $P = I - X(X^TX)^\dagger X^T$. Then, LLBoost samples $z \sim \mathcal{N}(\mathbf{0}, I)$, normalizes z and adds the perturbation γzP to w. LLBoost repeats this process and returns the perturbation that most improves the validation accuracy.

Remarks. As shown in Figure 1A, X and X_t are constructed by applying all but the last layer of the neural network to the training/test data. If X is not full rank (as is the case for neural networks that are not over-parameterized), we can create a low rank X by truncating the lower singular values of X. This correction only mildly affects training error as is shown in Section 4. For classification settings, y_t should be converted to 1-hot vectors. When the last layer has a bias, one can simply append a 1 to w and append an extra row of 1's to X and X_t. In most of our experiments, we consider up to $T = 100,000$ samples. When selecting γ, one can utilize a binary search approach starting with $\gamma = \frac{1}{\sqrt{d}}$ and either doubling or halving γ depending on whether the validation accuracy increased or decreased respectively. However, as a simple heuristic one can start with $\gamma = \frac{1}{\sqrt{d}}\|w\|$ (as is recommended by Theorem 3). Lastly, when the network has multiple outputs (as in multi-

Algorithm 1 LLBoost $(X, X_t, y_t, w, T, \gamma)$

Input: $X :=$ rank r training feature matrix; $X_t :=$ test feature matrix; $y_t :=$ test labels; $w :=$ last layer weights; $T :=$ number of samples; $\gamma :=$ radius of hyper-sphere
Output: $w_{best} :=$ adjusted last layer weights

$w_{best} = None$
$acc_{best} = 0;\ (1, d) = w.shape;\ U, \Sigma, V^T \leftarrow \text{SVD}(X)$
$P \leftarrow I_{d\times d} - UI_{r\times r}U^T$
for $t \leftarrow 1$ to T **do**
    $z \leftarrow \mathcal{N}(\mathbf{0}, I_{d\times d})$
    $z \leftarrow \frac{z}{\|z\|}$
    $w_r \leftarrow \gamma zP + w$
    $acc = GetTestAcc(X_t, y_t, w_r)$
    if $acc > acc_{best}$ **then**
        $w_{best} = w_r;\ acc_{best} = acc$
    end if
end for
return w_{best}

class problems), we let the entries of the weight matrix be i.i.d. standard normal variables and then normalize by the Frobenius norm.

LLBoost does not impact training error/accuracy. Since the perturbations constructed by LLBoost are always projected into the space orthogonal to the training features X, Lemma 1 implies that LLBoost does not alter the training predictions. This is an important aspect of our algorithm since otherwise, LLBoost could just overfit the test data.

GPU Implementation. We can easily remove the loop in Algorithm 1 by using the GPU since the dominating computations are vector-matrix products. A full implementation is provided using PyTorch (Paszke et al., 2019) and NumPy (Oliphant, 2006; Van Der Walt et al., 2011) in the footnote below[1]. In practice, computing the validation accuracy for each sample of w_r is the limiting factor in LLBoost. For large datasets, the functions used to compute the SVD and the projection matrix P need to be selected carefully. Note that with respect to SVD, the reduced SVD is sufficient for our algorithm. While for the ImageNet training features from FixResNext-101 the full SVD would require constructing a matrix with roughly 10^{12} entries, the reduced SVD only requires 10^8 entries.

4 Empirical Results

In Figure 2, we provide empirical evidence that LLBoost consistently improves the validation accuracy of state-of-the-art models on CIFAR10, ImageNet32 (Oord et al., 2016), and ImageNet. For models fine-tuned[2] on ImageNet32 or CIFAR10, we provide all training details in Appendix C. Importantly, as proven by Lemma 1, we note that the training accuracy remains unchanged after applying LLBoost.

Handling under-parameterized models. If n denotes the number of training samples, the size of the training feature matrix is $512 \times n$ for ResNets-18 and 34 while it is $2048 \times n$ for ResNet-50 and FixResNext-101 (Touvron et al., 2019). Hence ResNet-18 and 34 are over-parameterized on the training set when $n < 512$, while ResNet-50 and FixResNext-101 are over-parameterized for $n < 2048$. When the training feature matrix, X, has full rank, we truncate the lower singular values of X in order to be able to search the space orthogonal to X. Formally, if $X = U\Sigma V^T$, we construct $\widehat{X} = U\widehat{\Sigma}V^T$ where we zero out the bottom entries of Σ. When $\widehat{X}$ has rank k, we refer to it as the *rank k approximation* of X.

Our experiments are summarized as follows:

1. On ImageNet32, we fine-tune pre-trained ResNets of varying depth on subsets of varying size of two classes (Kit-Fox vs. English-Setter). There are a total of 2600 examples (1300 of each class). We then apply LLBoost to these pre-trained models and use a rank 12 approximation for the feature matrix when necessary. This set of experiments demonstrates that LLBoost improves transfer learned models on small datasets.

2. On CIFAR10, we fine-tune a pre-trained ResNet-18. We then apply LLBoost to the pre-trained network. In this setting, there are $50,000$ training examples and so the feature matrix is full rank. To make the model over-parameterized, we instead use the rank 212 approximation. Remarkably, this approximation does not impact the original training accuracy.

3. On ImageNet, we apply LLBoost to pre-trained FixResNext-101. ImageNet contains $1,281,167$ training examples and the feature matrix is again full rank. In this case, we use the rank 1000 approximation, and the original training accuracy decreases from 95.193% to 94.924% when applying the original weights to the rank 1000 approximation.

In all settings, LLBoost improves validation accuracy. In addition, as proven by Lemma 1, there is no impact on training accuracy in the over-parameterized setting. In under-parameterized settings such as FixResNext-101 on ImageNet, our experiments indicate only minor decrease in training accuracy when using a low rank approximation to X. In Appendix D, we additionally present: (1) the difference in ℓ_2/Frobenius norm between the LLBoost weights and the original weights; (2) the run-time of LLBoost; (3) the choice of γ used in the experiments. In Appendix F, we discuss how

[1] `https://anonymous.4open.science/r/b11fa900-efbf-45ae-aee8-a264d954cb51/`

[2] By fine-tuning, we mean training a pre-trained model on a new dataset. This is discussed further in the transfer learning tutorials provided by PyTorch (Paszke et al., 2019).

Dataset	Model	Train/Val. Acc. (Original)	Train/Val. Acc. (Ours)	Train/Val. Acc. (Standard Normal)
100 Ex. ImageNet32	ResNet-18	100%/80%	100%/**84%**	88%/81%
2600 Ex. ImageNet32	ResNet-18*	99.96%/95%	99.96%/**97%**	99.96%/95%
100 Ex. ImageNet32	ResNet-34	100%/85%	100%/**87%**	74%/82%
2600 Ex. ImageNet32	ResNet-34*	99.77%/95%	99.77%/**98%**	99.65%/97%
100 Ex. ImageNet32	ResNet-50	100%/83%	100%/**89%**	84%/83%
200 Ex. ImageNet32	ResNet-50	100%/87%	100%/**93%**	79.5%/87%
500 Ex. ImageNet32	ResNet-50	99.6%/91%	99.6%/**93%**	93.2%/92%
800 Ex. ImageNet32	ResNet-50	100%/94%	100%/**98%**	97.75%/96%
1000 Ex. ImageNet32	ResNet-50	100%/93%	100%/**97%**	98.5%/95%
2000 Ex. ImageNet32	ResNet-50	99.85%/95%	99.85%/**97%**	98.8%/97%
2600 Ex. ImageNet32	ResNet-50*	99.88%/95%	99.88%/**99%**	99.69%/98%
CIFAR10	ResNet-18*	99.99%/95.05%	99.99%/**95.25%**	42.53%/36.46%
ImageNet	FixResNext-101*	94.924%/86.26%	94.924%/**86.34%**	.57%/.56%

Figure 2: LLBoost consistently improves the performance of pre-trained neural networks across a variety of experiments on ImageNet32, CIFAR10, and ImageNet without impacting training accuracy. The last column indicates that applying a standard perturbation instead of LLBoost to the last layer of a neural network can significantly impact training and validation accuracy. $*$'s indicate that the training feature matrix X was full rank, and thus, we used a low rank approximation of X for LLBoost. See Appendix F for a description of how the rank was chosen, and an analysis of how low rank approximations affect train/validation accuracy.

the rank was chosen when approximating full rank feature matrices, and provide an analysis of how low rank approximations affect train/validation accuracy.

Remarks. In the last column of Figure 2, we also provide the performance of perturbing the last layer of a neural network with a usual distribution such as the standard normal. Note that such a standard normal perturbation significantly impacts the original training accuracy; in fact for larger datasets such as ImageNet, the training and validation accuracies drop below 1%. Hence, without the projection operator, perturbations to the last layer can be devastating to network performance. In Appendix E, we also demonstrate that when including the projection operator but using standard normal initialization (instead of uniform on the hyper-sphere as done in LLBoost) can similarly reduce validation accuracy. While we have thus far concentrated on validation accuracy, in Appendix G we show that in the setting of training, validation, and test splits, LLBoost not only improves validation accuracy, but also improves test accuracy without impacting training accuracy.

Improvement over Random Seed Search. We demonstrate in Figure 3 that LLBoost provides a significant computational advantage over random seed search. In particular, in Figure 3 we compare the performance of fine-tuning pre-trained models on ImageNet32 and CIFAR10 using random seeds. Naturally, LLBoost can be applied to boost the performance of all these models. Moreover, Figure 3 illustrates that LLBoost can boost the performance of a model trained using a poor random seed to that of a model trained using a well-selected seed. Since LLBoost only takes a fraction of the training time, these experiments identify LLBoost as a computationally efficient alternative to random seed search.

5 Theoretical Results

We now present our derivation and analysis of LLBoost for over-parameterized noiseless linear regression. In the noiseless regression setting, we assume that the labels $y \in \mathbb{R}^{1\times n}$ are generated by the product of $w^* \in \mathbb{R}^{1\times d}$ and data $X \in \mathbb{R}^{d\times n}$. Let $\widehat{w}_r = w^{(0)}(I - X(X^TX)^\dagger X^T) + yX^\dagger$ and $\widehat{w} = yX^\dagger$ denote interpolating solutions for the linear system $wX = y$ given by gradient descent from initialization $w^{(0)}$ and $\mathbf{0}$ respectively. In this section we establish the following:

1. Generalization bounds on w_r.

Seed	Val Acc.
200	94%
3	94%
70	97%
25	94%
500	95%
LLBoost on Seed 200	**98%**

(A) 800 Ex. ImageNet32 Classes 1, 2 ResNet-50

Seed	Val Acc.
200	95%
3	95%
70	95%
25	97%
500	95%
LLBoost on Seed 3	**97%**

(B) 2000 Ex. ImageNet32 Classes 1, 2 ResNet-50

Seed	Val Acc.
200	95.05%
3	94.98%
70	95.06%
25	95.14%
500	95.1%
LLBoost on Seed 3	**95.18%**

(C) CIFAR10 ResNet-18

Figure 3: LLBoost provides a computational advantage over random seed search. While LLBoost not only improves the performance of every pre-trained model above, our method boosts the performance of a model trained using a poor random seed to at least that of a model trained using a well-selected seed at a fraction of the computational cost.

2. There exist infinitely many $w^{(0)}$ such that $\|\widehat{w_r} - w^*\| < \|\widehat{w} - w^*\|$, i.e. there exist infinitely many random initializations that out-perform the minimum norm solution. In particular, we show that even if $\|w^{(0)} - w^*\|$ is large, $\|\widehat{w_r} - w^*\|$ can be arbitrary close to 0.
3. Sampling $w^{(0)}$ uniformly on the hyper-sphere of appropriate radius leads to $\widehat{w_r}$ out-performing the minimum norm solution with constant probability.

The following proposition (with proof in Appendix H) compares the generalization error of the random initialization solution, $\widehat{w_r}$ and the minimum norm solution, $\widehat{w}$.

Proposition 1. *Let $X = U\Sigma V^T$ denote the SVD of X and let $\Sigma^\perp = I - \Sigma\Sigma^\dagger$. Then,*

$$(a)\ \ \|\widehat{w} - w^*\| = \|w^* U \Sigma^\perp U^T\|,$$
$$(b)\ \ \|\widehat{w}_r - w^*\| = \|(w^{(0)} - w^*) U \Sigma^\perp U^T\| \leq \|w^{(0)} - w^*\|.$$

By using Proposition 1 and extending the proof technique of Belkin et al. (2019b), we establish the following generalization bounds for the solution starting from initialization $w^{(0)}$.

Theorem 2. *Assuming data $x, x^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{d\times d})$ for $i \in [n]$, then*

$$(1)\ \ \mathbb{E}_{x,X}[(y - \widehat{w}x)^2] = \mathbb{E}_X[\|\widehat{w} - w^*\|^2] = \|w^*\|^2 \left(1 - \frac{n}{d}\right),$$
$$(2)\ \ \mathbb{E}_{x,X}[(y - \widehat{w}_r x)^2] = \mathbb{E}_X[\|\widehat{w}_r - w^*\|^2] = \|w^{(0)} - w^*\|^2 \left(1 - \frac{n}{d}\right).$$

The proof is presented in Appendix I. From Theorem 2, the out of sample performance of the solution initialized at $w^{(0)}$ is governed by the distance between $w^{(0)}$ and w^*, which matches intuition. While this result is in expectation over the data drawn from an isotropic Gaussian, we now prove the remarkable fact that even if $\|w^{(0)} - w^*\| = c_1$ is large, $\|\widehat{w_r} - w^*\|$ can be any value between 0 and c_1.

Proposition 2. *Let r denote the rank of the data matrix X and let $c_1 \geq c_2 \geq 0$. Then there exists $w^{(0)}$ such that $\|w^{(0)} - w^*\| = c_1$ and $\|\widehat{w}_r - w^*\| = c_2$.*

The full proof is presented in Appendix J. Importantly, Proposition 2 provides the following intuition around the benefit of random initialization in over-parameterized models. When X has rank r for r small, the space orthogonal to X is large with dimension $d-r$. There is thus a priori no reason for the true w^* to lie in the span of the training data. Hence, by sampling $w^{(0)}$ to add a term orthogonal to X we can expect to obtain a boost in performance over the minimum norm solution. The following proposition and theorem present a sampling method for $w^{(0)}$, which provably provides a boost over the minimum norm solution with constant probability.

Proposition 3. *Let $\mathcal{U}_d$ represent the uniform distribution on the sphere. Assume that the data $x, x_i^{(2)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{d\times d})$ for $i \in [n]$ and that $w^{(0)} \sim \mathcal{U}_d$. Then,*

$$\mathbb{P}_{w^{(0)}}(\mathbb{E}_{x,X}[(y - \widehat{w}_r x)^2] \leq \mathbb{E}_{x,X}[(y - \widehat{w}x)^2]) = \frac{\Gamma\left(\frac{d}{2}\right)}{\sqrt{\pi}\Gamma\left(\frac{d-1}{2}\right)} \int_0^{\phi} \sin^{d-2}\theta d\theta, \tag{2}$$

where $\phi = \cos^{-1}\left(\frac{1}{2\|w^*\|}\right)$ and $\Gamma(x)$ denotes the Gamma function, which satisfies $\Gamma(x) = (x-1)\Gamma(x-1)$ with $\Gamma(1) = 0$ and $\Gamma(\frac{1}{2}) = \sqrt{\pi}$.

The proof is given in Appendix K. The benefit of Proposition 3 is that we can lower bound the right hand side of Equation (2) by a constant greater than 0. This is computed explicitly in Theorem 3.

Theorem 3. *Let $Z_{w^{(0)}} = \mathbb{E}_{x,X}[(y - \widehat{w}x)^2] - \mathbb{E}_{x,X}[(y - \widehat{w}_r x)^2]$ denote the difference in expected generalization error between the minimum ℓ_2-norm solution and the random initialization solution. If $w_0 \sim \mathcal{U}_d(\gamma)$, $\epsilon \geq 0$, and $\kappa = \frac{\gamma + \frac{\epsilon}{\gamma}}{2\|w^*\|}$, then*

$$\frac{1}{\pi}\cos^{-1}(\kappa) - \frac{\sqrt{d(d-1)}(\sqrt{2(d-4)}+1)\kappa\left(\sqrt{1-\kappa^2}\right)}{2\sqrt{\pi}(d-2)} \leq \mathbb{P}_{w^{(0)}}\left(Z_{w^{(0)}} \geq \epsilon\left(1 - \frac{n}{d}\right)\right) \leq \frac{1}{2}$$

In particular, if $\|w^*\| = 1$, $\gamma = \frac{1}{\sqrt{d}}$, and $\epsilon = \frac{1}{d}$, then $\kappa = \frac{1}{2\sqrt{d}}$ and

$$\frac{1}{2} - \frac{1}{2\sqrt{2\pi}} \leq \lim_{d\to\infty} \mathbb{P}_{w^{(0)}}\left(Z_{w^{(0)}} \geq \frac{1}{d}\left(1 - \frac{n}{d}\right)\right) \leq \frac{1}{2} \tag{3}$$

Remarks. Theorem 3 is at the core of LLBoost; it implies that the probability of $\widehat{w_r}$ having lower expected generalization error than $\widehat{w}$ is constant. Note that based on Equation (3) this constant is lower bounded by roughly .3 (meaning we get a 30% chance of improvement). Importantly, Theorem 3 can be trivially extended to demonstrate that LLBoost can reduce the generalization error of any interpolating solution with constant probability (provided that the generalization error is not already 0). We also remark that the lower bound in Equation (3) is nearly tight: in practice, this constant is roughly .31. Lastly, note that all of the above theory holds also for the regression setting with multiple outputs by replacing the usual ℓ_2 norm with the Frobenius norm.

While the full proof of Theorem 3 is presented in Appendix L, we here provide a sketch. The proof provides an approximation for the integral on the right hand side of Equation (2) by using Gautschi's Inequality (Gautschi, 1959) and integration by parts. Note that while it may seem at first that ϕ should be uniformly distributed when $w^{(0)}$ is uniformly distributed on the sphere (and thus the probability should go to 1/2 as d goes to ∞), it is in fact not true that ϕ is uniformly distributed when $w^{(0)}$ is uniformly distributed, which is why a more precise argument is required.

6 Discussion

In this work, we presented LLBoost, a theoretically-grounded, computationally-efficient method to improve the validation accuracy of pre-trained neural networks without impacting training accuracy. Through a variety of experiments on ImageNet32, CIFAR10, and ImageNet, we demonstrated that our method is practically relevant and can be used to boost the performance of state-of-the-art pre-trained models. A standard method to improve the performance of neural networks is random seed search. We showed that LLBoost provides a significant computational advantage over random seed search and can even boost the performance of a model trained on a poorly selected seed to that of a model trained on a well-selected seed. Thus, LLBoost is a computationally efficient alternative to random seed search. Lastly, we provided theoretical footing for our method by showing that LLBoost provably reduces the test error of any interpolating solution in over-parameterized linear regression. An interesting direction of future work is to identify alternate sampling schemes for LLBoost (other than uniform on the hyper-sphere) that provably yield a greater increase in validation accuracy without decreasing training accuracy.

REFERENCES

Peter L. Bartlett, Philip M. Long, Gábor Lugosi, and Alexander Tsigler. Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*, 2020.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019a.

Mikhail Belkin, Daniel Hsu, and Ji Xu. Two models of double descent for weak features. *arXiv:1903.07571*, 2019b.

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012. URL `http://arxiv.org/abs/1206.5533`.

K Bibas, Y. Fogel, and M. Feder. A new look at an old problem: A universal learning approach to linear regression. *arXiv preprint arXiv:1905.04708*, 2019.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

Leo Breiman and David Freedman. How many variables should be entered in a regression equation? *Journal of the American Statistical Association*, 78(381):131–136, 1983.

Heinz Werner Engl, Martin Hanke, and Andreas Neubauer. *Regularization of inverse problems*, volume 375. Springer Science & Business Media, 1996.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why Does Unsupervised Pre-training Help Deep Learning? *Journal of Machine Learning Research (JMLR)*, 11:625–660, 2010.

Yoav Freund and Robert Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, 1999.

Walter Gautschi. Some Elementary Inequalities Relating to the Gamma and Incomplete Gamma Function. *Journal of Mathematics and Physics*, 38(1-4):77–81, 1959.

Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures. *arXiv preprint arXiv:1608.06037*, 2016.

Trevor Hastie, Andrea Montanari, Saharon Rosset, and Ryan J Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation. *arXiv preprint arXiv:1903.08560*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft Hopcroft, and Kilian Q. Weinberger. Snapshot ensembles: Train 1, get m for free. In *International Conference on Learning Representations (ICLR)*, 2017.

Like Hui and Mikhail Belkin. Evaluation of Neural Architectures Trained with Square Loss vs Cross-Entropy in Classification Tasks. *arXiv preprint arXiv:2006.07322*, 2020.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Shengquiao Li. Concise formulas for the area and volume of a hyperspherical cap. *Asian Journal of Mathematics and Statistics*, 4(1):66–70, 2011.

Marco Loog, Alexander Mey, Jesse H. Krijthe, and David M. J. Tax. A brief prehistory of double descent. *Proceedings of the National Academy of Sciences*, 117(20):10625–10626, 2020.

Partha P. Mitra. Understanding overfitting peaks in generalization error: Analytical risk curves for l_2 and l_1 penalized interpolation. *arXiv preprint arXiv:1906.03667*, 2019.

Vidya Muthukumar, Kailas Vodrahalli, Vignesh Subramanian, and Anant Sahai. Harmless interpolation of noisy data in regression. *IEEE Journal on Selected Areas in Information Theory*, 1(1): 67–83, 2020.

Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuglu. Pixel recurrent neural networks. In *International Conference on Machine Learning (ICML)*, 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Fei-Fei Li. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition*, 2015.

Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

Jingjing Xie, Bing Xu, and Zhang Chuang. Horizontal and vertical ensemble with deep representation for classification. In *ICML Workshop on Representation Learning*, 2013.