Bypassing Stationary Points in Training Deep Learning Models

Jaeheun Jung, Donghun Lee

***Abstract*—Gradient-descent based optimizers are prone to slowdowns in training deep learning models, as stationary points are ubiquitous in the loss landscape of most neural networks. We present an intuitive concept of bypassing the stationary points and realize the concept into a novel method designed to actively rescue optimizers from slowdowns encountered in neural network training. The method, Bypass pipeline, revitalizes the optimizer by extending the model space and later contracts the model back to its original space with function-preserving algebraic constraints. We implement the method into the Bypass algorithm, verify that the algorithm shows theoretically expected behaviors of bypassing, and demonstrate its empirical benefit in regression and classification benchmarks. Bypass algorithm is highly practical, as it is computationally efficient and compatible with other improvements of first-order optimizers. Also, bypassing for neural networks leads to new theoretical research such as model-specific bypassing and neural architecture search.**

***Index Terms*—bypassing, stationary points, neural network, gradient descent**

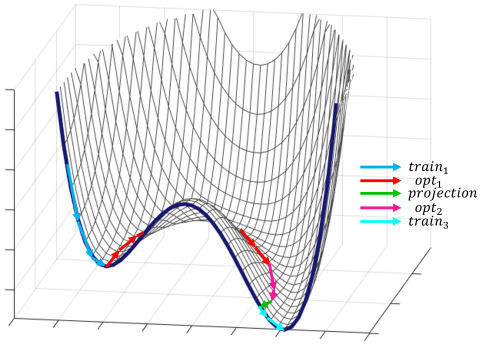


Fig. 1. The intuitive plot of bypass pipeline working in low dimensional example. The dark blue-colored curve and the surface including the curve represents the original and extended loss landscape. The colored directions represent the movement of the parameter in each phase of bypass pipeline.

I. Introduction

FIRST-ORDER optimization methods, such as stochastic gradient descent (SGD) and its modern improvements like Adam optimizer [1], are *de facto* methods to train deep neural network models. Plenty of previous works analyzed convergence of SGD algorithm and its variants [2]–[4], and also in practice, most gradient-based training methods with well-chosen hyperparameters will eventually find an acceptable model [5], [6]. However, the training loss reduction during the training process is prone to slowdowns near any stationary points – either the correct optimal point or any other suboptimal stationary points such as local minima or saddle points – resulting in real-time delays.

In particular, there are large number of saddle points corresponding to the suboptimal solutions in the loss function optimization with respect to a neural network model parameter, especially in a high dimensional case [7]. Hence, efficient algorithms to escape saddle points can alleviate potential delays in neural network training due to saddle points. For example, gradient-based saddle point escaping algorithms generally exploit weight perturbation [8]–[11], which aim to find second-order stationary points, including local minima. Such algorithms can be seen as making passive escapes from saddle points, since what they do directly is to invoke random sampling which may or may not give the wanted escape. To escape saddle points actively, it is generally required to compute the Hessian at such points to find the correct loss-reducing direction; however, this is often impractical in neural network models due to the large number of model parameters.

Local minima can also trap first-order optimizations without compatible escaping tricks at suboptimal points. It is reasonable to expect many local minima in most deep learning models in practice [12], [13], since only a small group of neural network models are free from suboptimal local minima under specific assumptions, such as deep linear networks [14]–[18] or shallow networks [15], [19] with square loss function. Local minima are troublesome in neural network training, because once gradient-based optimizers actually get trapped at a local minimum, there are not many options to escape from it: usual choices are random restarting or random weight perturbation. Both are making passive escapes, as they exploit randomness in a way unrelated to the loss function landscape parametrized by the current model parameter.

Therefore, it is desirable to have an efficient method that 1) *actively* guides first-order optimization methods to escape the slowdown near a stationary point, 2) works robustly in the vicinity of both local minima and saddle points, and 3) does not require additional checks to characterize the vicinity in which the optimizer is trapped before using the method. In this paper, we propose a novel method that complements first-order optimizers for neural network training, based on the key idea of "bypassing" the vicinity of a stationary point, which is the main source of the optimizer slowdown. The intuition of our method is visualized in Fig. 1, where the gradient-based

J.Jung is with Graduate school of Mathematics, Korea University, Seoul, South Korea.

D.Lee is with Department of Mathematics, Korea University, Seoul, South Korea, and is supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2020R1G1A1102828).

optimizer, trapped in the left-side suboptimal local minima in the paper-facing vertical space, makes a "bypass" through an extended space for a while, returns to the original space and eventually reaches the right-side optimal point.

First, in Section II, we present the theoretical preliminaries and the concept of bypassing in neural network model training, including the explanations on opt_1, opt_2 and other steps shown in Fig. 1. In Section III, we present the Bypass pipeline, a realization of bypassing with theoretical designs to be compatible with neural network training tasks. We then present Bypass algorithm, our implementation of the Bypass pipeline in Section IV, with a line-by-line walk-through and detailed commentaries on theoretically expected behaviors. Then, in Section V, we provide empirical evaluation results from two benchmark tasks, designed to verify theoretically expected behaviors and demonstrate the empirical benefits of applying bypassing in neural network training tasks. Additionally, in Section VI we present several ways of customizing the bypass algorithm and corresponding experimental results showing the effect of modification. Lastly, discussion on bypassing and future research directions are found in Section VII, and the conclusion follows in Section VIII.

Our Contribution

- We propose bypass pipeline, a novel method to bypass the vicinity of a suboptimal stationary point using extension-and-contraction strategy.
- We present a particular instance of bypass pipeline for neural networks and a constructive method to realize it with learnable activation.
- As a core component for bypass pipeline for neural networks, we propose novel contraction method to bring the extended model back to the original model space prior to extension.
- We present a detailed algorithm that implements the bypass pipeline, verify its theoretically expected behaviors and demonstrate its empirical benefit in both regression and classification tasks.

II. Theoretical Aspects of Bypassing

The key idea of bypassing is to show the trapped first-order optimizer a *new* loss-reducing direction, in which the "new" direction *transcends the original parameter space* by extending the loss landscape with larger dimension. The concept of bypassing begins with the presentation of the new direction which was not viable in the original loss landscape.

This step is intuitively depicted in Fig. 1 as "moving into the page", which is not present in the original loss function – the W-shaped dark blue-colored curve facing the page.

Once the optimizer realize new direction in extended space, it will be able to continue the training by following gradient signal as it would no longer trapped in vicinity of stationary point.

After getting advantage in extended space, it would be necessary to restore the model and loss landscape to prior one, to avoid violation on objective optimization problem.

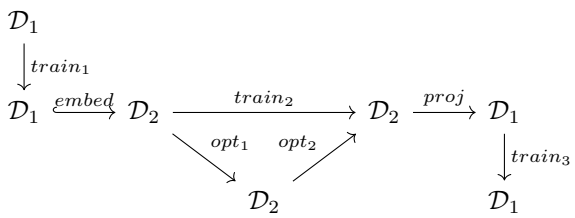


Fig. 2. Flowchart of the bypass pipeline. When $train_1$ arrives near the stationary point, the horizontally illustrated processes($embed, opt_1, opt_2$ and $proj$) allows to bypass the stationary point by finding learning trajectory in extended space. Finally, the $train_3$ in original parameter space $\mathcal{D}_1$ continues the training.

Based on those intuitive concepts, we propose **Bypass pipeline** for training deep learning models through first-order gradient-based optimizers in this section, with theoretical preliminaries.

A. Intuition of Bypassing

A (neural network) model is considered to be a parametrization of (continuous) functions denoted as:

$$\begin{aligned} \varphi_1 : \mathcal{D}_1 &\longrightarrow \mathcal{F} \\ \theta &\longmapsto f_\theta \end{aligned} \tag{1}$$

where $\mathcal{D}_1$ is real parameter space and $\mathcal{F}$ is a function space.

We define functional $\mathcal{L} : \mathcal{F} \longrightarrow \mathbb{R}$ that measures the difference between how the current parametrized function approximates the given data and the best possible function or distribution using the given data. Then, training a model φ_1 can be seen as solving the following minimization problem:

$$\underset{\theta \in \mathcal{D}_1}{\text{minimize}} \quad \mathcal{L}_1(\theta), \tag{2}$$

where $\mathcal{L}_1 = \mathcal{L} \circ \varphi_1$. Almost all methods of training modern deep learning models relies on first-order gradient based optimization methods.

Now, we introduce another model $\varphi_2 : \mathcal{D}_2 \to \mathcal{F}$, which is an extension of the first model φ_1 such that $Im(\varphi_1) \subseteq Im(\varphi_2)$. We use ω to denote elements of $\mathcal{D}_2$ in rest of the paper, in order to distinguish it from $\theta \in \mathcal{D}_1$,

Defining $\mathcal{L}_2 = \mathcal{L} \circ \varphi_2$, we recast (2) as the following constrained minimization problem:

$$\begin{aligned} \underset{\omega \in \mathcal{D}_2}{\text{minimize}} \quad & \mathcal{L}_2(\omega) \\ \text{subject to} \quad & \omega \in \varphi_2^{-1}(Im(\varphi_1)) \end{aligned} \tag{3}$$

This reformulation is not only equivalent to (2) in sense of global minima but also advantageous when the local behavior of a first-order optimizer in $\mathcal{D}_1$ is problematic. For example, local behaviors in solving (2) such as slow convergence speed or problematic local geometry is not necessarily preserved when a comparable optimizer operates in the extended space $\mathcal{D}_2$ as shown in (3).

B. Bypass pipeline

The key idea of the Bypass pipeline is "bypassing" via an extended parameter space, which 1) sends the parameter vector to an extended parameter space, 2) trains it within the extended space, and 3) contracts the model (or the extended parameter space) to original one. Precise description of each step follows in this subsection.

To send the parameter vector $\theta \in \mathcal{D}_1$ to the extended parameter space, we need to define the embedding map as follows:

Definition 1.

$$\begin{aligned} embed : \mathcal{D}_1 &\longrightarrow \mathcal{D}_2 \\ \theta &\longmapsto \omega \end{aligned} \tag{4}$$

such that $\varphi_2 \circ embed = \varphi_1$. This embedding map glues the $\mathcal{D}_1$ to $\mathcal{D}_2$ and hence generate attached (topological) space. We identify every points θ in original space $\mathcal{D}_1$ as a point ω in $\mathcal{D}_2$ via $embed$, with $\varphi_1(\theta) = \varphi_2(\omega)$.

As conceptually illustrated in Fig. 1 and described as a flowchart in Fig. 2, our Bypass pipeline starts with model φ_1 with initialization θ^0, 1) trains the model using given data until the convergence to get θ^*, 2) extends the model by embedding the model parameter θ^* to the extended parameter $\omega_0 = embed(\theta^*)$ and start to solve constrained minimization problem (3) by $train_2$ and get ω^* such that $\varphi_2(\omega^*) \in Im(\varphi_1)$, and finally 3) brings the extended parameter ω^* to $\overline{\theta} = proj(\omega^*)$ which is in $\mathcal{D}_1$ using projection map $proj$, with existence assumption and definition below, and continue to train with $train_3$ in original parameter space.

Definition 2. *We define a projection map $proj$ given original model φ_1 and its extended model φ_2 as*

$$\begin{aligned} proj : \varphi_2^{-1}(Im(\varphi_1)) &\longrightarrow \mathcal{D}_1 \\ \omega &\longmapsto \theta \end{aligned} \tag{5}$$

which satisfies $\varphi_2 = \varphi_1 \circ proj$ on $\varphi_2^{-1}(Im(\varphi_1))$

Notably, $\overline{\theta} = proj(\omega^*)$ need not be a stationary point of $\mathcal{L}_1$ in $\mathcal{D}_1$, so it is possible to continue the training with respect to proper training schedule. We denote this continual training as $train_3$.

To implement the Bypass pipeline given the original model φ_1, the following components are required in computable form:

1) Extended model $\varphi_2 : \mathcal{D}_2 \to \mathcal{F}$ and $embed$ function
2) Defining equation of $\varphi_2^{-1}(Im(\varphi_1))$ to construct a suitable projection map
3) Constrained optimization algorithm $train_2$ to find ω^* in $\varphi_2^{-1}(Im(\varphi_1))$

We discuss the theoretical requirements of these components in the next section with implemenation method in neural networks.

III. Bypass pipeline for neural networks

In this section, we focus on neural network models and describe a method to implement the three required components of the Bypass pipeline.

Definition 3. *We define φ_1 be a deep neural network model that contains two consecutive fully-connected layers, which are considered as a submodule. For such a submodule, we use the following notations:*

1) *Let $\overline{\varphi_1}$ be a submodule that consists of two fully-connected layers.*
2) *Let s_I be the dimension of input vector to the submodule, and s_W, s_A the numbers of nodes on each layer, respectively.*
3) *We denote σ_W and σ_A be the activation functions of each layer.*
4) *Let W, b_W be the weight and bias parameters of the first layer and A, b_A be the weight and bias of the second layer of the submodule. Then the submodule can be parametrized as*

$$\overline{\varphi}_1(W, b_W, A, b_A) : x \mapsto \sigma_A(b_A + A\sigma_W(b_W + Wx)). \tag{6}$$

Many deep learning models can be φ_1. For example, fully connected feedforward networks are in itself covered, and many modern deep learning models with prediction head are covered.

A. Model extension and embedding

To transform a deep learning model φ_1 to an extended model φ_2, we replace a submodule $\overline{\varphi_1}$ of φ_1 with an extended submodule $\overline{\varphi_2}$. The parameters θ of φ_1 are represented as $\theta = (\tilde{\theta}, W, b_W, A, b_A)$, where $\tilde{\theta}$ contains all parameters of φ_1 not associated to submodule $\overline{\varphi_1}$.

While extending $\overline{\varphi}_1$ into $\overline{\varphi}_2$ with additional parameters D, we need to consider following requirements, which represents existence of $embed$ and loss-reducing direction in $\mathcal{D}_2$.

1) Given θ, $\exists D : \varphi_2(D, \theta) = \varphi_1(\theta)$
2) $\nabla_\theta \mathcal{L}_1 = 0$ doesn't imply $\nabla_\omega \mathcal{L}_2 = 0$

There are many ways to implement $\overline{\varphi}_2$ with those requirements, such as adding new neurons or layers, but we propose to utilize a specific class of learnable activations

$$\psi_W(x_j) = \delta_j x_j + \sigma_W(x_j), \tag{7}$$

where x_j is jth entry of input vector $\mathbf{x}$ to the original submodule $\overline{\varphi_1}$, and δ_j is corresponding coefficient parameter found in the extended submodule $\overline{\varphi_2}$. This can be represented concisely as:

$$\psi_W(\mathbf{x}) = D\mathbf{x} + \sigma_W(\mathbf{x}) \text{ for } D = diag(\delta_1, \cdots, \delta_{s_W}). \tag{8}$$

The extension via ψ_W is the simplest one in learnable activations which satisfy the requirements. With unspecified learnable activation $h(D, x)$, the first requirement is now have following form:

$$\exists D \forall x : h(D, x) = \sigma_W(x) \tag{9}$$

or equivalently $\exists D : h'(D, x) = 0$ with $h' = h - \sigma_W$. Due to second requirement the h' must include x-valued term and hence the simplest choice here is Dx, which corresponds to ψ_W. This construction was directly motivated from the concept

of homotopy from σ_W to another function which gives an extension, since $h(0,x) = \sigma_W(x)$ and extended model with $h(D,x)$ and $D \neq 0$ will lie outside of $Im(\varphi_1)$. Note that any other choice of h' with those requirements can induce a proper bypass pipeline.

Given an extension ψ_W, the $embed$ map can be defined by setting $D = 0$:

$$\begin{aligned} embed : \mathcal{D}_1 &\longrightarrow \mathcal{D}_2 \\ \theta &\longmapsto (\theta, 0) = \omega_0 \end{aligned} \tag{10}$$

Since ψ_W is equivalent to σ_W when D is given by zero, $\varphi_1 \circ embed = \varphi_2$. Similar to θ, we denote $\omega = (\tilde{\theta}, W, b_W, A, b_W, D)$ to represent the extended parameter.

B. Comeback constraint and $proj$

In general, the constrained optimization problem (3) is hard to solve due to its unknown constraint set $\varphi_2^{-1}(Im(\varphi_1))$. To avoid the deadlock, we refine the problem with smaller constraint set which is algebraic and strictly larger than $embed(\mathcal{D}_1)$. The following theorem provides a method to refine the problem.

Theorem 1.

$$embed(\mathcal{D}_1) \subsetneq V(ADW) \subseteq \varphi_2^{-1}(Im(\varphi_1)) \tag{11}$$

for

$$V(ADW) := \{\omega \in \mathcal{D}_2 : ADW = 0\}. \tag{12}$$

recall that $\omega = (\tilde{\theta}, W, b_W, A, b_A, D)$.

Proof. Since $embed(\mathcal{D}_1) = V(D)$, first inequality is trivial because $D = 0$ implies $ADW = 0$ in matrix multiplication.

For the second inequality, consider the logits provided by the second activation σ_A. It would be given with

$$\begin{aligned} &\overline{\varphi_2}(W, b_W, A, b_A, D)(x) \\ &= \sigma_A(b_A + A\psi_W(b_W + Wx)) \\ &= \sigma_A(b_A + AD(b_W + Wx) + \sigma_W(b_W + Wx)) \\ &= \sigma_A(b'_A + \sigma_W(b_W + Wx)) \\ &= \overline{\varphi_1}(W, b_W, A, b'_A)(x) \end{aligned} \tag{13}$$

where $b'_A = b_A + ADb_W$. This shows that $\omega \in \varphi_2^{-1}(Im(\varphi_1))$ if $ADW = 0$. □

With Theorem 1, we induce the refined minimization problem:

$$\begin{aligned} \underset{\omega \in \mathcal{D}_2}{\text{minimize}} \quad & \mathcal{L}_2(\omega) \\ \text{subject to} \quad & \omega \in V(ADW) \end{aligned}, \tag{14}$$

and the corresponding projection map $proj$ as a degree-3 polynomial map from the proof.

$$\begin{aligned} proj : V(ADW) &\longrightarrow \mathcal{D}_1 \\ (\tilde{\theta}, W, b_W, A, b_A, D) &\longmapsto (\tilde{\theta}, W, b_W, A, b_A + ADb_W) \end{aligned} \tag{15}$$

Since the $proj$ is defined to be the polynomial map, this can be easily extended to entire $\mathcal{D}_2$, and hence we can apply the projection when ADW is not zero but close to $V(ADW)$.

Note that the minimization problem (14) is also equivalent to the original minimization problem (2) in sense of global minima too.

C. Constrained optimization $train_2$

To complete the Bypass pipeline, we need an algorithm to solve the constrained optimization problem (14):

$$\begin{aligned} \underset{\omega}{\text{minimize}} \quad & \mathcal{L}_2(\omega) \\ \text{subject to} \quad & ADW = 0 \end{aligned}, \tag{16}$$

as $train_2$ process introduced in Fig. 2.

We propose to split the $train_2$ into two parts, opt_1 and opt_2, as shown in Fig. 2, that correspond to unconstrained optimization and constrained optimization, respectively. Since the gradient $\nabla_\omega \mathcal{L}_2(\omega_0)$ is nonzero, it is expected that the unconstrained optimization opt_1 can reduce the loss value with proper step size. After applying opt_1, the parameters $opt_1(\omega)$ will not be in $\varphi_2^{-1}(Im(\varphi_1))$ in general and thus we construct opt_2 to let ω move toward $V(ADW)$.

To construct opt_2, we propose to use a modified loss function parametrized by new hyperparameter $\gamma(t)$, which will be updated during the training:

$$\bar{\mathcal{L}}(\omega_t) = \mathcal{L}_2(\omega_t) + \gamma(t)\|ADW\|, \tag{17}$$

with adaptive $\gamma(t)$ such that $\lim_{t\to\infty} \gamma(t) = \infty$ where t represents the number of training steps.

When opt_2 is run with a gradient-descent based optimizers, the parameters ω_t in $train_2$ approaches $V(ADW)$ as the gradient of second term $\gamma(t)\|ADW\|$ gives direction toward $V(ADW)$, the set of parameters whose $ADW = 0$ as defined in Theorem 1. This behavior is guaranteed because $\|ADW\|$ only has the global minimum 0 (shown in the following Theorem 2), as long as $\gamma(t)$ is kept large enough to cancel out the effect of $\mathcal{L}_2$ term.

Theorem 2. *For real matrices* $A \in \mathbb{R}^{m\times n}, W \in \mathbb{R}^{n\times l}$ *and diagonal matrix* $D \in \mathbb{R}^{n\times n}$ *such that* $ADW \neq 0$*, then*

$$\nabla\|ADW\| \neq 0. \tag{18}$$

Proof. Suppose that there exist $(\overline{A}, \overline{D}, \overline{W})$ such that $\nabla\|ADW\|(\overline{A}, \overline{D}, \overline{W}) = 0$ but $\overline{A}\overline{D}\overline{W} \neq 0$. Define $f(A, D, W) = \|ADW\|^2$ then

$$\nabla f = 2\|ADW\|\nabla(\|ADW\|) \text{ if } \|ADW\| \neq 0 \tag{19}$$

and hence

$$\nabla f(\overline{A}, \overline{D}, \overline{W}) = 0. \tag{20}$$

On the other hand, it follows from $f(A, D, W) = \sum_s^m \sum_t^l (ADW)_{s,t}^2$ that

$$\nabla f = (ADWW^TD, I\odot(WW^TDA^TA), DA^TADW) \tag{21}$$

for element-wise product $\odot$.

Therefore we have

$$ADWW^TD^TA^T = (ADW)(ADW)^T = 0 \tag{22}$$

on $(\overline{A}, \overline{D}, \overline{W})$ because D is diagonal and (20).

Since all matrices are real, we conclude $\overline{A}\overline{D}\overline{W} = 0$ which contradicts the assumption. Therefore, if $ADW \neq 0$ then $\nabla\|ADW\| \neq 0$. □

Therefore, opt_2 on ω_t will ensure the following condition asymptotically:

$$\omega^* = \lim_{t\to\infty} \omega_t \in V(ADW) \subseteq \varphi_2^{-1}(Im(\varphi_1)). \quad (23)$$

In practice, the trajectory of the first-order gradient based optimizers may become unstable when large $\gamma(t)$ is used, for example, due to increased step size if the learning rate of the optimizer is fixed. Using adaptive optimizers, such as Adam [1], helps mitigating this issue.

IV. Bypassing algorithm

Algorithm 1 Bypass Algorithm

1: **Input** $\theta^* = (\tilde{\theta}^*, W, b_W, A, b_A), \lambda, \lambda', \mathcal{L}_2, \epsilon, n, \gamma$
2: Extended model params $\omega_0 = (\tilde{\theta}^*, W, b_W, A, b_A, 0)$
3: **for** $i = 1$ **to** n **do**
4: $\quad \omega_i \leftarrow \text{Optimizer}_\lambda(\omega_{i-1}, \mathcal{L}_2)$
5: **end for**
6: $t \leftarrow 0$
7: **repeat**
8: $\quad \omega_{n+t+1} \leftarrow \text{Optimizer}_{\lambda'}(\omega_{n+t}, \mathcal{L}_2 + \gamma(t)\|ADW\|)$
9: $\quad t \longleftarrow t + 1$
10: **until** $\|ADW\| < \epsilon$
11: $\theta \longleftarrow proj(\omega_{n+t})$
12: **return** θ (and continue to $train_3$ in bypass pipeline)

In Algorithm 1, we present an implementation of the bypass pipeline described in Fig. 2, whose theoretical properties are analyzed in Section III. In particular, the line 2 of Algorithm 1 corresponds to the *embed* step, lines 3-5 to the *opt1* step, lines 6-10 to the *opt2* step, and line 11 to the *proj* step of the bypass pipeline. The $\text{Optimizer}_\lambda(\omega, \mathcal{L})$ in Algorithm 1 denotes invoking an iteration of a gradient-based optimizer with learning rate λ and loss function $\mathcal{L}$ with respect to model parameters ω. All other symbols correspond to the same symbols found in Section III.

The algorithm starts with the given model φ_1, defined by the input parameter θ^*, which is the result of $train_1$ step of the bypass pipeline. It is reasonable, but not necessary, to consider that θ^* is near a stationary point, witnessed by the slowdown of loss reduction in $train_1$ step.

Then, in line 2, the model is extended into φ_2 by predefined embedding map $embed$ in (10). This extension of parameter space into $\mathcal{D}_2$ will induce immediate growth of gradient norm and loss reduction since the directional derivative along additional parameters would be nonzero, even if the gradient vanishes in original space $\mathcal{D}_1$.

As the parameter space $\mathcal{D}_2$ have been extended by $embed$, even if θ^* were exactly on a stationary point, the starting point $\omega_0 = embed(\theta^*)$ will be no longer a stationary point in the extended space $\mathcal{D}_2$. In lines 3-5, the opt_1 step, the algorithm performs n iterations of $\text{Optimizer}_\lambda(\omega, \mathcal{L}_2)$, allowing the gradient-based optimizer to escape from the vicinity of the stationary point by simply continuing to follow the gradient and further reduce the loss. We verify this expected reduction of the loss during opt_1 step in Section V.

Then the Bypass algorithm proceeds to opt_2 step (lines 6-10), where a modified loss function $\overline{\mathcal{L}} = \mathcal{L}_2 + \gamma(t)\|ADW\|$ is minimized. Since during the opt_2 step gradually increases the relative weight of $\|ADW\|$ with adaptive $\gamma(t)$, repeated optimization iterations in line 8 eventually bring the extended parameter ω back to ϵ-close vicinity of $\varphi_2^{-1}(Im(\varphi_1))$. Hence, the $\|ADW\|$ value and the distance between ω and the starting point θ^* are expected to decrease during this return of ω, as predicted in the theoretical analysis. At the same time, the loss $\mathcal{L}_2$ may increase due to the additional constraint. Therefore, we empirically verify the expected behaviors of opt_2 step as well in Section V.

The $proj$ step of the bypass pipeline follows. Once the extended parameters ω arrive ϵ-close to $\varphi_2^{-1}(Im(\varphi_1))$ in terms of $\|ADW\|$ (line 10), we can invoke the method shown in (15) that projects ω to $\mathcal{D}_1$. The line 11 of Algorithm 1 implements the projection of $\omega \in V(ADW) \subset \mathcal{D}_2$ onto the original model's parameter space $\mathcal{D}_1$. The resulting parameter θ satisfies $\|ADW\| = 0$ but ω only $\|ADW\| < \epsilon$, so there may be a jump in the loss $\mathcal{L}_2$ before and after the $proj$ step. Also, the projection from ω to θ, even if ω is ϵ-close in terms of $\|ADW\|$, can cause a relatively large change of position in the extended parameter space $\mathcal{D}_2$, due to the additive term ADb_W introduced to b_A by $proj$ as shown in (15). The abovementioned expected behaviors are also empirically verified in Section V.

The output parameter θ of Bypass algorithm is not necessarily another stationary point. Therefore, according to the bypass pipeline, we recommend continuing $train_3$ step, in which further optimization or training of the neural network model in original parameter space $\mathcal{D}_1$ takes place. We expect the bypass algorithm to generate θ that does not revert back to the vicinity of θ^*, the starting stationary point, even when the same gradient-based methods used in $train_1$ is used in $train_3$. The expected behavior of θ after $train_3$ is also empirically verified in Section V.

A. relaxed bypass algorithm

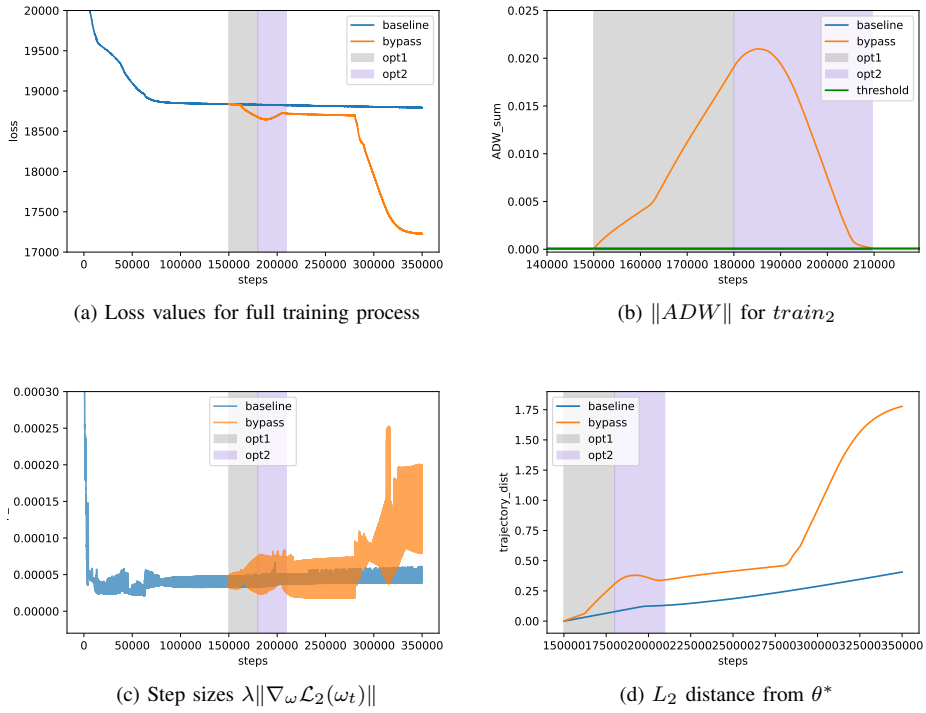
The bypass algorithm was built on assumption of two consecutive dense layers. Some classes of modern deep learning models are not covered, due to model structure containing residual connection or layer of different type. Instead of implementing structure-dependant bypass algorithm, we can relax the requirement of two consecutive dense layer by modifying Algorithm 1.

Without consecutive layer assumption, we can construct bypass algorithm by defining φ_2 and $embed$ equivalently with learnable activation and solving

$$\begin{array}{ll} \underset{\omega}{\text{minimize}} & \mathcal{L}_2(\omega) \\ \text{subject to} & D = 0 \end{array}, \quad (24)$$

instead of (16), since $D = 0$ also implies that $\omega \in \mathcal{D}_2$. With this relaxed constraint, we will adapt following changes for the bypass algorithm:

1) opt_2 will minimize $\mathcal{L}_2 + \gamma(t)\|D\|$ instead of $\overline{\mathcal{L}}_2$
2) $proj$ becomes trivial and linear. Now the $proj$ maps $\omega = (\tilde{\theta}, W, b_W, A, b_A, D)$ to $(\tilde{\theta}, W, b_W, A, b_A)$.



(a) Loss values for full training process

(b) $\|ADW\|$ for $train_2$

(c) Step sizes $\lambda\|\nabla_\omega \mathcal{L}_2(\omega_t)\|$

(d) L_2 distance from θ^*

Fig. 3. Dynamics of Eggholder regression task using gradient descent, with or without bypassing. The grey and purple-colored areas correspond to the opt_1 and the opt_2 steps of bypassing, respectively. Fig. 3a illustrates the change of loss values $\bar{\mathcal{L}}(\omega_t)$ during the training process. Fig. 3b shows the change of $\|ADW\|$ in the $train_2$ step of bypassing. Fig. 3c shows the actual change size of parameter values $\|\omega_{t+1} - \omega_t\|$ during gradient descent iterations. Fig. 3d plots the distance $\|\theta^* - \omega_t\|$ from the staring point $\theta^* = \omega_0$ of the bypassing.

The new objective function $\mathcal{L}_2 + \gamma(t)\|D\|$ still converges to $V(D)$ when $\gamma(t)$ cancels out the effect of $\mathcal{L}_2$ term, since $\|D\|$ is convex. We show the relaxed bypass algorithm also works with Resnets [20], in Section VI-C

V. Experiments I: Bypassing behaviors

We examine the dynamic behaviors of the loss and other parameters from neural network training with bypassing, in order to verify empirically the expected behaviors of the Bypass algorithm discussed in Section IV. When needed, we contrast the results between "Baseline" case (without Bypass) and "Bypass" case (baseline method with Bypass algorithm). To demonstrate empirical usefulness of Bypass algorithm, we present two benchmark neural network model training scenarios: A) a regression task using the eggholder function, and B) a classification task using CIFAR10 dataset.

A. Regression: Eggholder with Gradient Descent

We consider a small neural network with minimal nonlinearity and a regression task to approximate a synthetic dataset. This task is designed to verify the expected behavior and demonstrate the advantage of using the Bypass algorithm with a gradient-based training algorithm. Therefore, to elucidate the theoretical insights from the empirical result as much as possible, this scenario is designed to rule out many stochastic noises such as those in the dataset and the training algorithm.

The regression objective is to approximate the eggholder function, a bivariate real function with an intricate eggholder-like shape, defined as:

$$Eggholder(x_1, x_2) = -x_1 \sin(\sqrt{|x_1 - x_2 + 47|}) + (-x_2 + 47) \sin\left(\sqrt{|x_2 + \frac{x_1}{2} + 47|}\right).$$

We choose the eggholder function to be the approximation target since it contains many local minima by construction.

The neural network model φ_1 in this experiment is comprised of three layers, with ten nodes on each hidden layers, in which RELU activations are used for the second hidden layer. Our model φ_1 is chosen to be small (10-node-wide, 3-layer-deep), such that the underparametrized model may admit many stationary points in the loss function surface of approximating the eggholder function, which contains many local minima.

We construct a synthetic Eggholder dataset D_{egg} by sampling 100 points from a square shaped domain $(-250, 250)^2$ and their eggholder function values, resulting in a collection of $(\vec{x}, y)$, where $\vec{x} = (x_1, x_2)$ and $y = Eggholder(x_1, x_2)$.

1) Experiment Design: The baseline case trains the model φ_1 with a full-batch gradient descent on D_{egg} the eggholder dataset with mean-squared-error (MSE) loss function, which is a reasonable choice of loss function for regression tasks. We set the learning rate for all gradient descent iterations to be fixed at 3×10^{-8}, and continue iterations until the loss reduction slows down to obtain θ^*. Obtaining θ^* marks the end of $train_1$ step, but for the baseline method, the training

simply continues and eventually generates a training trajectory that corresponds to the baseline case, in which bypassing at θ^* is not performed.

The bypass case is initialized exactly the same as the baseline method to facilitate comparison between the two cases. This includes all hyperparameters in the baseline gradient descent method. Once the apparent slowdown of the $train_1$ gradient descent happens and θ^* is obtained, the difference between the baseline case and the bypass case appears. The Bypass algorithm is invoked to attempt escaping from the stationary point in the vicinity of θ^*. Once the Bypass algorithm completes, $train_3$ step restarts, with the exactly same setup as the baseline case but at θ, a different location from θ^* where the bypassing started. Hyperparameters for the Bypass algorithm are set as follow: $\gamma(t) = 0.6t$, and $\epsilon = 10^{-4}$, the learning rate λ' for opt_2 to be same with learning rate of the baseline method.

2) Results and Interpretation: Bypassing reduces the loss significantly, as shown in Fig. 3a where the baseline loss trajectory is slowed down after 100k iterations, but the bypass loss trajectory decreases significantly after bypassing, which begins at 150k iteration point. In particular, the bypass loss trajectory dips down as opt_1 progresses, which is the first expected behavior of bypassing, as the $embed$ step affects to the model weights and gradient to escape from the vicinity of stationary point. Also, during the opt_2 step, the rebound of the bypass loss trajectory is observed, an expected behavior since the opt_2 objective function $\overline{\mathcal{L}}$ defined in (17) introduces an additional $\gamma(t)\|ADW\|$ term. Recall that the hyperparameter $\gamma(t)$ is set to increase with t, which increases opt_2 proceeds, so it is natural for the loss trajectory to increase in opt_2 near its end.

Additionally, the trajectory of $\|ADW\|$ during the bypassing, shown in Fig. 3b, confirms that the Bypass algorithm works according to its theoretical foundations. The trajectory of $\|ADW\|$ confirms the key intended behavior of bypassing: the model parameter ω moves away from the initial model space $\mathcal{D}_1$ during opt_1, and eventually returns to the vicinity of $\mathcal{D}_1$ in terms of $\|ADW\|$ as opt_2 proceeds. Unlike opt_2 in which the optimization has gradually increasing regularization term $\gamma(t)\|ADW\|$, opt_1 is not affected by $\|ADW\|$ value and allows ω wanders away from the stationary point (decreasing loss) into the newly extended model space (increasing $\|ADW\|$).

From the perspective of each iteration in gradient-based optimizer, as shown in Fig. 3c, the baseline and the bypass trajectories are significantly different. In Fig. 3c, we plot the trajectory of the norm of the full gradient $\|\nabla_\omega \mathcal{L}_2(\omega_t)\|$, which shows how "wide" each steps taken in each iteration. Since $\omega_{t+1} = \omega_t - \lambda \mathcal{L}_2(\omega_t)$, the plotted values are interpreted as $\|\omega_{t+1} - \omega_t\|$. Once bypassing begins with opt_1, the bypass trajectory shows wider steps than the baseline counterpart, and this is an evidence on empirical effectiveness of Bypass algorithm in escaping from the stationary point θ^* where the model was at the 150k-th step. Another interesting observation is that the bypass trajectory shows much larger stepsize even after opt_2 step when the bypassing is completed and the parameter is back to the original model space $\mathcal{D}_1$. Wide steps even after returning to the original model space is not expected from theoretical analysis but empirically evident in Fig. 3c, and its effect is clearly reflected in the bypass loss trajectory in Fig. 3a.

Fig. 3d shows how the distance of the model parameters changes from the starting parameter θ^* of bypassing. First, we directly verify that bypassing leads to ω_t that is different from the start point θ^*. Also, the bypass method leads to a different parameter point from the baseline method after the same number of iterations as shown in the difference at the end of opt_2 step. Moreover, achieving the design purpose, Bypass algorithm actually led to a significantly different attraction point, as the distance difference between the bypass trajectory and the baseline trajectory widens as $train_3$ proceeds in the original space D_1 with the same gradient descent algorithm.

B. Classification: CIFAR10 with Minibatch SGD

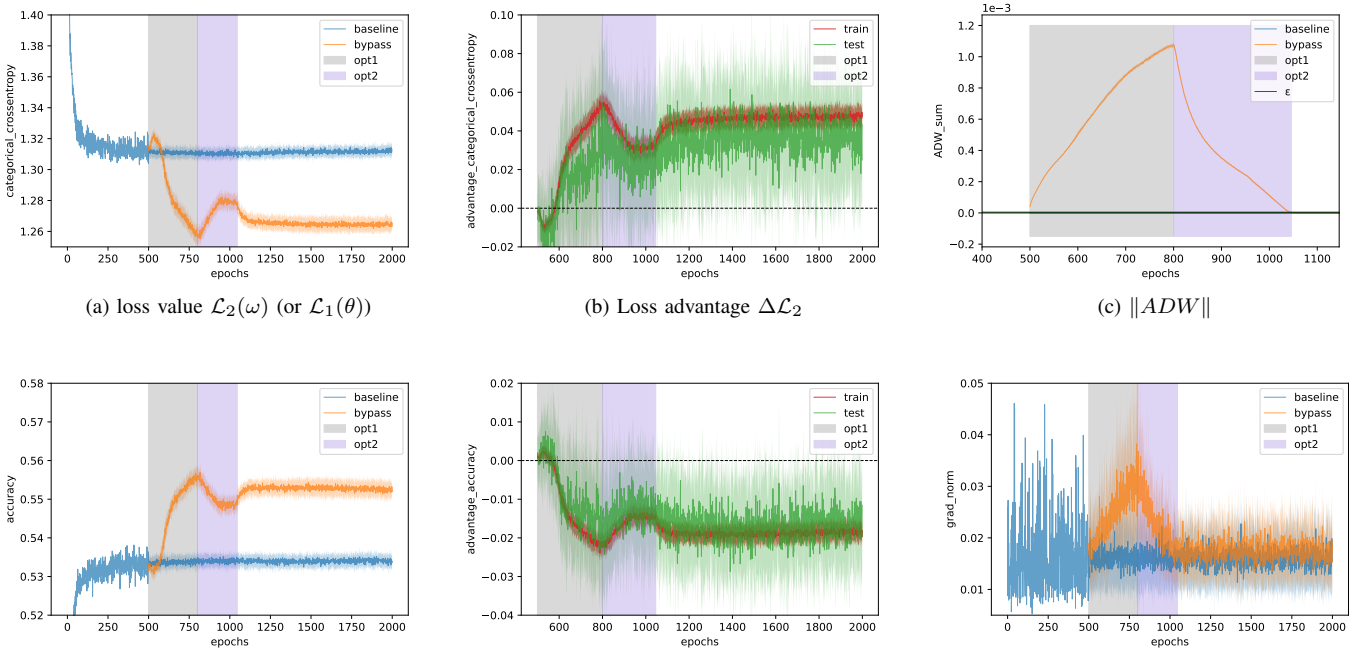
In this classification task, we demonstrate the effect of Bypass algorithm to stochastic gradient descent (SGD) with minibatches. CIFAR10 dataset [21] contains 60k images across 10 classes with a predefined train-test split which has been widely used as a benchmark dataset for multi-class image classification task. We use the provided dataset split and 10-class-classification task as the goal of this task.

We modify 2c2d model from DeepOBS [22] to use it as the classifier in this task. The 2c2d model, composed of two-convolution layers and two-dense layers with ReLU activations, contains total of 1,703,370 learnable parameters. We introduce 1,024 learnable parameters to 2c2d model to allow bypassing. Also, we modify the input channels of 2c2d model to accept the images from CIFAR10.

1) Experiment Design: As the baseline case, we train vanilla 2c2d model on CIFAR10 classification task using Adam optimizer as a minibatch SGD optimizer with batch size 128 and constant learning rate 0.004. Cross-entropy loss with weight decay of 0.005 (equivalently, with regularization loss term $0.005\|\theta\|_2^2$) is used. We train the model up to epoch 500 and use the parameter as the starting point θ^* for Bypass algorithm. The loss function and the training accuracy, as shown in the blue line plots of Fig. 4a and Fig. 4d with noise due to minibatching, suggest that the training process may be near a suboptimal stationary point. Using θ^* from epoch 500, we perform 20 independent trials of baseline training to account for the innate randomness in minibatch SGD.

The bypass case invokes Bypass algorithm at θ^* with $\epsilon = 5 \times 10^{-6}$ and $\gamma(t) = 0.005t$. All the other settings were set to be the same as the baseline case. To ensure the 20 trials of bypassing are independent, both SGD and Bypass algorithm are initialized with different random seeds at the 500th epoch. Note that the scale of noise in loss and accuracy looks to be decreased after epoch 500 because they're averaged over 20 repetitions.

2) Results in Model Performance: Similar to the eggholder regression task, bypassing improves the empirical performance of the 2c2d model by rescuing it from the local suboptimality. The Adam optimizer with minibatch does not rescue the model from the local suboptimality, which is clearly shown in the



(a) loss value $\mathcal{L}_2(\omega)$ (or $\mathcal{L}_1(\theta)$) (b) Loss advantage $\Delta\mathcal{L}_2$ (c) $\|ADW\|$

(d) Accuracy of ω_t (e) Accuracy difference ΔAcc (f) Step sizes $\lambda\|\nabla_\omega \mathcal{L}_2(\omega_t)\|$

Fig. 4. Training results of baseline and bypassing in CIFAR10 classification task. Gray shaded epochs correspond to the opt_1 step, and purple to the opt_2 step of bypassing. All plots from 500th epoch are averages over 20 repetitions, with their sample standard deviation as the error bar shades. Fig. 4a compares the training set loss trajectories. Fig. 4b plots both the training loss and the test loss advantages per epoch, which is the difference of each loss by the model from the bypass case and the model from the baseline case. Fig. 4c shows the trajectory of $\|ADW\|$ during bypassing. Fig. 4d compares the training set accuracy trajectories. Fig. 4e plots both the training accuracy and the test accuracy advantages per epoch, in which advantages are computed in the same manner as Fig. 4b. Fig. 4f compares the trajectories of $\lambda\|\nabla_\omega \mathcal{L}_2(\omega_t)\|$, in which ω_t is the parameter at the end of epoch t, obtained from learning the training set. This value can be seen as the cumulative step size of epoch t.

baseline loss trajectory in Fig. 4a. Applying Bypass algorithm allows the same optimizer to escape the suboptimal region and achieve much lower loss value, as shown in the bypass loss trajectory in Fig. 4a. This reduction in loss translates to the improved classification accuracy in the bypass case as shown in Fig. 4d.

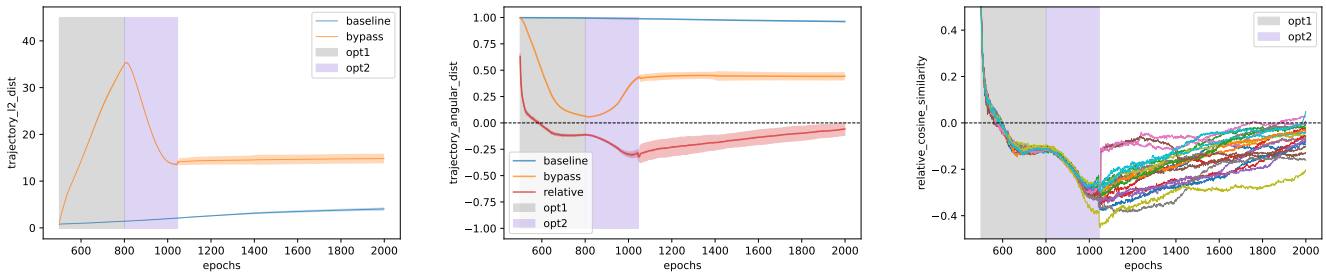
3) Results in Bypassing Behavior: Specific behaviors of the bypass trajectory during opt_1 step are in accordance to the theoretical properties of bypassing and the empirically observed results of Bypass algorithm in the eggholder task. Reduced loss (Fig. 4a) and increased model performance (Fig. 4d) are achieved in opt_1 as the model parameters move away from the original model space, which is clearly shown by the increase in $\|ADW\|$ (Fig. 4c). In opt_2, the model returns to the original model space as its $\|ADW\|$ decreases (Fig. 4c), during which the loss and the performance improvements are affected negatively.

The overall empirical improvements of using Bypass algorithm is significant, both immediately after bypassing and through the subsequent $train_3$ step. When the model completes bypassing near the end of opt_2 with $\|ADW\| = 0$ around 1045th epoch (Fig. 4c), the model consistently returns to the original model space with significantly lower loss (Fig. 4a) and higher accuracy (Fig. 4d) than the baseline trajectory averages. Note that both the loss and the accuracy plots validate the expected behavior that the model with Bypass algorithm escapes from the stationary vicinity from which the model could not escape in the baseline trajectory.

Since CIFAR10 dataset offers predefined training/test set split, we also verify the generalizability of empirical improvements of Bypass algorithm. We plot the difference between the performance of the models from the bypass case and the baseline case, from the training set and the test set of CIFAR10, in terms of loss (Fig. 4b) and accuracy (Fig. 4e). As shown in the two figures, the loss reduction and the performance improvements from the training set is reflected to the test set trajectory, which strongly suggests that the empirical benefit of using Bypass algorithm on training set generalizes to the test set performance as well.

To provide additional insights on how the expected behavior of bypassing materializes under the innate randomness of minibatch SGD, we plot the trajectories of the learned model parameters from both the baseline case (θ_t) and the bypass case (ω_t) in Fig. 5. In particular, to consistently plot both parameter trajectories across the bypassing and the baseline epochs, we compute all parameters' norms in the extended parameter space $\mathcal{D}_2$.

First of all, applying minibatch SGD with bypassing consistently leads to significantly different final model parameter from applying only minibatch SGD. As shown in Fig. 5a, there is a significant difference in the average of L_2 distance trajectories of model parameters from the bypass and the baseline cases that started at the same location θ^* on epoch 500. The increase in L_2 distance during opt_1 and the decrease



(a) L_2 distance from θ^* (b) cosine of angular distances (c) diverse trajectories

Fig. 5. Measurements of the training trajectory θ_t and ω_t, the model parameters of baseline and bypass case at epoch t, respectively. Gray shaded area and purple colored area represents the opt_1 and opt_2 respectively. Gray shaded epochs correspond to the opt_1 step, and purple to the opt_2 step of bypassing. The plot value and its error bar shades in Fig. 5a and Fig. 5b are the average and the sample standard deviation from 20 repeats. Fig. 5a plots the L_2 distance $\|\theta^* - \theta\|$, between the model parameter immediately before bypassing starts θ^* and the current model θ, which may be θ_t or ω_t depending on the epoch and the case. Fig. 5b plots the cosine values of three angles over epochs – "baseline" showing $\angle\theta^* O\theta_t$, "bypass" showing $\angle\theta^* O\omega_t$, and "relative" showing $\angle\theta_t\theta^*\omega_t$. Fig. 5c plots the individual trajectory of "relative" $\angle\theta_t\theta^*\omega_t$ from different trials of bypassing,

during opt_2 are also expected, similar to the behavior captured by the ADW norm trajectory shown in Fig. 4c.

We observe significant difference not only in the L_2 distance but also in terms of angular distance between the parameters from the baseline and the bypass direction, as shown in Fig. 5b. The "baseline" trajectory of $\cos(\angle\theta^* O\theta_t)$ staying near 1 means θ^* and θ_t stay in the same direction from the origin O, which implies that the direction of minibatch SGD in the baseline case does not change much over 1500 epochs. On the other hand, the "bypass" trajectory of $\cos(\angle\theta^* O\omega_t)$ in Fig. 5b directly confirms how the behavior expected according to the theoretical analysis of bypassing interacts with minibatch SGD:

- In opt_1 step of bypassing, the model parameter ω_t moves away from θ^* in the direction *out of* the original model space D_1, in which θ^* and θ_t are confined. The deviation of the "bypass" trajectory from the "baseline" trajectory is partially attributed to the fact that the angles behind each trajectory involve the origin O that may be relatively far away from θ^*.
- In opt_2 step of bypassing, the angular distance between ω_t and θ^* decreases, which may be partially due to the minimization objective function in opt_2 encourages $\|ADW\|$.However, even at the end of opt_2, when $\|ADW\|$ is reduced to ϵ as shown in Fig. 4c, it is notable that the cosine value is significantly different from 1. Hence, Bypass algorithm generates the return points that are consistently different not only in terms of distance (Fig. 5a) but also in terms of the angular direction from the origin (Fig. 5b).

Direct angle between θ_t and ω_t are captured by "relative" trajectory, which is the average of $\cos(\angle\theta_t\theta^*\omega_t)$ over 20 trials. Note that the cosine value average drops down much faster than the "bypass" trajectory, because "relative" trajectory corresponds to the angle $\angle\theta_t\theta^*\omega_t$ measured not from the origin O but from the bifurcation point θ^* between the bypass case and the baseline case. Also, the average of the cosine value tending to 0 throughout subsequent $train_3$ suggests that the first-order optimization parameter trajectory tends to directions vastly different from the baseline trajectory, which is empirically verified with the 20 different trajectories shown in Fig. 5c. Additionally, the diversity of trajectories in Fig. 5c suggests that the empirical effect of Bypass algorithm is much more complex and far-reaching than applying random perturbations towards different directions.

VI. Experiments II: Bypass algorithm in practice

The bypass pipeline and algorithm have multiple components to be customized. For example,

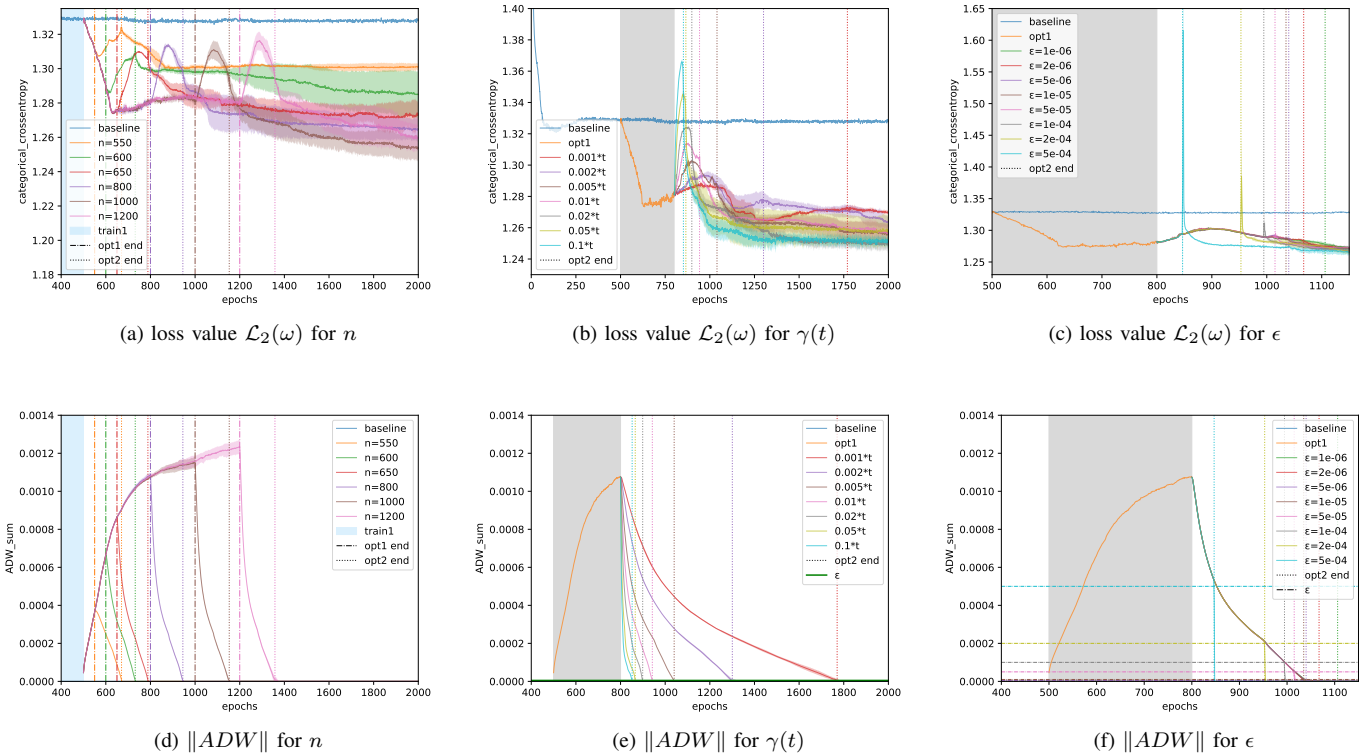
1) The hyperparameters $n, \gamma(t)$ and ϵ for opt_1 and opt_2.
2) The starting point θ^* of the bypass algorithm.
3) The bypass pipeline, which consists of three components explained in Section II-B

In this section, we show alternative experimental results when various modifications are applied. In Section VI-A, we show an sensitivity analysis for hyperparameters proposed in Algorithm 1, compared to the results in Section V-B. We then show what happens if the starting point θ^* is changed to the early stage of the training, which is not empirically converged, by showing a result of extreme case $\theta^* = \theta^0$ in Section VI-B with same problem of Section V-B. Lastly, we consider Resnet model with different bypass pipeline, which employs relaxed bypass algorithm explained in Section IV-A on multiple number of activations, to show the practical availability of variant of bypass algorithm also works.

A. sensitivity analysis on hyperparameters

The proposed bypass Algorithm 1 is controlled by several hyperparameters: n, $\gamma(t)$ and ϵ, which are needed to be tuned for practical use. In this section, we explain the role of each hyperparameters with corresponding experimental results with different choices of them.

1) n : number of opt_1 iteration: During opt_1 phase, we desire an optimizer to explore and gain amount of advantages in extended space $\mathcal{D}_2$. Therefore it is natural to expect smaller iterations for opt_1 training will induce less exploration and fewer advantage gain.



(a) loss value $\mathcal{L}_2(\omega)$ for n (b) loss value $\mathcal{L}_2(\omega)$ for $\gamma(t)$ (c) loss value $\mathcal{L}_2(\omega)$ for ϵ

(d) $\|ADW\|$ for n (e) $\|ADW\|$ for $\gamma(t)$ (f) $\|ADW\|$ for ϵ

Fig. 6. Experimental results of bypass algorithm in CIFAR10 classification task with different hyperparameters.Fig. 6a,Fig. 6b and Fig. 6c show loss reduction and Fig. 6d,Fig. 6e,Fig. 6f show the change of $\|ADW\|$ during the bypass training. Fig. 6a and Fig. 6d show the result of bypass training when the opt_1 iteration changes in epochs. The light-blue shaded area represents $train_1$,dot-dashed line shows when opt_1 stopped, and dotted line shows when the opt_2 ended. Fig. 6b and Fig. 6e show the results when the $gamma(t)$ differs, and Fig. 6c and Fig. 6f show the results when the ϵ changes. All plot values and its error bars after θ^* are the average and the sample standard deviation from 5 repeats.

Compared to results in Section V-B with 300 epochs for opt_1, we iterated same experiments with fixed θ^* and same settings but number of iterations. As illustrated in Fig. 6a and Fig. 6d, the fewer n induces less advantages in loss and smaller value in $\|ADW\|$ growth, as expected. The larger number of iterations did not give better results since the additional iterations did not give further loss reduction in opt_1 phase.

2) $\gamma(t)$: The $\gamma(t)$, weight of $\|ADW\|$ in $\overline{\mathcal{L}}$ is designed to have two roles:

1) Cancel out $\mathcal{L}_2$ to arrive at $\{\omega : \omega \in \|ADW\| < \epsilon\}$
2) Control the balance between speed of loss reduction and coming back to $V(ADW)$.

The first role of $\gamma(t)$ with $lim_{t\to\infty}\gamma(t) = \infty$ is shown in previous sections that $\|ADW\|$ did ϵ-closely converged to 0. Here, we show the second role by showing results with different choice of $\gamma(t)$.

The second role gives the trade-off relation between the stability in loss $\mathcal{L}_2(\omega_0)$ and the $\|ADW\|$-reducing speed. More precisely, if $\gamma(t)$ is chosen to be larger, the opt_2 phase training will reduce $\|ADW\|$ faster but sacrifice the advantages gained during opt_1 more.

We show this trade-off relation by corresponding experiments. We fix the opt_1 trajectory and iterated experiments with same hyperparameters in Section V-B but $\gamma(t)$. Fig. 6b and Fig. 6e show this trade-off relation clearly. If the $\gamma(t)$ grows fast, the opt_2 reduces $\|ADW\|$ with smaller iterations, but the loss peak of during opt_2 appears to be significantly large, which is even larger than the baseline. This explosion of loss above the baseline may lead to worse destination compared to baseline since we cannot guarantee that the $train_3$ would compensate this consumption of loss advantage.

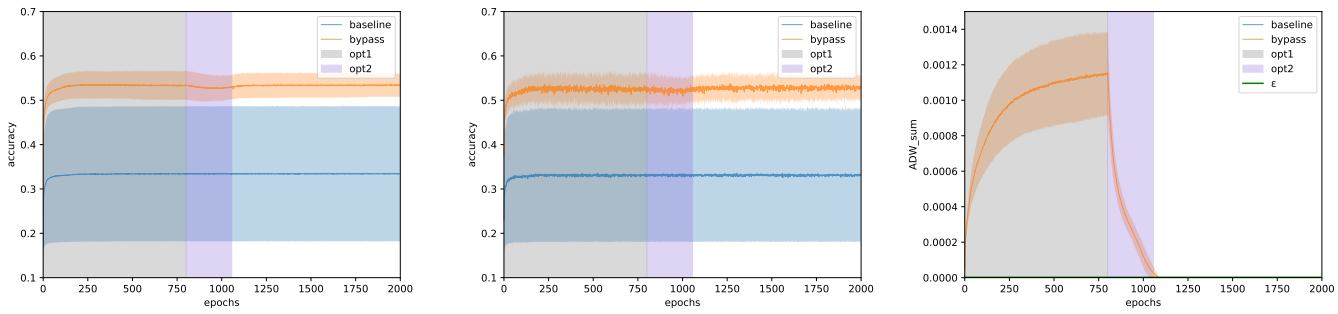
For the smaller $\gamma(t)$, the opt_2 returns stable loss trajectory below the baseline curve, but consumes more iterations for opt_2 due to slower reduction of $\|ADW\|$.

3) ϵ: At the final stage of the bypass algorithm, the $proj$ will contract the model into original parameter space $\mathcal{D}_1$. This projection would be function-preserving when $\|ADW\| = 0$ but this condition was relaxed to $\|ADW\| < \epsilon$ for empirical setting. Therefore the choice of large ϵ would be harmful for the function-preserving property and hence related to the stability of the bypass algorithm, because if the loss explode in any phase of the bypass algorithm, we cannot expect that the destination of $train_3$ would be better than the baseline.

The results of corresponding experiments are shown in Fig. 6c and Fig. 6f. Increase of ϵ shows the reduction of opt_2 steps, but significantly violates the $proj$ stability. Later the $train_3$ resolved this explosion, but this behavior might not always appear.

B. starting at initialization

The bypass algorithm is designed to escape from vicinity of θ^* without restarting, under assumption of vanishing gradient.



(a) train accuracy (b) test accuracy (c) $\|ADW\|$

Fig. 7. Result of starting at initialization θ^0 on CIFAR10 classification task. The gray and purple colored areas represent opt_1 and opt_2 respectively. The plot values and error bars are the mean and sample standard deviation from 20 repetitions. All hyperparameters except the epochs of $train_1$ are chosen equal to the experiments in Section V-B.

But the vanishing gradient assumption is usually not satisfied especially with SGD, and thus $embed$ will be applied when the loss is still reducing. However, the bypass algorithm would work at any starting point of neural network training since the extension & contraction does not consider the gradient of the loss function. As an extreme case, we can skip $train_1$ and consider the case of $\theta^0 = \theta^*$. In this case, the result of opt_1 is expected to be better than the baseline, at least not worse, since it minimizes same loss function with larger model which is exactly an extension of baseline model. The expected behavior of remaining parts of bypass are expected to be similar to Section V-B.

To show that the bypass algorithm works when $\theta^* = \theta^0$, we fixed θ^0 and iterated bypassing 20 times with same hyperparameters utilized in Section V-B and different random seeds. Due to randomness on SGD, the destinations of baseline training were not fixed, but the result of bypass trajectories was always advantageous compared to the baseline. Fig. 7 shows the advantages gained by bypass algorithm compared to baseline trajectory with same random seed in accuracy plots, and change of $\|ADW\|$ during the training phase.

Typical amount of advantage is observed during opt_1 in Fig. 7a and Fig. 7b, with growth of $\|ADW\|$, as opt_1 solves different minimization problem with expanded neural network. Continue with opt_2, the $\|ADW\|$ shrinks to 0 with sacrifice of advantage with small amount shown in Fig. 7c, followed by $proj$ which completes graceful reduction of model size back to original model size prior to $embed$. We emphasize that the amount of performance drop in opt_2 and $proj$ are observed to be small and still advantageous compared to the model found by vanilla training. This small gap induces an special benefit of bypassing that the resulting model have similar performance with extended model, which is smaller in model size.

The results with $\theta^* = \theta^0$ also show in Fig. 7a and Fig. 7b that the bypass algorithm is advantageous when we restart the training, when the θ stuck in the stationary point empirically. If we just apply restarting, the training trajectory would lie in the blue colored area, which is suboptimal to the destination of bypassing results.

C. relaxed bypass algorithm with Resnet

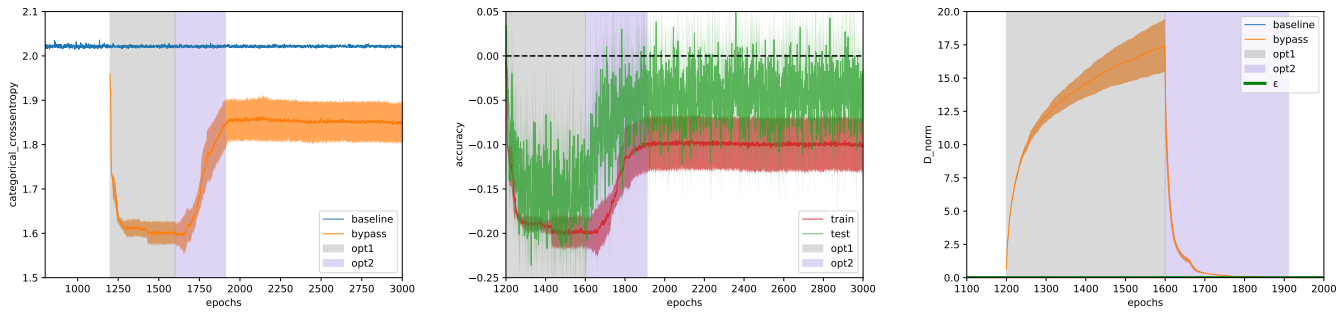
We analyze the effect of relaxed bypass algorithm on Resnet, which is a popular neural network architecture with residual connections. Since there is no consecutive fully connected layers in its architecture, the relaxed version of bypass algorithm is required. In this section, we apply the Resnet18 model [20] with relaxed bypass algorithm to the CIFAR10 classification problem and show that the relaxed bypass algorithm also works for the model with residual connection, using multiple number of learnable activations. The Torchvision-implemented version [23] of Resnet18 model is employed here for the experiment.

1) Experiment Design: We employ Resnet-18 structure and apply $embed$ on every RELU activations in residual blocks. The opt_2 will minimize sum of loss function $\mathcal{L}_2$ and all $\|D\|$ of each extended learnable activations. During opt_2, each learnable activations will be projected immediately when $\|D\|$ reaches $< \epsilon$.

For the baseline, we train Resnet-18 model up to 1200 epochs by Adam optimizer with cross-entropy loss, batch size 128, constant learning rate 0.0005 and weight decay of 0.1 to place θ^* in vicinity of suboptimal stationary point.

After setting θ^* as a starting point, we apply $embed$ to the model and start to follow the trajectory generated by the Bypass algorithm. The opt_1 explores $\mathcal{D}_2$ for 800 epochs and the opt_2 with $\epsilon = 0.002$ and $\gamma(t) = 3 \times 10^{-6}t$ makes the model come back to $\mathcal{D}_1$. After 60 epochs in opt_2, we raise the growth rate $\gamma(t)$ to $3 \times 10^{-5}t$.

The results are shown in Fig. 8. Fig. 8a shows the result similar to Section V. The opt_1 gains meaningful advantage in extended space $\mathcal{D}_2$ and opt_2 consumes the advantage but with smaller amount while the baseline trajectory stuck in suboptimal region of stationary. In advantage plot Fig. 8b shows that the bypass algorithm gives advantage also in accuracy, with significant amount. Fig. 8c show that the opt_2 can solve the constrained optimization problem even with multiple extensions.



(a) train loss (b) Accuracy difference Δ Acc (c) Sum of $\|D\|$

Fig. 8. Results on CIFAR10 experiment with Resnet18 model. The gray and purple colored area represents opt_1 and opt_2 respectively. All plots after epoch 1200 are the results of 5 iterations with different random seed. Fig. 8a shows the categorical cross entropy loss of training process. Fig. 8b plots both the training and test accuracy advantages per epoch, in which the negative value represents the growth of accuracy. Fig. 8c show the change of sum of $\|D\|$ value during the bypassing.

VII. Discussion

Bypass algorithm, which is our implementation of bypass pipeline with learnable activation and corresponding comeback constraint $ADW = 0$, exhibits many behaviors that are theoretically expected. According to the theoretical expectation and observed in Fig. 5c that the translation of parameter vector

$$\|ADb_W\| = \|proj(\omega^*) - \omega^*\|. \tag{25}$$

is not necessarily vanishing, the $proj$ of bypass algorithm may allow a second escape. Even if $\nabla_\omega \mathcal{L}_2(\omega)$ is empirically converged, the nonlinear projection $proj$ and non-vanishing translation may provide the loss-reducing gradient on $\mathcal{D}_1$ in $train_3$.

Practical superiority of bypassing arises from the computational cost that the additional cost compared to vanilla training is low. Modifications required for bypassing layer introduces additional $O(s_W)$ computation cost in forward and backward propagation, which is linear to the width s_W of the bypassing layer. Additional cost arises from the computation of $\|ADW\|$ and backpropagation during opt_1 and opt_2, which require constant number of additional matrix multiplications per iteration. Further reduction of additional computation cost is possible with relaxed bypass algorithm described in Section IV-A which minimizes $\|D\|$ instead of minimizing $\|ADW\|$, but at the same time forcing the $proj$ to be trivial, which may affect performance of subsequent optimization after $proj$.

Yet, our Bypass algorithm is not a cure-all, because bypassing via learnable activation may not be always successful due to diverse structural properties of models φ_1 and φ_2 and local loss landscape. To get the most advantage out of bypassing, the optimization iterations of opt_1 must be effective in reducing the loss, which may not happen depending on the given model and parameter θ^*. A practical safeguard to contain the risk from such unwanted scenarios is to test out a few iterations of opt_1 from θ^* to determine the prospective effect of using the Bypass algorithm before fully committing.

Both the $proj$ step under comeback constraint and the $embed$ step in the bypass pipeline can be seen as function-preserving operators. The function-preserving properties of operators working on neural network models has been extensively researched in various methods of neural architecture search (NAS), for instance in growing networks methods [24]–[27] or network morphism methods [28]–[33]. From the perspective of NAS, the bypass pipeline as a whole can be considered as a network morphism explicitly designed to explore the network architecture space based on the functional properties of the given model and its parameters.

The main difference between NAS and bypassing comes from the contraction method, since function preserving operators suggested in NAS only increases the model size (in number of parameter) but our contraction method suggest to decrease the model size if the parameter satisfies corresponding algebraic constraints in proposal.

As Bypass algorithm exploits algebraic constraints of the given parameters to explore the model space, the approach is distinct from other NAS methods exploiting properties of the building blocks of the model, such as decomposing a single convolution into two consecutive convolutions [29] or inserting additional block of layers which is initialized as identity map [33]. Hence, the bypass pipeline can be seen as a just-in-time network architecture search method whose search direction depends on both the model and its current parameter, with the explicit goal of escaping local hostilities that can trap most network training methods used in practice.

The bypass algorithm by learnable activation is applicable to convolution layer and attention mechanism with minimal modification, due to linearity of convolution and Q,K,V computation of attention. For those complicated structure of modern deep neural networks, handling multiple layers is practically required, and expected to be complicated to analyze the behavior in first paper proposing bypass pipeline. The near-future research would be applied on those in-practice modern neural networks.

Bypass pipeline is very practical, as it is compatible with not only different optimizers of neural networks but also a wide range of helping methods, such as warm-up, saddle point escaping algorithms, or even another bypass pipeline. While our carefully chosen set of experiments focus on demonstrating the empirical validity of bypassing, this practical

compatibility is another theoretically expected properties of bypassing. Moreover, we expect even greater empirical success of bypassing with different combination of techniques and methods of neural network training.

VIII. Conclusion

We present bypassing, a novel extension-contraction-based concept to mitigate the inevitable slowdown of first-order optimization methods widely used in neural network training when they encounter stationary points or their vicinity. We propose bypass pipeline, a particular formulation of bypassing based on introducing a specific form of learnable activation for extension and imposing algebraic constraints required for contraction. We analyze the theoretical characteristics of the pipeline, and verify the expected behavior with a Bypass algorithm, which is our implementation of the bypass pipeline. In addition to behavior verification, we also demonstrate empirical effectiveness of the Bypass algorithm with benchmark regression task and classification task. Moreover, we present various customization on the bypass algorithm with alternative training scenarios, for practical use. Bypass algorithm shows promising empirical results and behaviors expected from theoretical analysis, which suggests the conceptual validity of using a bypass pipeline to accelerate gradient-based optimizers used in neural network training even further.

References

[1] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[2] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM Journal on optimization*, vol. 19, no. 4, pp. 1574–1609, 2009.

[3] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.

[4] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1225–1234. [Online]. Available: https://proceedings.mlr.press/v48/hardt16.html

[5] T. M. Breuel, "The effects of hyperparameters on sgd training of neural networks," *arXiv preprint arXiv:1508.02788*, 2015.

[6] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade*, 2012.

[7] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," *Advances in Neural Information Processing Systems*, vol. 27, pp. 2933–2941, 2014.

[8] R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping from saddle points — online stochastic gradient for tensor decomposition," in *Proceedings of The 28th Conference on Learning Theory*, ser. Proceedings of Machine Learning Research, P. Grünwald, E. Hazan, and S. Kale, Eds., vol. 40. Paris, France: PMLR, 03–06 Jul 2015, pp. 797–842. [Online]. Available: https://proceedings.mlr.press/v40/Ge15.html

[9] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan, "How to escape saddle points efficiently," in *International conference on machine learning*. PMLR, 2017, pp. 1724–1732.

[10] K. Y. Levy, "The power of normalization: Faster evasion of saddle points," *arXiv preprint arXiv:1611.04831*, 2016.

[11] C. Jin, P. Netrapalli, R. Ge, S. M. Kakade, and M. I. Jordan, "On nonconvex optimization for machine learning: Gradients, stochasticity, and saddle points," *Journal of the ACM (JACM)*, vol. 68, no. 2, pp. 1–29, 2021.

[12] C. Yun, S. Sra, and A. Jadbabaie, "Small nonlinearities in activation functions create bad local minima in neural networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rke_YiRct7

[13] T. Ding, D. Li, and R. Sun, "Sub-optimal local minima exist for almost all over-parameterized neural networks," *ArXiv*, vol. abs/1911.01413, 2019.

[14] S. Arora, N. Cohen, N. Golowich, and W. Hu, "A convergence analysis of gradient descent for deep linear neural networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=SkMQg3C5K7

[15] Y. Zhou and Y. Liang, "Critical points of linear neural networks: Analytical forms and landscape properties," in *Proc. Sixth International Conference on Learning Representations (ICLR)*, 2018.

[16] C. Yun, S. Sra, and A. Jadbabaie, "Global optimality conditions for deep neural networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=BJk7Gf-CZ

[17] K. Kawaguchi, "Deep learning without poor local minima," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/f2fc990265c712c49d51a18a32b39f0c-Paper.pdf

[18] B. D. Haeffele and R. Vidal, "Global optimality in neural network training," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7331–7339.

[19] P. Cheridito, A. Jentzen, and F. Rossmannek, "Landscape analysis for shallow neural networks: complete classification of critical points for affine target functions," *Journal of Nonlinear Science*, vol. 32, no. 5, p. 64, 2022.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[21] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[22] F. Schneider, L. Balles, and P. Hennig, "Deepobs: A deep learning optimizer benchmark suite," in *7th International Conference on Learning Representations (ICLR)*, May 2019. [Online]. Available: https://openreview.net/pdf?id=rJg6ssC5Y7

[23] T. maintainers and contributors, "Torchvision: Pytorch's computer vision library," https://github.com/pytorch/vision, 2016.

[24] X. Yuan, P. H. P. Savarese, and M. Maire, "Growing efficient deep networks by structured continuous sparsification," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=wb3wxCObbRT

[25] Q. Chen and W. Hao, "A homotopy training algorithm for fully connected neural networks," *Proceedings of the Royal Society A*, vol. 475, no. 2231, p. 20190662, 2019.

[26] U. Evci, B. van Merrienboer, T. Unterthiner, F. Pedregosa, and M. Vladymyrov, "Gradmax: Growing neural networks using gradient information," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=qjN4h_wwUO

[27] G. Wang, X. Xie, J. Lai, and J. Zhuo, "Deep growing learning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2812–2820.

[28] T. Chen, I. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," *arXiv preprint arXiv:1511.05641*, 2015.

[29] T. Wei, C. Wang, Y. Rui, and C. W. Chen, "Network morphism," in *International conference on machine learning*. PMLR, 2016, pp. 564–572.

[30] T. Wei, C. Wang, and C. W. Chen, "Stable network morphism," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[31] ——, "Modularized morphing of deep convolutional neural networks: A graph approach," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 305–315, 2021.

[32] J. Lu, W. Ma, and B. Faltings, "Compnet: Neural networks growing via the compact network morphism," *arXiv preprint arXiv:1804.10316*, 2018.

[33] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ByME42AqK7