Can Theoretical Algorithms Efficiently Escape Saddle Points in Deep Learning?

Sean Farrell [*1] **Carlos Quintero Peña** [*2]

Abstract

This document provides a literature review for the most important recent works related to optimizing high-dimensional non-convex functions in the presence of saddle points mostly for machine learning applications. The inspiration came from reviewing the paper "How to Escape Saddle Points Efficiently?" (Jin et al., 2017a). A large research effort has been devoted to proposed methods that can converge to second order stationary points efficiently. Of special interest is the set of methods that do not rely on Hessian computation, mainly driven by applications in machine learning where this may not be feasible. Although, many important theoretical results have been proposed, many of them have not be tested in real experiments, especially in the context of training a deep neural network. We have designed experiments with different network architectures and state-of-the-art datasets to observe the behavior of perturbed versions of gradient descent. Initial results show that an improvement in experimental convergence rate can be seen only for small and shallow networks. These results, although still encouraging, does not allows us to conclude on the practicality of the analyzed algorithms.

1. Literature Review

1.1. Context

The pioneering work of (Dauphin et al., 2014) is one of the first works that bring attention to the analysis of convergence of gradient-based optimization methods in the presence of high-dimensional non-convex functions in a variety of applications, including statistical physics and neural networks, among others. In general, the proliferation of saddle points as the dimension of the problem increases is exponential, suggesting that saddle points and not local minima are responsible for slowing down the convergence of both first order and second order methods. Their algorithm called saddle-free Newton method (SFN) uses curvature information to define a trust region allowing it to escape saddle points. Their observations are experimentally justified in the context of neural networks, however, no theoretical characterization is provided.

The intuition gained from experimentation in low-dimensional problems can lead us to misleading conclusions when extending the concepts to higher-dimensional problems. In an extreme case, consider the random matrix theory. Choosing an eigenvalue with exact eigenvalue of 0 has probability 0, while for larger dimensions it is exponentially unlikely to get all eigenvalues either positive or negative, which means that most critical points will be saddle points.

(Choromanska et al., 2014) shows one of the first theoretical attempts to explain the optimization of highly non-convex multi-layer neural network functions using results from the prism of spin-glass theory.

These results have inspired a plethora of works that aimed at: i) understanding the behavior of gradient descent (GD) under the presence of saddle points, ii) the proposal of GD modifications that aim at improving its convergence properties to converge to second-order stationary points and iii) the proposal of new algorithms that are not GD modifications that achieve improved performance over traditional optimization algorithms in the presence of saddle points. Most of these efforts have been heavily driven by applications in machine learning and signal processing, where non-convex and high-dimensional problems easily arise.

A highly influential line of work is the one that strives to understand and improve the behavior of GD-based algorithms in the presence of saddle points. Originally, (Ge et al., 2015b) showed how a simple variation of GD named Noisy Stochastic Gradient can converge to a local minimum in $d^4\text{poly}(\epsilon^{-1})$, where d is the dimension and ϵ is the size of the gradient at the critical point (see Definition 1). In their approach the idea is to add noise to the gradient sampled uniformly from the unit sphere. Their choice of adding noise

[*]Equal contribution [1]Department of Electrical and Computer Engineering, Rice university, Houston, Texas, USA [2]Department of Computer Science, Rice university, Houston, Texas, USA. Correspondence to: Sean Farrell <smf5@rice.edu>, Carlos Quintero Peña <carlosq@rice.edu>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

to the gradient is based on observations that even though gradient-based methods will not move when converged to a stationary point (one with $\nabla f(x) = 0$), the randomness in stochastic gradient updates helps the algorithm to escape unstable stationary points, such as saddle points. In their proof, they use the fact that when the algorithm is near a saddle point, there is a finite number of steps in which the expected value of the function will slightly decrease with respect to the one in the stationary point. In other words, the update of the algorithm guarantees that the point will move in a direction of negative curvature and will remain close to it in directions of positive curvature. They prove this using martingale theory. Later, (Lee et al., 2016) showed that gradient descent converges asymptotically to minima with either random initialization or noise. For this, the authors use the Stable Manifold Theorem. Intuitively, if the initial point of the algorithm has a component outside the subspace spanned by the standard basis vectors corresponding to the positive eigenvalues of the Hessian, then GD will converge to the corresponding saddle point. The probability of the initial point landing in this subspace is zero. In their proof, they use the fact that there exists a diffeomorphism between the neighborhood of a critical point and a stable center manifold that contains the points that are locally forward not escaping. This results may apply even when the noise is not artificially added which means that variants of GD such as stochastic gradient descent also fall under this analysis. However, the noise coming from the stochastic gradients may not be good enough in the required directions.

(Levy, 2016) showed how a variant of normalized gradient descent (NGD) could be extended to efficient escape strict saddle points within $\mathcal{O}(\eta^{-2})$ which is an improvement to (Ge et al., 2015a) Noisy gradient descent (Noisy-GD) convergence guarantees. The proposed algorithm is called Saddle-NGD, and it adds zero mean Gaussian noise θn_t with a variance dependent on the dimensionality of the problem once every N_0 iterations. The main difference between this method and Noisy-GD is the fact that only the direction of the gradient is taken into account and the noise is sampled from a different distribution after a certain number of iterations. This method is beneficial because near a saddle point the gradients approach minute values near zero. The normalization step helps to ensure a fast escape from a saddle point because once the algorithm arrives at a saddle point the most negative eigenvalue will be sufficiently large relative to the other eigenvalues. Experimentally Levy (2016) tested his proposed Saddle-NGD algorithm against (Ge et al., 2015a) Noisy-GD method on online tensor data significant in big data applications. The results show that Saddle-NGD has slower initial convergence improvements, compared to Noisy-GD, but after a specific critical point dependent on the learning rate, Saddle-NGD shows significant improvements in decreasing the reconstruction error.

Work presented by (Du et al., 2017) showed that general gradient descent will require exponential time to escape strict saddle points in general non-convex smooth functions using natural random initialization methods (Du et al., 2017). They further tested (Jin et al., 2017a) PGD algorithm, showing that it outperforms GD by taking only polynomial time instead of exponential time to escape saddle points. Through theoretical and experimental work Du et al. (2017) showed that GD takes at least t_d exponential time to escape d saddle points following $t_d \geq (\frac{L+\gamma}{\gamma})^d$. Where L and γ are characteristics of the non-convex function being optimized. The PGD algorithm for the same experiments and theory required an approximately constant number of iterations of approximately $\frac{1}{\eta\gamma}$, where η is the specified learning rate. For general non-convex optimization problems (Du et al., 2017) showed the significant improvements PGD has versus GD with random initialization to reduce the convergence time requirements from exponential time to polynomial time. This can be significant for high dimensional neural network problems to help reduce training time. However, these results cannot be generalized to all non-convex problems, because there can be classes of problems and initialization schemes where GD can have better performance in the presence of saddle points (Du et al., 2017).

1.2. Perturbed Gradient Descent

Here is where the paper selected to review for this project comes in. In (2017a), Jin et al., proposed Perturbed Gradient Descent (PGD), a simple variation of GD capable of achieving convergence to ϵ-second order stationary points in a number of iterations that is almost “dimension-free”, which means that its complexity depends only poly-logarithmically on the problem dimension. This result outperforms previous results for the following reasons:

- Previous works were able to characterize the convergence behavior of GD algorithms either asymptotically or bounding their complexity polynomially in terms of the problem dimensionality. This work attained improved results by providing sharp bounds for second order stationary points
- Their analysis leads to a convergence complexity that matches those of the original GD (Nesterov) up to a poly-logarithmic factor
- Their results apply to a more general class of non-convex functions instead of being problem-specific
- This work is an important step towards reducing the gap between practice and theory in this field, since it better characterizes the behavior observed in practice when optimizing high-dimensional non-convex functions using GD-based algorithms

The dynamics of the PGD algorithm are mostly that of the GD, except that it keeps track of the gradient's norm to identify when the current iteration is close to a stationary point. At that point, PGD adds noise to the current iterate by sampling uniformly from a d-dimensional ball, only a maximum amount of iterations. If the function value does not decrease enough, the algorithm is potentially in a local minimum and returns the current point. Conversely, if the function value decreases enough, it has escaped the saddle point and regular GD iterations are put in place again. The algorithm is shown in Algorithm 1.

Algorithm 1 Perturbed Gradient Descent (PGD)

```
**Input:** $x_0, l, \rho, \epsilon, c, \delta, \Delta_f$
$\chi \leftarrow 3\max\{\log\left(\frac{dl\Delta_f}{c\epsilon^2\delta}\right), 4\}, \eta \leftarrow \frac{c}{l}, r \leftarrow \frac{\sqrt{c}}{\chi^2}\frac{\epsilon}{l}$,
$g_{thres} \leftarrow \frac{\sqrt{c}}{\chi^2}\epsilon . f_{thres} \leftarrow \frac{c}{\chi^3}\sqrt{\frac{\epsilon^3}{\rho}}, t_{thres} \leftarrow \frac{\chi}{c^2}\frac{l}{\sqrt{\rho\epsilon}}$
$t_{noise} \leftarrow -t_{thres} - 1$
**for** $t = 0, 1, ...,$ **do**
  **if** $\|\nabla f(x_t)\| \leq g_{thres}$ and $t - t_{noise} > t_{thres}$ **then**
    $\tilde{x} \leftarrow x_t, t_{noise}$
    $x_t \leftarrow \tilde{x_t} + \xi_t$, $\xi_t$ uniformly $\mathbb{B}_0(r)$
  **end if**
  **if** $t - t_{noise} = t_{thres}$ and $f(x_t) - f(\tilde{x}_{t_{noise}}) > -f_{thres}$ **then**
    **Return** $\tilde{x}_{t_{noise}}$
  **end if**
  $x_{t+1} \leftarrow x_t - \eta\nabla f(x_t)$
**end for**
```

For further clarification, we briefly present the following definitions:

Definition 1. *For a differentiable function f, we say that x is a first-order stationary point if $\|\nabla f(x)\| = 0$; we also say x is an ϵ-first-order stationary point if $\|\nabla f(x)\| \leq \epsilon$.*

Definition 2. *For a differentiable function f, we say that x is a local minimum if x is a first-order stationary point, and there exists $\epsilon > 0$ so that for any y in the ϵ-neighborhood of x, we have $f(x) \leq f(y)$; we also say x is a saddle point if x is a first-order stationary point but not a local minimum. For a twice-differentiable function f, we further say a saddle point x is strict (or non-degenerate) if $\lambda_{min}(\nabla^2 f(x)) < 0$*

Definition 3. *For a ρ-Hessian Lipschitz function f, we say that x is a second-order stationary point if $\|\nabla f(x)\| = 0$ and $\lambda_{min}(\nabla^2 f(x)) \geq 0$; we also say x is ϵ-second-order stationary point if $\|\nabla f(x)\| \leq \epsilon$ and $\lambda_{min}(\nabla^2 f(x)) \geq -\sqrt{\rho\epsilon}$*

Note that these definitions define ϵ-first order and second order stationary points in terms of ϵ to make explicit the relation between the gradient and Hessian. Jin et al., (2017a) main results is stated below:

Theorem 1. (Jin et al., 2017a) *Assume that f satisfies that is l-smooth and ρ-Hessian Lipschitz. Then there exists an absolute constant c_{max} max such that, for any $\delta > 0$, $\epsilon \leq \frac{l^2}{\rho}, \Delta_f \geq f(x_0) - f^*$, and constant $c \leq c_{max}$, $PGD(x_0, l, \rho, \epsilon, c, \delta, \Delta_f)$ will output an ϵ-second-order stationary point, with probability $1-\delta$, and terminate in the following number of iterations:*

$$O\left(\frac{l(f(x_0) - f^*)}{\epsilon^2}\log^4\left(\frac{dl\Delta_f}{\epsilon^2\delta}\right)\right). \quad (1)$$

In addition to this general result, the authors also show analysis for strict saddle property, strong convexity and provide examples of matrix factorization. The proof is based on a geometric interpretation of the perturbation ball achieved by the PGD algorithm. In general, the perturbation region is divided by two disjoint regions; the escaping region which contains all the points where the algorithm can escape the critical point and the stuck region where that is not possible. The shape of these regions is in general not known; however, the authors bound the volume of the stuck region using the smallest eigendirection of $\nabla^2 f(\tilde{x})$ as the thickness of such small band. In this way, they can guarantee that PGD will escape the critical points with high probability. Figure 1 shows graphically the described concept in $2D$ and $3D$. For the former, a thin band is created near the point where the perturbation is performed; in $3D$, it becomes a thin disk.

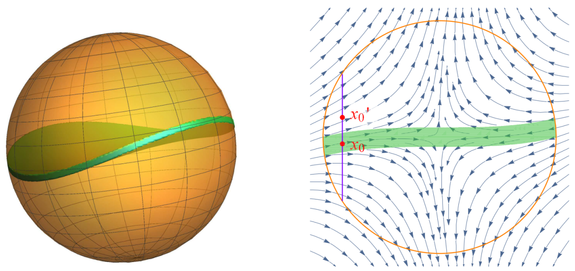

Figure 1. Ball region created by the perturbation in PGD algorithm showing the escaping and stuck region. The region in green corresponds to the stuck region and the authors show that its volume is small. Figure taken from (Jin et al., 2017a).

As stated above, this result shows only a poly-logarithmic convergence on the dimension, which can be written as $\tilde{O}\left(\epsilon^{-2}\right)$ and can be compared to the convergence of original GD (Nesterov) to achieve a ϵ-first order stationary point of $O\left(\frac{l(f(x_0)-f^*)}{\epsilon^2}\right)$, or equivalently $O\left(\epsilon^{-2}\right)$.

1.3. Perturbed Accelerated Gradient Descent

The natural way to improve on the PGD convergence rate involved momentum-based techniques. Previously, (Nesterov) showed that an accelerated version of gradient descent can achieve a $\mathcal{O}(\frac{1}{\sqrt{\epsilon}})$ to an ϵ-first order stationary point which is faster than traditional gradient descent. Jin et al. (2017b) investigated the theoretical possibilities of developing a momentum-based algorithm that could achieve faster convergence then GD in the presence of strict saddle points. Their algorithm is called Perturbed Accelerated Gradient Descent (PAGD) with the pseudo-code presented in Algorithm 2 (Jin et al., 2017b). The first *if-statement* in PAGD is used for the perturbation step. A perturbation to the current position is added when the gradient is small and the previous perturbation was added over $\mathfrak{T}$ iterations ago. The noise is sampled uniformly from a d-dimension ball with radius r (Jin et al., 2017b). The following three lines correspond to the AGD component of the algorithm. The last *if-statement* in PAGD is used for the negative curvature exploitation (NCE). The NCE steps make sure the Hamilton will decrease per iteration. Intuitively this helps to restrict the momentum incorporated in PAGD from growing too large where negative curvature can not be exploited. Jin et al. (2017b), show that their PAGD method is one of the first Hessian-free single loop algorithms to find second-order stationary points faster than GD in $\mathcal{O}(1/\epsilon^{7/4})$ iterations (Jin et al., 2017b).

Algorithm 2 Perturbed Accelerated Gradient Descent (PAGD)

```
Input: initial point $x_0$, step size$\eta$, $\gamma$, $s$, $r$, $\mathfrak{T}$, and $\mathrm{v}_0$.
for $t = 0, 1, ...,$ do
    if $\|\nabla f(x_t)\| \leq \epsilon$ and no perturbation in last $\mathfrak{T}$ steps then
        $x_t \leftarrow x_t + \chi_t \quad \chi_t = Unif(B_0(r))$
    end if
    $y_t \leftarrow x_t + (1-\theta)\, \mathrm{v}_t$
    $x_{t+1} \leftarrow y_t - \eta\nabla f(y_t)$
    $v_{t+1} \leftarrow x_{t+1} - x_t$
    if $f(x_t) \leq f(y_t) + \{\nabla f(y_t), x_t - y_t\} - \frac{\gamma}{2}\|x_t - y_t\|^2$ then
        if $\|v_t\| \geq s$ then
            $x_{t+1} \leftarrow x_t$
        else
            $\delta = s \cdot v_t / \|v_t\|$
            $x_{t+1} \leftarrow \text{argmin}_{x \in (x_t+\delta, x_t-\delta)} f(x)$
            return $(x_{t+1}, 0)$
        end if
    end if
end for
```

1.4. Stochastic Gradient Descent Algorithm

The standard SGD algorithm does not require the complete gradient $\nabla f(\cdot)$ but instead uses a stochastic gradient $\mathbf{g}(\mathbf{x},\theta)$ when at some location $\mathbf{x}$. The θ term is a random variable sampled from a specific distribution $\mathcal{D}$. The SGD algorithm is proven to converge to an ϵ-first order stationary point in $\mathcal{O}(\epsilon^{-4})$ (Ghadimi & Lan, 2013). The general convergence for SGD is shown in Theorem 2.

Theorem 2. (Ghadimi & Lan, 2013) *Let the function f satisfy Lipschitz gradients and the stochastic gradient $\mathbf{g}$ has an expectation equal to the true gradient and the sampling distribution has strongly bounded tails. Let the step size scale as $\eta = \tilde{\Theta}(\ell^{-1}(1+\sigma^2/\epsilon^2)^{-1})$. Then with probability $1-\delta$, SGD will find an ϵ-first order stationary point in the following iteration shown in Eq.2.*

$$\tilde{\mathcal{O}}\left(\frac{\ell(f(x_0)) - f^*}{\epsilon^2} \cdot \left(1 + \frac{\sigma^2}{\epsilon^2}\right)\right) \tag{2}$$

The SGD algorithm pseudo-code is presented in Algorithm 3. This method can also be used in a mini-batch method where several random variables are sampled from the distribution and used to calculate and average stochastic gradient.

Algorithm 3 Stochastic Gradient Descent (SGD)

```
Input: initial point $x_0$, step size $\eta$.
for $t = 0, 1, ...,$ do
    sample $\theta_t \sim \mathcal{D}$
    $x_{t+1} \leftarrow x_t - \eta(\mathrm{g}(x_t;\theta_t))$
end for
```

1.5. Perturbed Stochastic Gradient Descent Algorithm

The PSGD algorithm is a variant of SGD that theoretically has been proven to converge to an ϵ-second-order stationary point in $\tilde{\mathcal{O}}(\epsilon^{-4})$ when the gradients are Lipschitz (Jin et al., 2019). If the Lipschitz assumption cannot be applied then a linear dimensional dependence d arises, bounding convergence to an ϵ-second-order stationary point in $\tilde{\mathcal{O}}(d\epsilon^{-4})$ (Jin et al., 2019). The complete theorem formulation of this convergence criterion is shown in Theorem 3.

Theorem 3. (Jin et al., 2019) *Let the function f satisfy Lipschitz gradients and the stochastic gradient $\mathbf{g}$ has an expectation equal to the true gradient and the sampling distribution has strongly bounded tails. For any ϵ, $\delta > 0$, the PSGD algorithm with chosen parameters (η, r) will find an ϵ-second order stationary point in the following number of iterations in Eq. 3, with probability $1-\delta$.*

$$\tilde{\mathcal{O}}\left(\frac{\ell(f(x_0)) - f^*}{\epsilon^2} \cdot \mathfrak{N}\right) \tag{3}$$

The parameters η, r, and $\mathfrak{N}$ are chosen based on Eq.4-5 by setting theoretical parameters about the optimized functional space.

$$\eta \;=\; \tilde{\Theta}\left(\frac{1}{\ell \cdot \mathfrak{N}}\right), \quad r = \tilde{\Theta}(\epsilon\sqrt{\mathfrak{N}}), \tag{4}$$

$$\text{where } \mathfrak{N} = 1 + \min\left\{\frac{\sigma^2}{\epsilon^2} + \frac{\tilde{\ell}^2}{\ell\sqrt{\rho\epsilon}}, \frac{\sigma^2 d}{\epsilon^2}\right\} \tag{5}$$

Recall that SGD finds convergence to an ϵ-first-order stationary point in $\mathcal{O}(\epsilon^{-4})$ (Ghadimi & Lan, 2013). Thus, PSGD can achieve the same convergence as SGD to a ϵ-second order stationary point with a poly-logarithmic dependence in dimension d. This is significant because convergence to second-order stationary points eliminates the possibility of the convergence point being a strict saddle point. However, that being said the convergence point could either be a local/global minimum or degenerate saddle point.

The general PSGD algorithm pseudo code developed by Jin eta al. (2019) is presented in Algorithm 4 and the mini-batch version is presented in Algorithm 5 (Jin et al., 2019). For both algorithms random noise is added to the gradient each iteration. The noise ξ_t is sampled from a normal distribution with zero mean and covariance $(r^2/d\mathbf{I})$. If r is selected as $r = \tilde{\Theta}(\epsilon)$ then, theoretically PSGD will find an ϵ-second order stationary point following Theorem 3.

Algorithm 4 Perturbed Stochastic Gradient Descent (PSGD)

```
Input: initial point $x_0$, step size $\eta$, perturbation radius $r$.
for $t = 0, 1, ...,$ do
    sample $\theta_t \sim \mathcal{D}$
    $x_{t+1} \leftarrow x_t - \eta(g(x_t;\theta_t) + \xi_t), \quad \xi_t \sim \mathcal{N}(0, (r^2/d)\mathbf{I})$
end for
```

Algorithm 5 Mini-batch Perturbed Stochastic Gradient Descent (Mini-batch PSGD)

```
Input: initial point $x_0$, step size $\eta$, perturbation radius $r$.
for $t = 0, 1, ...,$ do
    sample $\{\theta_t^{(1)}, ..., \theta_t^{(m)}\} \sim \mathcal{D}$
    $g_t(x_t) \leftarrow \sum_{i=1}^{m} g(x_t; \theta_t^{(i)})/m$
    $x_{t+1} \leftarrow x_t - \eta(g(x_t;\theta_t) + \xi_t), \quad \xi_t \sim \mathcal{N}(0, (r^2/d)\mathbf{I})$
end for
```

A very recent work by Fang et al., (2019) showed how SGD can benefit from added dispersive noise and converge to a ϵ-second order stationary point in $\tilde{\mathcal{O}}(\epsilon^{-3.5})$, which improves on the PSGD convergence rate and is the sharpest result so far for stochastic gradient-based methods. Its worth mentioning that these convergence results that are almost dimension free, holding when the gradients are Lipschitz continuous. When this condition does not hold, a linear dependency of the dimensions appears in the analysis (Fang et al., 2019; Jin et al., 2019).

1.6. Beyond first-order methods

Other researchers have contributed important work to approaches that do not rely solely on the gradient, but also use higher-order information. For instance, in (Agarwal et al., 2017), the *FastCubic* algorithm is based on Nesterov's cubic regularization and is capable of finding ϵ-second order stationary points in $\tilde{O}\left(\epsilon^{-7/4}\right)$. Although, these algorithms improve convergence analysis, they require Hessian information which may not be available in large scale machine learning problems.

Similar to the perturbed accelerated version of GD (Jin et al., 2017b), in (O'Neill & Wright, 2017; Sun et al., 2019), the authors analyze the behavior of accelerated methods such as the heavy-ball method when they are close to saddle points by using the stable manifold theorem proving that it is not very likely that accelerated methods get stuck in saddle points and that they can escape them faster than gradient-descent.

(Anandkumar & Ge, 2016) proposes the use of third order derivatives to escape degenerate saddle points. They also show that fourth order derivatives and higher is NP hard. Also, (Reddi et al., 2017) proposes a method that alternates between first-order and second-order subroutines to reduce the complexity of computing Hessians while allowing it to escape saddle points. The problem of escaping saddle points has also been explored in constraint optimization for quadratic objectives subject to convex sets (Mokhtari et al., 2018), as well as in the presence of Riemannian Manifolds (Criscitiello & Boumal, 2019; Sun et al., 2019).

Other important techniques are based on negative curvature information. For example, in (Xu et al., 2018; Allen-Zhu & Li, 2017) the authors propose NEON and NEON2, a methodology in which negative curvature information is extracted from the Hessian using only first order information attaining convergence rate of $\tilde{O}\left(\epsilon^{-4}\right)$ and $\tilde{O}\left(\epsilon^{-3.5}\right)$ respectively. Similarly, by adding regularization to the negative curvatures approach (Allen-Zhu, 2017) attains $\tilde{O}\left(\epsilon^{-3.25}\right)$ using Natasha2. Finally, using variance reduced gradient techniques SPIDER (Fang et al., 2018) is capable of achieving $\tilde{O}\left(\epsilon^{-3}\right)$ which is the current state of the art stochastic gradient computational cost.

1.7. Second order Stationarity in Machine Learning

In machine learning and signal processing non-convex problems it has been shown over the last years that all second-order stationary points are global minima, which means that if one can find second-order stationary points, this is equivalent to globally solving the problem. Areas where this has been shown include Tensor decomposition (Ge et al., 2015a), Dictionary Learning (Sun et al., 2016a), Phase Retrieval (Sun et al., 2016b), Synchronization and Max-Cut (Bandeira et al., 2016), Smooth Semidefinite Programs (Nicolas Boumal & Bandeira, 2016), Matrix sensing (Bhojanapalli et al., 2016), Matrix Completion (Ge et al., 2016) and Robust Principale Components (Rong Ge & Zheng, 2017). In most of these applications, the following is true:

1. All local minima are global minima
2. All saddle points have at least one direction with strictly negative curvature (are strict saddle points)

Note that for a function that meets 1 and 2, all second-order stationary points correspond to global minima. This observation further increases the importance of algorithms capable of finding second-order stationary points efficiently, as those reviewed here, since that guarantees global convergence in these non-convex problems.

One exciting area that has also received significant attention is that of Deep Learning. Several authors had been interested in the surface error of the training process of neural networks. (Dauphin et al., 2014) argues about the proliferation of saddle points when training a neural network and provides experimental evidence in this direction. Also, (Kawaguchi, 2016) provides a theoretical analysis showing that just like in other machine learning areas, in a Deep Network every local minimum is a global minimum and the remaining are saddle points. Furthermore, they argue that in deeper networks degenerate saddle points appear whereas strict saddle points are present in shallow networks (3 layers or less). This was an important result since it helped characterizing the optimization problem that needs to be solved in the training process of a neural network, showing that although it may be difficult it is easier than the general non-convex problem.

A more practical work by (Sankar & Balasubramanian, 2017) showed through an experimental setup that deep neural networks actually converge to saddle points and not to local minima and furthermore that these saddle points are degenerate. They claim that the theoretical work around convergence to saddle points only considers strict saddle points and therefore its motivation is questionable. We note that the definition of strict and degenerate saddle points between this work and the most notorious theoretical works (Ge et al., 2015b; Rong Ge & Zheng, 2017; Ge et al., 2016; Jin et al., 2017a; 2019) differs. For Sankar et al., (2017), degenerate saddle points can have positive, negative and zero eigenvalues of the Hessian. Furthermore, they call the number of zero eigenvalues the degree of degeneracy of the saddle point. For the theoretical works, a strict saddle point only requires the minimum eigenvalue of the Hessian to be negative, meaning that a saddle point with zero, positive and negative eigenvalues is still strict and can be analyzed under their framework.

In (Sankar & Balasubramanian, 2017), the authors characterize the critical point where the algorithms converge when training deep neural networks by explicitly computing the Hessian and its eigenvalue decomposition. Using this, they conclude that actually deep networks converge to saddle points and not to local minima. Although the conclusion somewhat agrees with other's authors observations, we point out that their experiments show one-layer neural networks that achieve comparable performance to deep networks due to the difficulty on computing the Hessian of a practical deep network. We believe that these conclusions can not be extracted from this approximation since the optimization problem may completely show different characteristics when using shallow and deep networks. It is widely known by the community that although shallow networks can match the performance of deep networks, this is not possible by using the gradient-based optimization algorithms that are usually used in this context.

In another work, (Daneshmand et al., 2018) show theoretically that when learning half-spaces using neural networks, the stochastic gradient has strong components in the direction of negative curvature. Using this observation, they propose a variation of PGD (Rong Ge & Zheng, 2017) where the noise is not added isotropic but the iteration is replaced by that of stochastic gradient descent. Their analysis show that this is enough to achieve convergence to second order stationary points.

Based on these theoretical and practical results and observations we propose to study the performance of perturbed gradient-based algorithms such as PGD and SPGD (Rong Ge & Zheng, 2017; Jin et al., 2019), in the training of deep neural networks.

2. Experimental Results

In this section, we will present preliminary results evaluating the perturbed version of stochastic gradient descent (PSGD), using the mini-batch method, versus vanilla stochastic gradient descent (SGD). The experimental setup is outlined at a high level in Fig. 2. This setup describes two main tests where PSGD and SGD convergence rate accuracy and loss are compared using relatively shallow and deep networks. The MLP architecture is considered a relatively shallow network in the experiments due to it containing only 5 layers. Similarly, the VGG3 architecture is considered a relatively deep network because it contains 22 layers. The CIFAR-10 MLP has an input shape of (32,32,3) that is flattened. It then has a dense layer with 128 hidden neurons using sigmoid activation. This is followed by a 25% percent dropout, another dense 32 hidden neuron layer, and lastly 10 output neurons using softmax activation. The CIFAR-10 VGG3 has an input shape of (32,32,32) into a convolution layer followed by a max pooling and dropout layers. This layer sequence is repeated three times using Relu activation with the dropout increasing from 20% to 40%. The last four layers are flattening, dense 128 hidden neurons, 20% dropout, and 10 output neurons using softmax activation.

All networks were developed using Keras back-end onto TensorFlow. The networks were trained in Google Colab using Tesla K80 GPUs with 12 GB of RAM. Due to the execution time limitations set by Google Colab we were only able to run each experiment for 200 epochs. We used 50,000 images from the CIFAR-10 dataset for training and 10,000 for testing the network. In the following sections experimental results are presented for each algorithm based on the shallow and deep network tests depicted in the experimental setup.

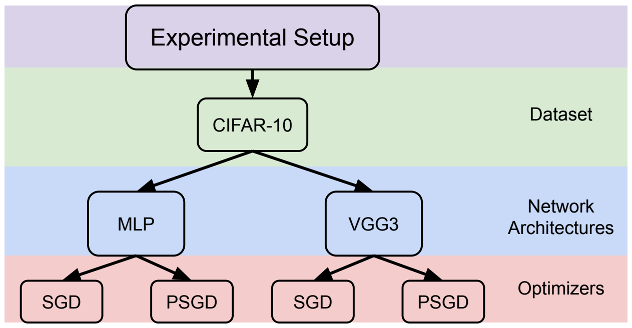

Figure 2. Experimental setup testing the performance of PSGD against SGD on relatively shallow (MLP) and deep (VGG3) networks.

2.1. Shallow Network Results

In this section both training and testing accuracy and loss results are presented for CIFAR10 using the MLP network architecture. The batch size was set to 128 and learning rate $\eta = 0.01$ for all experiments.

The training and test accuracy for SGD and PSGD, at various r values is shown in Fig.3-4 for 200 epochs. The corresponding training and testing loss for both algorithms is shown in Fig.5-6. Focusing on the test accuracy results in Fig.4, the maximum testing accuracy approaches 47% in 200 epochs when the PSGD algorithm is used with $r = 0.015$. When compared to the SGD algorithm the testing accuracy results show a significant increase in the convergence rate for the PSGD algorithm when $r = 0.015$. This increase starts at approximately 25 epochs and continues to converge exponentially faster than SGD with increased epochs. PSGD has a similar trend when $r = 0.1$ but after 150 epochs its convergence rate approximates that of SGD. The same results are present in the training data between PSGD and SGD as shown in Fig.3. However, in the training results the magnitude of convergence improvement in using PSGD with $r = 0.015$ is less pronounced when compared to the performance of SGD.

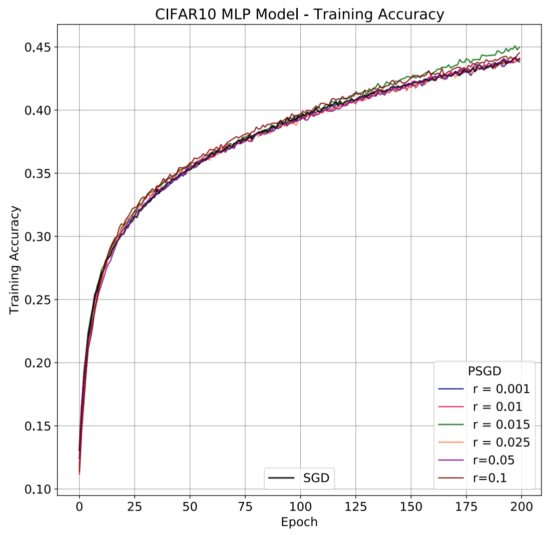

Figure 3. Training accuracy results for PSGD and SGD using a MLP network architecture. Implemented PSGD using mini-batch method described by Algorithm 5 at various r values. PSGD has moderate increase in convergence when $r = 0.1$ and significant increase when $r = 0.015$ compare to SGD. All other PSGD r values tested show negligible differences to SGD convergence rates.

In the testing loss results shown in Fig.6 there appears to be an inflection point around 25 epochs. The PSGD loss for all r values tested was greater than or equal to the SGD loss up to 25 epochs. After this point, up to 200 epochs, the PSGD algorithm with r values of 0.015 and 0.1 have significantly lower loss when compared to the SGD algorithm. When $r = 0.1$ the PSGD algorithm appears to have a fixed magnitude of lower loss compared to the SGD loss. However, when $r = 0.015$ the loss difference between PSGD and SGD increases in magnitude with increased epochs. The training loss shown in Fig.5 presents similar results to the testing

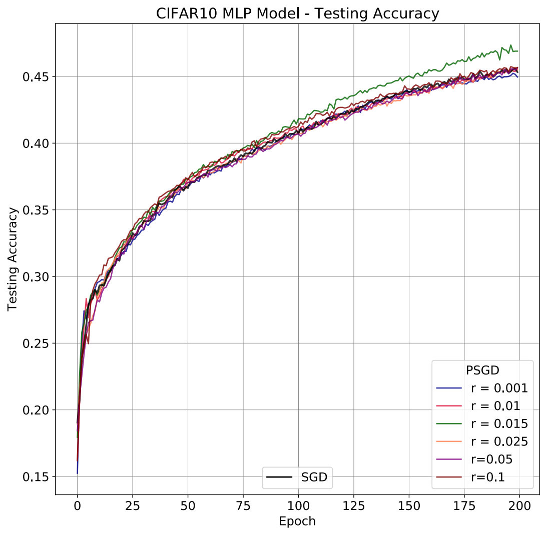

Figure 4. Testing accuracy results for PSGD and SGD using a MLP network architecture. Implemented PSGD using mini-batch method described by Algorithm 5 at various r values. PSGD has moderate increase in convergence when $r = 0.1$ between 10 -150 epochs and significant increase when $r = 0.015$ compare to SGD for all 200 epochs. All other PSGD r values tested show negligible differences to SGD convergence rates.

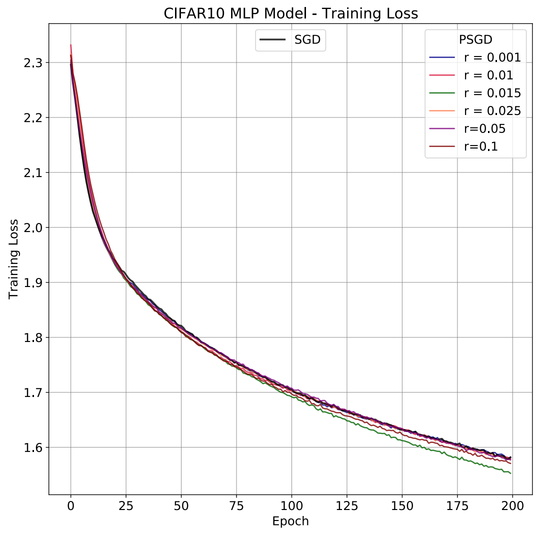

Figure 5. Training loss results for PSGD and SGD using a MLP network architecture. Implemented PSGD using mini-batch method described by Algorithm 5 at various r values. Comparing PSGD to SGD, it has a moderate decrease in L2 loss magnitude when $r = 0.1$ between 25 -150 epochs and significant decrease in L2 loss magnitude when $r = 0.015$ from 25 200 epochs. All other PSGD r values tested show negligible differences to SGD convergence rates.

loss except the PSGD algorithm with $r = 0.1$ starts to moderately converge to the SGD loss after 175 epochs.

Since the previous results shown in Figs.3-6 only represent data from one testing and training cycle, the improved PSGD performance could have been due to the random initialization. Thus, the training and testing process was repeated 10 times to validate that the improved convergence rate with $r = 0.015$ shown in Fig.4 was due to the PSGD algorithm. The average training and testing accuracy results for PSGD with $r = 0.015$ and SGD is shown in Figs.7-8. The corresponding average training and testing loss is shown in Figs.9-10.

The average testing accuracy in Fig.8 shows that the PSGD algorithm repeatably starts to converge faster than the SGD algorithm after approximately 75 epochs. After this point the standard deviation for both algorithms becomes very small supporting the notion that the PSGD algorithm with $r = 0.015$ is converging faster than the SGD algorithm in this experimental setup. The average training accuracy in Fig.7 shows a similar but not as distinguished trend as in the average testing accuracy results.

The average testing loss in Fig.9 supports the average accuracy results with the PSGD loss increasingly measuring a lower loss than SGD after 75 epochs. The significant difference between the PSGD and SGD average testing loss appears to be induced by algorithmic variations because the standard deviation in both results is negligible after 75 epochs. The average training loss in Fig.10 shows the same trend present in the average testing loss results.

Overall, the initial experimental results testing the shallow network in Figs.3-6 highlight the fact that the PSGD algorithm with $r = 0.015$ has significantly better convergence rate than the SGD algorithm for this experimental setup in 200 epochs. The average testing results for PSGD with $r = 0.015$ and SGD shown in Figs.7-10 suggest this convergence increase is partially algorithmically induced. Previous work by Sankar and Balasubramanian (2017) studied the relationship between saddle point degeneracy and neural networks and proposed that convergence points of deep neural networks tend to be saddle points that increase in degeneracy with increased depth (Sankar & Balasubramanian, 2017). Recall that the MLP architecture is 5 layers deep which means it could contain many low degenerate saddle points. However, during training if the gradient descent method converged to a saddle point we should see it plateau

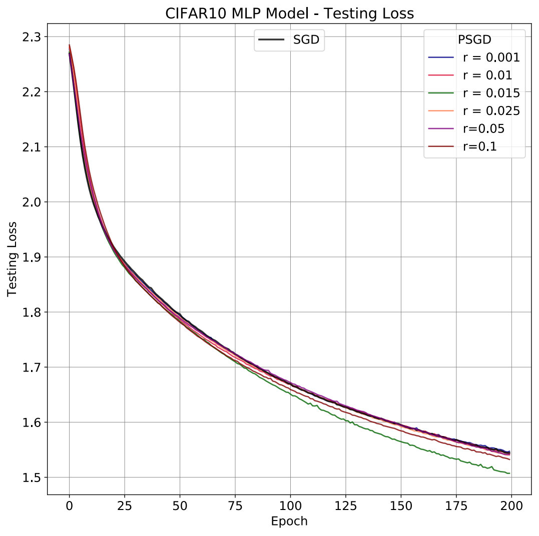

Figure 6. Testing loss results for PSGD and SGD using a MLP network architecture. Implemented PSGD using mini-batch method described by Algorithm 5 at various r values. PSGD shows the same trends, just more pronounced as presented in the training loss results shown in Fig.5.

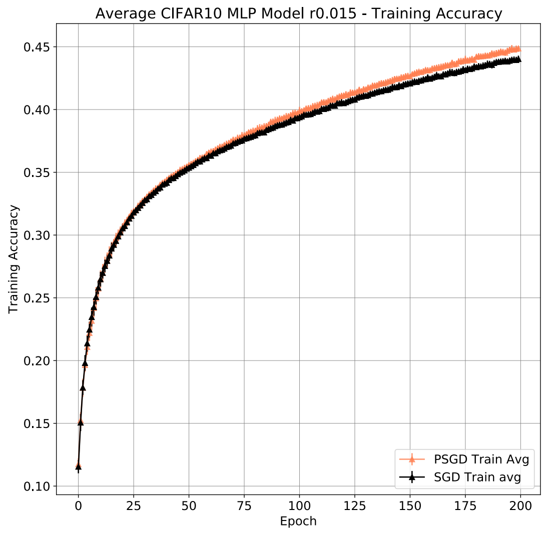

Figure 7. Training accuracy for PSGD using mini-batch method described in Algorithm 5 at r = 0.015 for 10 trials and compared to a single SGD trial. PSGD has an average convergence rate that is moderately improved compared to the SGD average convergence rate.

in accuracy or loss per iteration. It is unclear in the 200 epoch presented results if the SGD algorithm and all other PSGD algorithms with r values other than 0.015 are converging to a degenerate saddle point. If that is the case then based on this experimental setup setting the r magnitude to 0.015 seems to optimally tune the high dimensional zero-mean Gaussian distribution that the gradient perturbation is sampled from in the PSGD algorithm. However, these results are preliminary and require further testing before a conclusion about the practicality of the analyzed algorithms can be made.

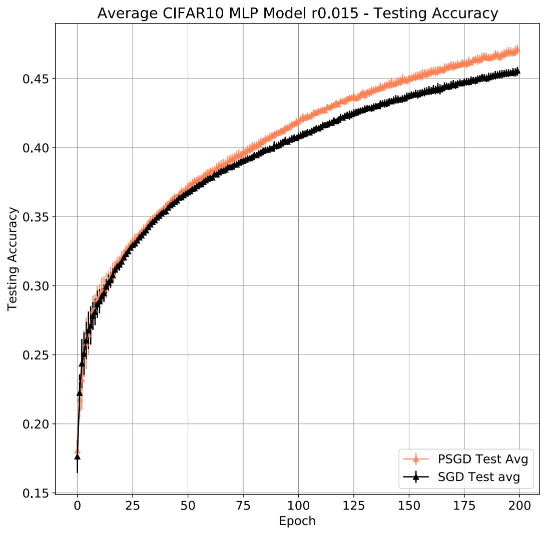

Figure 8. Testing accuracy for PSGD using mini-batch method described in Algorithm 5 at r = 0.015 for 10 trials and compared to a single SGD trial. PSGD has an average convergence rate that is significantly improved compared to the SGD average convergence rate between 75-200 epochs. The PSGD convergence rate is increasingly improving with each epoch compared to SGD.

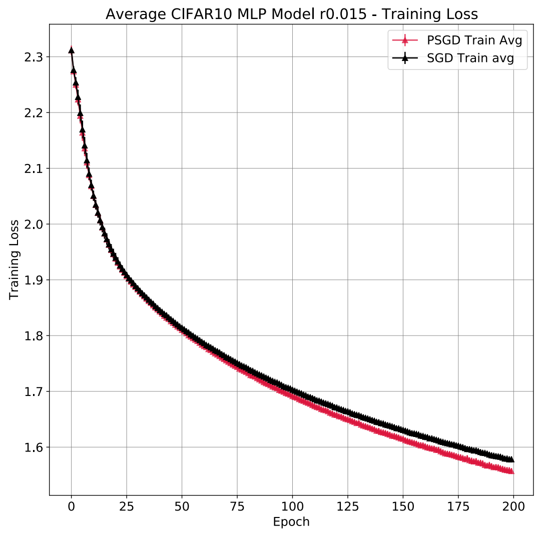

Figure 9. Training loss for PSGD using mini-batch method described in Algorithm 5 at r = 0.015 for 10 trials and compared to a single SGD trial. PSGD has an average ℓ_2 loss is moderately improved compared to the SGD average ℓ_2 loss.

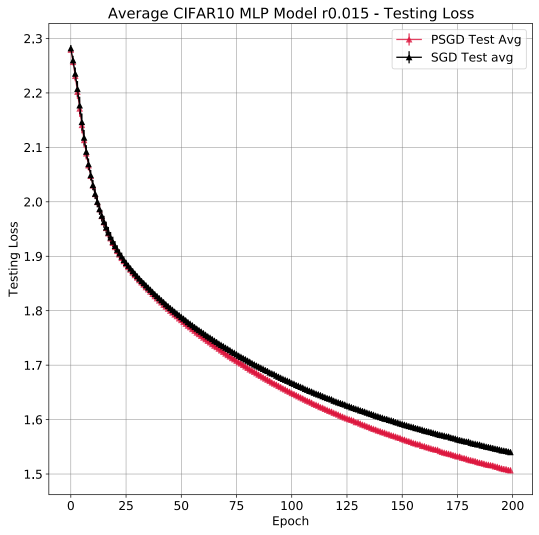

Figure 10. Testing loss for PSGD using mini-batch method described in Algorithm 5 at r = 0.015 for 10 trials and compared to a single SGD trial. PSGD has an average ℓ_2 loss that is significantly improved compared to the SGD average ℓ_2 loss between 75-200 epochs. The PSGD ℓ_2 loss increasingly lowers per epoch compared to the SGD ℓ_2 loss.

2.2. Deep Network Results

In this section both training and testing accuracy and loss results are presented for CIFAR10 using the VGG3 network architecture. The batch size was set to 128 and learning rate $\eta = 0.001$.

The training accuracy over 200 epoch for SGD and PSGD at various r values is shown in Fig.11. The corresponding testing accuracy is shown for the same case in Fig.12. The training and testing loss are presented in Figs.13-14 respectively. The general trend throughout the accuracy plots in Figs.11-12 is increasing the parameter r decreases the convergence rate and overall performance of the network when trained using the PSGD optimizer. In this setup the maximum testing accuracy in 200 epochs approaches 70% when using the SGD optimizer. The SGD optimizer appears to be an upper bound on the performance of the PSGD optimizer during this 200 epoch range with this specific experimental setup. However, the only immediate exception to this claim is when PSGD has the r parameter set to 0.025. At this setting PSGD converges faster then SGD between 20-100 epochs and then converges slower than SGD after 130 epochs.

Further tests were run using PSGD with $r = 0.025$ to verify that the slightly faster convergence compared to SGD, shown in Figs.11-12, was due to the PSGD algorithm and not the random initialization point. The results in Figs.15-16 show the training and testing accuracy for 10 PSGD optimized trials averaged and compared to SGD on the CIFAR-10 dataset using the VGG3 architecture. In Fig.16 the testing accuracy results show that the PSGD algorithm convergence rates have significant varaince between 20-100 epochs and the average approximates the convergence performance of the SGD algorithm. This means that in this specific experimental setup the PSGD algorithm has at a maximum the same performance as SGD both in convergence rate and accuracy.

The training and testing loss plots in Figs.13-14 where PSGD for various r values is compared to SGD, depicts a similar trend as seen in the accuracy plots for this test. The loss magnitude for both training and testing generally increases with and increased r value. Analogous to the accuracy results, the average loss of PSGD with $r = 0.025$ for 10 trials shown in Figs.17-18 support the findings that the PSGD algorithm does not outperform the traditional SGD algorithm for this experimental setup.

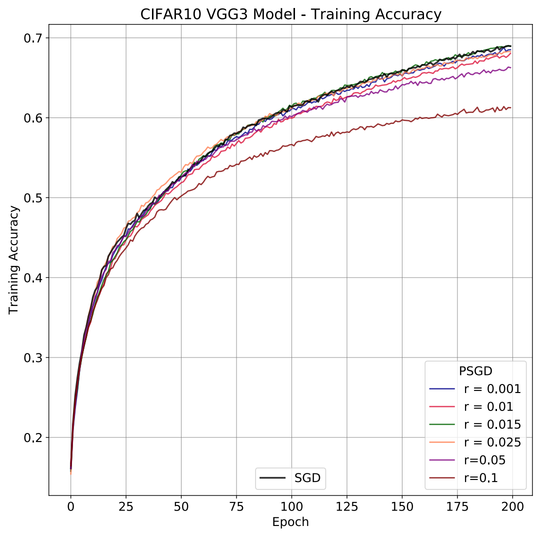

Figure 11. Implemented PSGD using mini-batch method described in Algorithm 5 at various r values. Training accuracy convergence rate is similar to SGD with increasing r decreasing PSGD convergence rate.

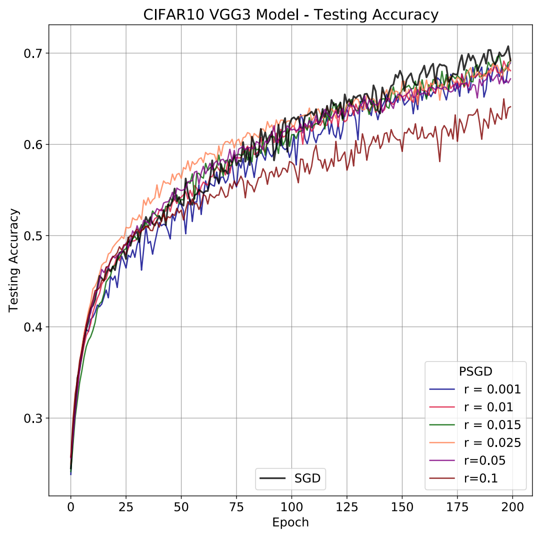

Figure 12. Implemented PSGD using mini-batch method described in Algorithm 5 at various r values. Testing accuracy convergence rate is similar to SGD with increasing r decreasing PSGD convergence rate. Case of PSGD with $r = 0.025$ appears to have improved convergence between 20-100 epochs but this is due to initialization point.

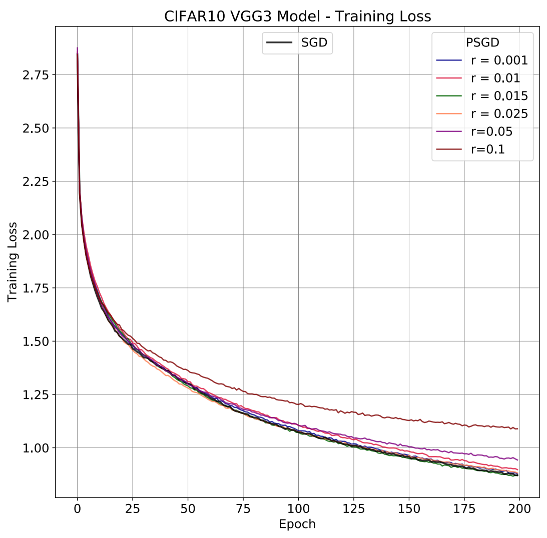

Figure 13. Implemented PSGD using mini-batch method described in Algorithm 5 at various r values. Training loss is similar to SGD with increasing r increasing PSGD loss magnitude.

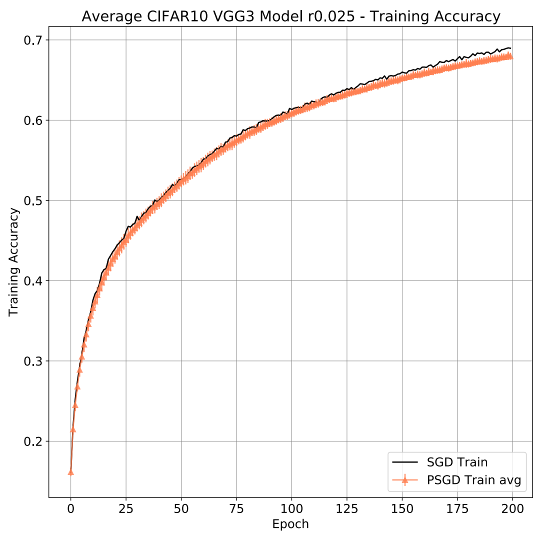

Figure 15. Training accuracy for PSGD using mini-batch method described in Algorithm 5 at $r = 0.025$ for 10 trials and compared to a single SGD trial.

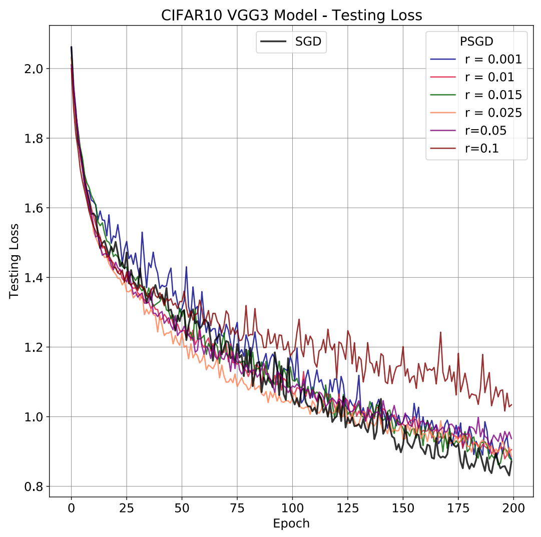

Figure 14. Implemented PSGD using mini-batch method described in Algorithm 5 at various r values. Testing loss is similar to SGD with increasing r increasing PSGD loss magnitude. Case of PSGD with $r = 0.025$ appears to have improved loss between 20-100 epochs but this is due to initialization point.

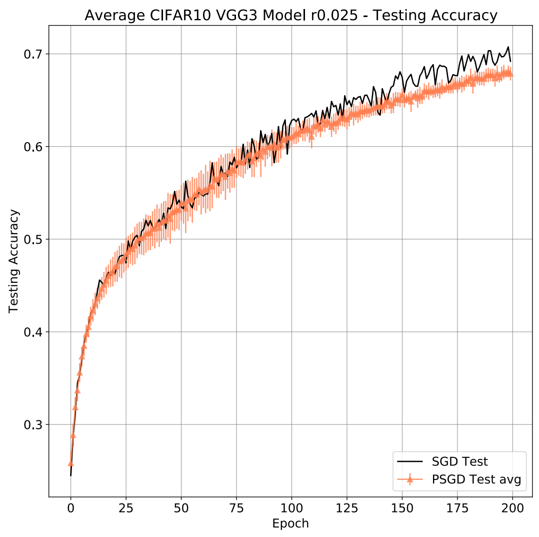

Figure 16. Testing accuracy for PSGD using mini-batch method described in Algorithm 5 at $r = 0.025$ for 10 trials and compared to a single SGD trial. Testing accuracy shows high variance between 20-100 epochs with the average approximating the SGD convergence result.

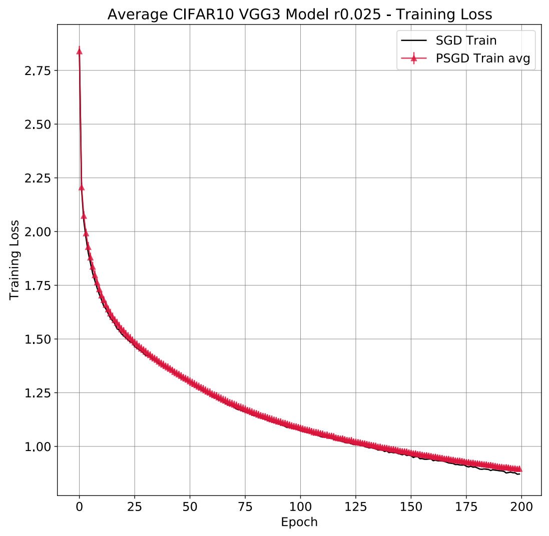

Figure 17. Training loss for PSGD using mini-batch method described in Algorithm 5 at $r = 0.025$ for 10 trials and compared to a single SGD trial.

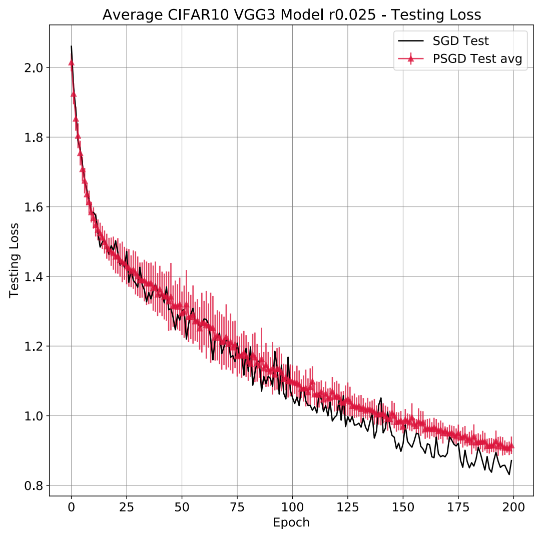

Figure 18. Testing loss for PSGD using mini-batch method described in Algorithm 5 at $r = 0.025$ for 10 trials and compared to a single SGD trial. Testing accuracy shows high variance between 20-100 epochs with the average approximating the SGD convergence result.

3. Conclusion

This work provides a comprehensive literature review of gradient based algorithms designed to efficiently escape saddle points in non-convex settings. The literature review originated from (Jin et al., 2017a) work ”How to Escape Saddle Points Efficiently?”. The authors derived a gradient descent variant algorithm called perturbed gradient descent (PGD) that theoretically will converge to a ϵ-second order stationary point in a number of iterations that is almost ”dimension-free”. A majority of recent work is focused on improving the convergence rate of gradient based algorithms that can still escape strict saddle points efficiently. Future work could be focused on achieving third-order stationary points in the presence of saddle points and determining the GD convergence rates for this condition (Jin et al., 2019). However, significant applications of when third-order stationary points are beneficial will also need to be established.

This work also provides experimental convergence results for perturbed versions of gradient descent on different MLP and VGG3 network architectures using the CIFAR-10 dataset. This is some of the first experimental results testing some of the current proposed theoretical perturbed gradient descent techniques on neural networks. Initial results show an improved experimental convergence rate for perturbed stochastic gradient descent with $r = 0.015$ between 10-150 epochs when compared to general SGD using a MLP architecture. There was no significant improvement to using a perturbed version of SGD with the VGG3 architecture. While results are promising they are preliminary and further experimentation need to be conducted before a conclusion can be made on the practicality of the perturbed version of SGD.

The next stage for these experiments is to run the tests for an increased number of epochs with access to GPUs. These experiments could be further extended to investigate the following open ended questions: possibility of adding scheduled perturbations, auto-detection for adding perturbation when necessary, determining what type of perturbation (uniform dimensional ball or normal distribution) is the best to add. Also investigating ways to use randomized algorithms to approximate the minimized eigenvalue of the Hessian; to help characterize the type of convergence point the algorithms are converging to during each experiment.

References

Agarwal, N., Allen-Zhu, Z., Bullins, B., Hazan, E., and Ma, T. Finding approximate local minima faster than gradient descent, 2017.

Allen-Zhu, Z. Natasha 2: Faster non-convex optimization than sgd, 2017.

Allen-Zhu, Z. and Li, Y. Neon2: Finding local minima via first-order oracles. *CoRR*, abs/1711.06673, 2017. URL http://arxiv.org/abs/1711.06673.

Anandkumar, A. and Ge, R. Efficient approaches for escaping higher order saddle points in non-convex optimization. *CoRR*, abs/1602.05908, 2016. URL http://arxiv.org/abs/1602.05908.

Bandeira, A. S., Boumal, N., and Voroninski, V. On the low-rank approach for semidefinite programs arising in synchronization and community detection. *Conference on Learning Theory*, pp. 361–382, 2016.

Bhojanapalli, S., Neyshabur, B., , and Srebro, N. Global optimality of local search for low rank matrix recovery. *Advances in Neural Information Processing Systems*, pp. 3873–3881, 2016.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. The loss surface of multilayer networks, 2014. URL arXiv:1412.0233.

Criscitiello, C. and Boumal, N. Efficiently escaping saddle points on manifolds, 2019.

Daneshmand, H., Kohler, J. M., Lucchi, A., and Hofmann, T. Escaping saddles with stochastic gradients. *CoRR*, abs/1803.05999, 2018. URL http://arxiv.org/abs/1803.05999.

Dauphin, Y. N., Pascanu, R., Gülçehre, Ç., Cho, K., Ganguli, S., and Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *CoRR*, abs/1406.2572, 2014. URL http://arxiv.org/abs/1406.2572.

Du, S. S., Jin, C., Lee, J. D., Jordan, M. I., Poczos, B., and Singh, A. Gradient descent can take exponential time to escape saddle points, 2017.

Fang, C., Li, C. J., Lin, Z., and Zhang, T. Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 689–699. Curran Associates, Inc., 2018.

Fang, C., Lin, Z., and Zhang, T. Sharp analysis for nonconvex sgd escaping from saddle points, 2019.

Ge, R., Huang, F., Jin, C., and Yuan, Y. Escaping from saddle points - online stochastic gradient for tensor decomposition. *arXiv: 1503.02101v1 [cs.LG]*, pp. 1–46, March 2015a.

Ge, R., Huang, F., Jin, C., and Yuan, Y. Escaping from saddle points — online stochastic gradient for tensor decomposition. In Grnwald, P., Hazan, E., and Kale, S. (eds.), *Proceedings of The 28th Conference on Learning Theory*, volume 40 of *Proceedings of Machine Learning Research*, pp. 797–842, Paris, France, 03–06 Jul 2015b. PMLR. URL http://proceedings.mlr.press/v40/Ge15.html.

Ge, R., Lee, J. D., and Ma, T. Matrix completion has no spurious local minimum. *Advances in Neural Information Processing Systems*, pp. 2973–2981, 2016.

Ghadimi, S. and Lan, G. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013. doi: 10.1137/120880811. URL https://doi.org/10.1137/120880811.

Jin, C., Ge, R., Netrapalli, P., Kakade, S. M., and Jordan, M. I. How to escape saddle points efficiently. *arXiv: 1703.00887v1 [cs.LG]*, pp. 1–35, March 2017a.

Jin, C., Netrapalli, P., and Jordan, M. I. Accelerated gradient descent escapes saddle points faster than gradient descent. *CoRR*, abs/1711.10456, 2017b. URL http://arxiv.org/abs/1711.10456.

Jin, C., Ge, R., Netrapalli, P., Ge, R., Kakade, S. M., and Jordan, M. I. On nonconvex optimization for machine learning: Gradients, stochasticity, and saddle points. *arXiv: 1902.04811v2 [cs.LG]*, pp. 1–31, September 2019.

Kawaguchi, K. Deep learning without poor local minima, 2016.

Lee, J. D., Simchowitz, M., Jordan, M. I., and Recht, B. Gradient descent converges to minimizers, 2016.

Levy, K. Y. The power of normalization: Faster evasion of saddle points. *CoRR*, abs/1611.04831, 2016. URL http://arxiv.org/abs/1611.04831.

Mokhtari, A., Ozdaglar, A. E., and Jadbabaie, A. Escaping saddle points in constrained optimization. *CoRR*, abs/1809.02162, 2018. URL http://arxiv.org/abs/1809.02162.

Nesterov, Y. *Introductory lectures on convex programming volume i: Basic course*.

Nicolas Boumal, V. V. and Bandeira, A. The non-convex burer-monteiro approach works on smooth semidefinite programs. *Advances in Neural Information Processing Systems*, pp. 2757–2765, 2016.

O'Neill, M. and Wright, S. J. Behavior of accelerated gradient methods near critical points of nonconvex functions, 2017.

Reddi, S. J., Zaheer, M., Sra, S., Póczos, B., Bach, F. R., Salakhutdinov, R., and Smola, A. J. A generic approach for escaping saddle points. *CoRR*, abs/1709.01434, 2017. URL http://arxiv.org/abs/1709.01434.

Rong Ge, C. J. and Zheng, Y. No spurious local minima in nonconvex low rank problems: A unified geometric analysis. *arXiv preprint arXiv:1704.00708*, 2017.

Sankar, A. R. and Balasubramanian, V. N. Are saddles good enough for deep learning?, 2017.

Sun, J., Qu, Q., and Wright., J. Complete dictionary recovery over the sphere i: Overview and the geometric picture. *IEEE Transactions on Information Theory*, 2016a.

Sun, J., Qu, Q., and Wright., J. A geometric analysis of phase retrieval. *2016 IEEE International Symposium on Information Theory (ISIT)*, pp. 2379–2383, 2016b.

Sun, T., Li, D., Quan, Z., Jiang, H., Li, S., and Dou, Y. Heavy-ball algorithms always escape saddle points, 2019.

Xu, Y., Jin, R., and Yang, T. First-order stochastic algorithms for escaping from saddle points in almost linear time, 2018.