Better Generalization by Efficient Trust Region Method

Anonymous Authors[1]

Abstract

In this paper, we develop an efficient trust region method for solving non-convex optimization problems, especially for training deep neural networks. At each iteration, trust region method computes the search direction by solving a non-convex subproblem. Solving this subproblem is non-trivial—existing methods have only sub-linear convergence rate. We show that a simple modification of gradient descent algorithm can converge to a global minimizer of the subproblem with an asymptotic linear convergence rate. Moreover, our method only requires Hessian-vector products, which can be computed efficiently by back-propagation in neural networks. We show that our method is much faster than existing second order methods for training large-scale convolutional neural networks such as VGG and MobileNets. Furthermore, our method converges to a solution with lower generalization (test) error than SGD, and the difference becomes even more significant in large batch training. Finally, we conduct several interesting experiments to support our conjecture that the trust region method can avoid sharp local minima.

1. Introduction

Training neural network requires solving a large-scale non-convex optimization problem. Despite having many optimization algorithms in the literature (Nesterov, 2013; Wright & Nocedal, 1999; Boyd & Vandenberghe, 2004), by far the most popular methods for training neural networks are SGD and its variants (Zeiler, 2012; Duchi et al., 2011; Kingma & Ba, 2014; Hinton et al., 2006). Compared to these stochastic first order methods, second order methods are rarely used in training deep neural networks due to large-scale data and high dimensional parameter space. Prior to our work, there aren't many works on second order methods for neural network optimization (Martens, 2010; Kiros, 2013; Botev et al., 2017), and most of them are tested on multi-layer perceptrons. The most practical implementation is K-FAC (Martens & Grosse, 2015; Grosse & Martens, 2016) in tensorflow, but it is still slow in terms of training time.

In this paper, we develop an efficient trust region method for non-convex optimization, especially for training deep neural networks. At each iteration, trust region method approximates the objective function by a quadratic term with a norm constraint, known as the trust-region subproblem. The main difficulty in large-scale applications is how to solve this *nonconvex* subproblem efficiently (Yuan, 2000). Approximate solvers such as (Steihaug, 1983; Powell, 1970) cannot converge to the exact minimizer, while the best method for computing the global minimizer has sub-linear convergence rate (Hazan & Koren, 2016). We propose a new solver that is guaranteed to converge to the global minimum of the non-convex trust region sub-problem with an asymptotic linear convergence rate. What's more, our method only requires Hessian-vector product, which can be computed efficiently in neural networks. Empirically we show our method is much faster than other existing second order methods in terms of running time.

Apart from convergence rate, generalization ability is another important issue to practitioners. Starting from (Zhang et al., 2016) which proposed the generalization paradox of deep models, intensive works have been done to find ways to explain and close generalization gap in large batch SGD (Keskar et al., 2016; Kawaguchi et al., 2017; Wilson et al., 2017). Interestingly, we find through experiments that using trust region method alone can effectively escape from sharp minima and achieve lower test error than SGD, especially on large-batch training. For CIFAR-10 with VGG network, our trust region based method achieves 86.8% accuracy and our hybrid method have 88.1%, both of them are better than SGD's solution with 86.7% accuracy when batch size is 128. As batch size grows to 2048, our trust region method still has 85.5% accuracy and our hybrid method is 87.0%, in contrast, SGD only has 80.3% accuracy. Arguably, our method is a more natural solution to large batch training because it can match the small batch accuracy simply by switching to trust region method without changing network structure, loss function and batch size.

2. Background and Related Work

2.1. (Stochastic) Trust region method

Algorithm 1 (Stochastic) Trust region method

1: **Input:** Objective function $F(x)$.
2: **Return:** Suboptimal solution: x_{N+1}.
3: **for** $t = 0, 1, \cdots, N$ **do**
4: Calculate gradient $g = \nabla F(x_t)$, $H = \nabla^2 F(x_t)$ or their stochastic counterparts
5: Expand objective function:

$$\arg\min_{z:\|z\|_2 \le r_t} f(z) = \frac{1}{2} z^\intercal H z + g^\intercal z \tag{1}$$

6: Solve (1) to get z_t, compute $q_t = \frac{F(x_t) - F(x_t + z_t)}{f(0) - f(z_t)}$
7: (Deterministic version) Update iterate and radius:

- If $q_t \ge 0.9$, then $x_{t+1} = x_t + z_t$, $r_{t+1} = 2r_t$.
- If $q_t \ge 0.1$, then $x_{t+1} = x_t + z_t$, $r_{t+1} = r_t$.
- If $q_t < 0.1$, then $x_{t+1} = x_t$, $r_{t+1} = 0.5 r_t$.

8: (Stochastic version) Update iterate and radius:

- If $q_t \ge 0.9$ and $\|g\| \ge r_t$, then $x_{t+1} = x_t + z_t$, $r_{t+1} = \min\{2r_t, r_{\max}\}$.
- Otherwise, $x_{t+1} = x_t$, $r_{t+1} = 0.5 r_t$.

9: **end for**

We consider solving a general non-convex optimization problem:

$$\min_x F(x).$$

Trust region (TR) method (Conn et al., 2000) (Algorithm 1) is a classical second order method for solving nonlinear optimization problems. At each iteration, TR forms a second order approximation around the current solution and updates the solution by minimizing this quadratic function within the "trust region". The trust region, usually a hyper ball, is chosen to make the second order approximation close enough to the original one and thus by minimizing the subproblem, the objective function will very likely to descent. More formally, at iteration x_t the subproblem can be written as:

$$\arg\min_z f(z) := \frac{1}{2} z^\intercal H z + g^\intercal z, \ \text{ s.t. } \|z\|_2 \le r. \tag{2}$$

Where $H = \nabla^2 F(x_t)$, $g = \nabla F(x_t)$, and $r > 0$ is the radius of trust region.[1]

If objective function F has finite sum structure, $F(x) = \frac{1}{n}\sum_{i=1}^n f_i(x)$ or more generally $F(x) = \mathbb{E}_\xi f(x, \xi)$ where ξ is random variable with bounded variance, then we can turn to stochastic trust region method discussed in (Chen et al., 2015), where both $\nabla F(x_t)$ and $\nabla^2 F(x_t)$ is computed by subsamples. Both deterministic and stochastic trust region method have very similar algorithm as one can see in Algorithm 1. As to convergence rate, (Blanchet et al., 2016) shows that the stochastic version is guaranteed to find a stationary point $\|\nabla f(x)\| \le \epsilon$ in $\mathcal{O}(\epsilon^{-2})$ iterations, while the deterministic version is able to find ϵ-suboptimal stationary point within $\mathcal{O}(\epsilon^{-3/2})$ iterations (Curtis et al., 2017). For large scale problem when we cannot calculate the gradient and Hessian on the whole dataset, stochastic method is the only choice.

The main computation in TR is to solve the trust region subproblem. Since the Hessian matrix H may not be positive definite, (2) is a non-convex problem with constraint and it is hard to compute its global minimizer. Furthermore, for large-scale applications (e.g., training deep networks), H is very large (millions by millions) and cannot be formed explicitly. Several algorithms have been proposed in the literature. Traditionally, dogleg method (Powell, 1970) for $H \succ 0$ and Steihaugs method (Steihaug, 1983) for indefinite H can be applied to get an approximated solution. However if a more precise solution is desired, we need to find other ways, e.g. (Hazan & Koren, 2016) is able to find an ϵ-suboptimal solution in $\tilde{\mathcal{O}}(M/\sqrt{\epsilon})$ time, where M is the matrix size (number of nonzero elements). Other methods include subspace minimization (Erway & Gill, 2009; Erway et al., 2009), cone based methods (Ho-Nguyen & Kilinc-Karzan, 2017), etc. In this paper, we design an efficient solver based purely on gradient information that can find an ϵ-suboptimal solution in asymptotically linear time $\tilde{\mathcal{O}}(M \log(1/\epsilon))$. Furthermore, our method only requires matrix-vector products, which can be computed efficiently for neural networks.

2.2. Other Second Order Methods for Training Neural Networks

Another related technique for non-convex optimization is adaptive regularization with cubic (ARC) (Griewank, 1981). Similar to TR method, ARC uses cubic term of positive coefficient $\frac{\sigma}{3}\|z\|_2^3$ to replace constraint in (2):

$$\arg\min_z \frac{1}{2} z^\intercal H z + g^\intercal z + \frac{\sigma}{3}\|z\|_2^3. \tag{3}$$

ARC method can be seen as the "twin" method of TR, since they share many similar properties (e.g. converge rates). Intuitively the radius parameter r in (2) plays the same role as σ in ARC: a larger r and smaller σ both mean larger "trust-region". However, since the ARC subproblem is unconstrained, the treatment is relatively easy: for example, generalized Lanczos-type iterations (Gould et al., 1999) or gradient descent (Carmon & Duchi, 2016) can be used. Our

[1] In general the constraint can be $\|\cdot\|_M$ for any PSD matrix M, but we only discuss the simpler case here because a change-of-variable $z' = \sqrt{M} z$ and $H' = \sqrt{M}^{-\intercal} H \sqrt{M}^{-1}$ can transform it to (2).

proposed subproblem solver for TR is also based on gradient descent, but due to constraint in the TR subproblem our algorithm and proof are different from (Carmon & Duchi, 2016).

Recently, several works have applied second order methods to neural network optimization, including (Martens, 2010; Botev et al., 2017; Xu et al., 2017b;a; Martens & Grosse, 2015; Grosse & Martens, 2016). However, many of them only consider small-scale fully connected networks with few layers. Among them, K-FAC (Martens & Grosse, 2015; Grosse & Martens, 2016) makes use of curvature information by efficiently approximating Fisher matrix and turns out to be successful in convolutional neural networks; (Xu et al., 2017b;a) adopt full gradient and subsampled Hessian and solve the trust region subproblem in the subspace spanned by Cauchy point and minimum eigenvector $v_1(H)$. We are different from this recent paper in two aspects. First, we develop a new trust region subproblem solver with better theoretical guarantee. Second, the full gradient computation in their algorithm is too expensive for large-scale deep network training. In contrast, we implement our algorithm with subsampled gradient and Hessian, thus we are able to run large-scale experiments and observe better generalization compared with SGD.

2.3. Generalization Ability for First Order Methods

SGD and other stochastic first order methods are popular for training large scale neural networks since gradient for each training sample can be computed in one forward-backward cycle. Recently there are many works on the effect of batch size on training time as well as generalization ability. Indeed, a larger batch makes more use of parallel processors, so the program can scale perfectly to more GPUs. Moreover, more samples make the gradient less noisy and thus a larger learning rate and higher converge rate are possible, see (Goyal et al., 2017; You et al., 2017a;b) and references therein.

At the same time, large batch in SGD is harmful to generalization ability, experiments in (Keskar et al., 2016) show that batch size can affect the geometry around the solution that SGD finds. Specifically, SGD with a larger batch size tends to find sharper local minima, so even a tiny change of data from training set to test set will lead to high loss, i.e. bad generalization ability. As to the converge rate to local minima, although (Lee et al., 2016) shows gradient descent always converges to local minima, (Du et al., 2017) proves the converge process can take exponential time. On the other hand by adding extra noise, (Ge et al., 2015) claims SGD is able to escape strict saddle in poly(d/ϵ) iterations, and further (Jin et al., 2017) design a full gradient with noise method that improves the result to poly-$\log(d)/\epsilon^2$ steps.

Notations: Throughout this paper, we use $\mathcal{S}^{n-1}$ to denote the unit sphere in $\mathbb{R}^n$. The model parameters $x \in \mathbb{R}^d$ and objective function $F(x) = \frac{1}{N}\sum_{i=1}^N f_i(x)$ is in $\mathcal{C}^2$; its gradient and Hessian at step x_t are denoted as $g_t = \nabla F(x_t)$ and $H_t = \nabla^2 F(x_t)$, and if the index t is unimportant we will simplify them as g and H respectively. For the Hessian matrix H, its eigenvalue decomposition is $H = \sum_{i=1}^d \lambda_i v_i v_i^\intercal$ such that $v_i^\intercal v_j = \delta_{ij}$ and $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_d$, accordingly we decompose any vector $a \in \mathbb{R}^d$ as $a = \sum_{i=1}^d a^{(i)} v_i$. Denote operator norm $\beta = \|H\|_{\text{op}} = \max\{|\lambda_1|, |\lambda_d|\}$. $\|\cdot\|$ represents 2-norm if not stated explicitly, $\mathrm{I}_d \in \mathbb{R}^{d\times d}$ is the identity matrix. Finally, following (Carmon & Duchi, 2016) we use s to represent the global minimizer of trust region subproblem (2).

3. Efficient Trust Region subproblem solver

Algorithm 2 Proposed trust region subproblem solver

Input: Gradient g, Hessian matrix H
Return: Approximated solution z
Initialize z_0 and η so that Assumption 1,2 are met.
for $t = 0, 1, \ldots,$ max_iter **do**
  if $\|z_t\| < 1$ **then**
    $z_{t+1} = z_t - \eta(Hz_t + g)$
  else
    break
  end if
end for
$z_{t'} = z_t$
for $t' = t, t+1, \ldots,$ max_iter **do**
  Choose $\alpha_{t'}$ by Armijo line search
  $z_{t'+1} = \frac{z_{t'} - \alpha_{t'}(\mathrm{I}_d - z_{t'}z_{t'}^\intercal)(Hz_{t'}+g)}{\|z_{t'} - \alpha_{t'}(\mathrm{I}_d - z_{t'}z_{t'}^\intercal)(Hz_{t'}+g)\|}$
end for
Return $z_{\text{max_iter}}$

In this section, we show an efficient gradient-descent based algorithm with proper initialization can find the global minimum of the trust region subproblem (2). For simplicity, we assume trust region radius $r = 1$. If the global minimum lies inside of the sphere then gradient descent itself is guaranteed to find it; Otherwise we first conduct gradient descent until the iterate hits the spherical constraint, then a manifold gradient descent on the sphere can converge to the solution. We prove this simple procedure can return the global minimum of the non-convex trust region subproblem and has asymptotically linear convergence rate. The details are shown in Algorithm 2.

3.1. Properties of global minimum

The necessary and sufficient condition of global minimum comes from KKT condition, see (Wright & Nocedal, 1999) for details.

Lemma 1. *(Global minimum) s is the global minimum of*

(2) *if and only if* $\|s\| \leq 1$ *and there is a scalar* $\lambda \geq 0$ *such that:*

$$\begin{aligned} \textit{gradient condition: } & (H + \lambda \mathrm{I}_d)s + g = 0, \\ \textit{complementary slackness: } & \lambda(1 - \|s\|_2) = 0, \\ \textit{Hessian: } & H + \lambda \mathrm{I}_d \succeq 0. \end{aligned} \tag{4}$$

Proof details are postponed to appendix. When $\lambda_1 \geq 0$, the objective is convex, so each stationary point is a global minimum, and further if $\|s\| < 1$ then we only need to use gradient descent to solve it (that the case for Line 11 in Algorithm 2), otherwise $\|s\| = 1$ and our manifold based algorithm also applies. So in the following we only need to consider the non-convex case ($\lambda_1 < 0$). Notice that in this case we always have $\|s\| = 1$. In the following lemma we try to distinguish global minimum from other stationary points, restricted to $g^{(1)} \neq 0$ case (we call it "easy-case", as oppose to "hard-case" in (Wright & Nocedal, 1999)). As to the "hard-case" $g^{(1)} = 0$, in theory we can apply a small perturbation to b as (Carmon & Duchi, 2016) does: $b' = b + \varepsilon$ where ε is a small Gaussian noise. In practice due to rounding error this case is hardly seen so we choose to ignore it in our implementation.

The following lemma gives a sufficient condition of global minimum:

Lemma 2. *When* $g^{(1)} \neq 0$ *and* $\lambda_1 < 0$*, among all stationary points if* $s^{(1)}g^{(1)} \leq 0$ *then* s *is the global minimum.*

3.2. Convergence of iteration

To prove our algorithm converges to a global minimum, we need to show that the solution found by Algorithm 2 will satisfy Lemma 2. We show this is true under the following assumptions:

Assumption 1. *(Bounded step size) Step size* $\eta < 1/\beta$.

Assumption 2. *(Initialize)* $z_0 = -\alpha \frac{g}{\|g\|}$, $0 < \alpha < 1$.

We remark that Assumption 2 is a good guess of global minimum if ignoring the curvature information, and using this alone as subproblem solution will reduce trust region method to gradient descent. Under these assumptions, Algorithm 2 is guaranteed to find (one of) the global minimum, which is formally stated in the following theorem:

Theorem 3. *Under proximal gradient descent update:* $z_{t+1} = \mathsf{Prox}_{\|\cdot\|}\big(z_t - \eta \nabla f(z_t)\big)$*, and Assumption 1 if* $z_t^{(i)} g^{(i)} \leq 0$ *then* $z_{t+1}^{(i)} g^{(i)} \leq 0$*. Combining with Assumption 2 and Lemma 2, if* $\lambda_1 < 0$, $g^{(1)} \neq 0$ *then* z_t *converges to a global minimum* s*.*

Here we use $\mathsf{Prox}_{\|\cdot\|}$ to include both the cases in Line 6 and Line 14 in Algorithm 2. To see the converging process more clearly, we divide the iterations into two phases: in the first phase z_t stays strictly inside the sphere $\mathcal{S}^{n-1}$: $\|z_t\| < 1$, and during this phase we will show that $\|z_t\|$ is monotone increasing until it hits the sphere and that is when the second phase begins. In the second phase the iterates adhere to the sphere and converge to the global minimum with an asymptotic linear rate. First of all we show the monotone increasing property based on following lemma:

Lemma 4. *For* z_t *defined above, we have* $z_t^\intercal H \nabla f(z_t) \geq \beta z_t^\intercal \nabla f(z_t)$ *(recall we define* β *as the operator norm of* H*).*

Now *imagine* there is another iterate $\tilde{z}_t$ that does "plain" GD (i.e. without projection): $\tilde{z}_{t+1} = \tilde{z}_t - \eta \nabla f(\tilde{z}_t)$, using the same step size as z_t and same initialization $\tilde{z}_0 = z_0$. Such a iteration rule guarantees that as long as $\|z_t\| < 1$ for all $z \leq \tau$ then $\tilde{z}_t = z_t$. By showing $\|\tilde{z}_t\|$ is monotone increasing, we actually prove monotone increasing property of $\|z_t\|$ in the first phase.

Theorem 5. *Suppose* $\tilde{z}_t$ *is in the region such that proximal gradient update equals to plain GD:* $\tilde{z}_{t+1} = \tilde{z}_t - \eta \nabla f(\tilde{z}_t)$*, then under this update rule,* $\|\tilde{z}_t\|$ *is monotone increasing.*

Another key observation from Theorem 5 is the unboundedness of $\|\tilde{z}_t\|$, i.e. $\|\tilde{z}_t\| \to \infty$ as $t \to \infty$ (shown in Eq. (A3) in appendix). Indeed we have the following lemma:

Lemma 6. *(Finite phase I) Assuming* $\lambda_1 < 0$*, suppose* t^* *is the index that* $\|\tilde{z}_{t^*}\| < 1$ *and* $\|\tilde{z}_{t^*+1}\| \geq 1$*, then* t^* *is bounded by:*

$$\begin{aligned} t^* \leq \log(1 - \eta\lambda_1)^{-1} \Big[& \log\big(\frac{1}{\eta|g^{(1)}|} - \frac{1}{\eta\lambda_1}\big) \\ & - \log\big(\frac{-\tilde{z}_0^{(1)}}{\eta g^{(1)}} - \frac{1}{\eta\lambda_1}\big)\Big]. \end{aligned} \tag{5}$$

So after t^* iterations the algorithm will reach phase II. Then we turn to manifold optimization method, which is discussed extensively in (Absil et al., 2009) and for the manifold theory we refer to (Do Carmo & Flaherty Francis, 1992). Here we list some concepts and its explicit form in our problem: denote $\mathcal{M}$ as the smooth manifold, and $z \in \mathcal{M}$ can be any point on manifold $\mathcal{M}$. The *tangent space* of z, denoted as $\mathcal{T}_z$, is the set of all tangent vectors to $\mathcal{M}$ at z. When $\mathcal{M} = \mathcal{S}^{n-1}$ then $\mathcal{T}_z = \{\xi \in \mathbb{R}^n : z^\intercal \xi = 0\}$. A *Retraction* is a mapping $R_z(\xi) : \xi \in \mathcal{T}_z \mapsto \mathcal{M}$, in $\mathcal{S}^{n-1}$ manifold, one of the commonly used retraction is $R_z(\xi) = \frac{z+\xi}{\|z+\xi\|}$; *Projection* onto $\mathcal{T}_z$, denoted as $P_z\xi$ is a mapping from $\mathbb{R}^n$ to $\mathcal{T}_z$, for $\mathcal{S}^{n-1}$ the projection is simply $P_z(\xi) = (\mathrm{I}_d - zz^\intercal)\xi$. Based on the above concepts, we can write the gradient descent on $\mathcal{M}$ as: $z_{t+1} = R_{z_t}(-\alpha_t \mathrm{grad} f(z_t))$, and α_t is tactically selected by line search such that $f(z_t) - f(z_{t+1}) \geq \sigma\alpha_t \|\mathrm{grad} f(x_t)\|$, where $\sigma \in (0, 1)$.

Because we already know at least one of the global minimum lies on $\mathcal{S}^{n-1}$, and according to Lemma 6 after at most t^* iterations we start to use manifold based optimization, i.e.

the subproblem becomes:

$$\min_{z\in\mathcal{S}^{n-1}} f(z) = \frac{1}{2}z^\intercal Hz + g^\intercal z. \tag{6}$$

Therefore, our method shrinks the search space from $\|z\| \leq 1$ to $\|z\| = 1$, by doing so we can apply well studied manifold optimization theory (Absil et al., 2009; Udriste, 1994) to our problem. Indeed we have following theorems:

Theorem 7. *Let $\{z_t\}$ be an infinite sequence of iterates generated by line search gradient descent, then every accumulation point of $\{z_t\}$ is a stationary point of the cost function f.*

This above only guarantees convergence to stationary points, however, according to Theorem 3 if the step size α_t is not too large, it actually converges to global minimum. Then it remains to show a linear convergence rate, as guaranteed by the following theorem:

Theorem 8. *Let $\{z_t\}$ be an infinite sequence of iterates generated by line search gradient descent, suppose it converges to s. Let $\lambda_{H,\min}$ and $\lambda_{H,\max}$ be the smallest and largest eigenvalues of the Hessian at s. Assume that s is a local minimizer then $\lambda_{H,\min} > 0$ and given r in the interval $(r_*, 1)$ with $r_* = 1-\min\big(2\sigma\bar{\alpha}\lambda_{H,\min}, 4\sigma(1-\sigma)\beta\frac{\lambda_{H,\min}}{\lambda_{H,\max}}\big)$, there exists an integer K such that:*

$$f(z_{t+1}) - f(s) \leq r\big(f(z_t) - f(s)\big),$$

for all $t \geq K$.

Remarks: $\lambda_{H,\min}$ and $\lambda_{H,\max}$ are the minimum and maximum eigenvalue of Riemannian Hessian. Specifically the Riemannian Hessian can be calculated by (Proposition 5.5.4, Absil et al. (2009)):

$$\begin{aligned} \text{Hess } f(x) &= \text{Hess}(f\circ \text{Exp}_x)(0_x), \\ \langle \text{Hess } f(x)[\xi],\xi\rangle &= \langle \text{Hess } (f\circ \text{Exp}_x)(0_x)[\xi],\xi\rangle. \end{aligned} \tag{7}$$

By direct calculation (in appendix) we shall see: $\langle \text{Hess} f(s)[\xi],\xi\rangle = -s^\intercal Hs + \xi^\intercal H\xi - g^\intercal s$, where $\xi \in \text{Null}(s)$, $\|\xi\| = 1$. By optimal condition (4), we have:

$$s^\intercal(H+\lambda \text{I}_d)s + s^\intercal g = 0 \Rightarrow -s^\intercal Hs - g^\intercal s = \lambda, \tag{8}$$

so $\langle \text{Hess} f(s)[\xi],\xi\rangle = \lambda + \xi^\intercal H\xi \geq \lambda + \lambda_1 \overset{!}{>} 0$. Where $\overset{!}{>}$ is guaranteed by gradient condition in (4):

$$(\lambda_1 + \lambda)s^{(1)} + g^{(1)} = 0, \tag{9}$$

in "easy-case", $g^{(1)} \neq 0$, so $\lambda_1 + \lambda \neq 0$ and Hessian condition in (4) can be improved to $\lambda_1 + \lambda > 0$.

Based on above discussion, we know $\lambda_{H,\min} \geq \lambda + \lambda_1 > 0$ and $\lambda_{H,\max} \leq \lambda + \lambda_n$.

4. Experiments

In this section, we examine the performance of our algorithm. First of all, we compare the performance of our method with other state-of-the-art second order methods. After that, we focus on generalization ability of the solution that trust region method returns. For studying time vs function value and accuracy, we need to constrain the computing resources to 1 Xeon E5-2620 CPU, 512G memory and 1 Titan Xp GPU. To examine the large batch generalization ability, we do not put such constraint.

We include following algorithms in our experiments:

- **SGD**: We tune the step size to get the fastest convergence.
- **K-FAC** (Martens & Grosse, 2015): This method makes use of curvature information by approximating the Fisher information. We use the implementation in Tensorflow: `tensorflow/contrib/kfac`.
- **Cubic**: Traditionally we need exact Hessian and gradient information, recently the stochastic version of cubic regularized method is theoretically justified in (Tripuraneni et al., 2017).
- **TR**: Trust region method with our proposed subproblem solver. For each subproblem, we choose $N_1 = N_2 = 1$ (defined in Algorithm 2). The trust region radius is set to 1 and the step size of subproblem solver is set to 0.1. Other implementation details are postponed to appendix.
- **Hybrid**: Our proposed hybrid method, it runs trust region method until the accuracy no longer increases, and then uses the last epoch to initialize SGD method. When the training accuracy stabilizes, it returns the final model. Both TR and SGD here follow the same hyper-parameters above.

The detailed setting and more experiments on the trust region subproblem can be found in appendix.

4.1. Comparing second order methods on mini-batch

In our first experiment, we compare several recently proposed second order methods on convolutional neural network, this includes K-FAC, Cubic, and TR. A simple 5-layer convolutional neural network (see appendix for more information) with 5×5 filters is adopted as image classifier on CIFAR10 dataset. To evaluate the performance, we compare the training accuracy with respect to running time/epoch to show the relative convergence rate. The results are shown in Fig. 1, we find that although our TR method is slower than K-FAC with respect to epochs, our cost per iteration is significantly lower due to the fast subproblem solver.

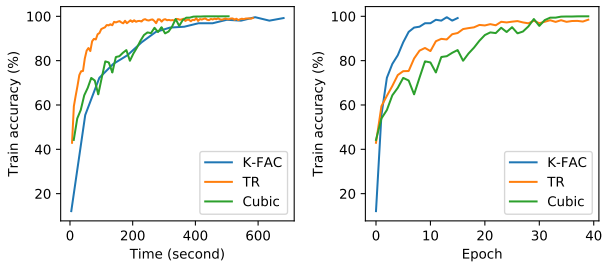


Figure 1. Training accuracy and running time/epoch comparison between TR method and K-FAC on 5-layer convolutional neural network. Both of them eventually reach 100% accuracy but TR method is faster at running time.

4.2. Comparing with SGD on large batch

Next, we test the performance of our proposed trust region method (the stochastic version is used, denoted as TR) on training deep CNNs, including VGG (Simonyan & Zisserman, 2014) and MobileNets (Howard et al., 2017). VGG/MobileNets are adopted to classify CIFAR10/STL10 data. The main focus in this experiment is to investigate the quality of solution obtained by TR and compare it with the solution obtained by SGD. We did not include other SGD variants since it is known that SGD generalize better than adaptive methods in large-batch (Wilson et al., 2017). For reference, we also list the best accuracy obtained by Adam with small batch size in Table 1, which shows Adam's accuracy is only slightly better than SGD for small batch training but still worse than our hybrid method.

In the first experiment, we compare the training/testing accuracy w.r.t. running time and epoch on VGG16+CIFAR10/STL10 dataset. For SGD we choose step size $\eta = 0.1$, which is the best step size in $\{10^n \mid n = 0, \pm 1, \pm 2, \dots\}$. For trust region subproblem we choose step size $\eta' = 0.1$ for all the experiments. Both of them use batch size $B = 512$ (for CIFAR10) and $B = 1024$ (for STL10). The experimental results are shown in Fig. 2. We observe that although our TR method takes slightly more epochs to converge, the final model has significantly lower test error.

4.3. Better generalization ability in large batch training

Training neural network with larger batch size has become an important issue for faster training. To fully exploit the computational power of multi-GPU systems, we need to increase the batch size, at least proportional to number of cores. However, large batch size can affect the generalization ability (Keskar et al., 2016), so it would be interesting to see if this dilemma can be solved merely by choosing a different optimization method.

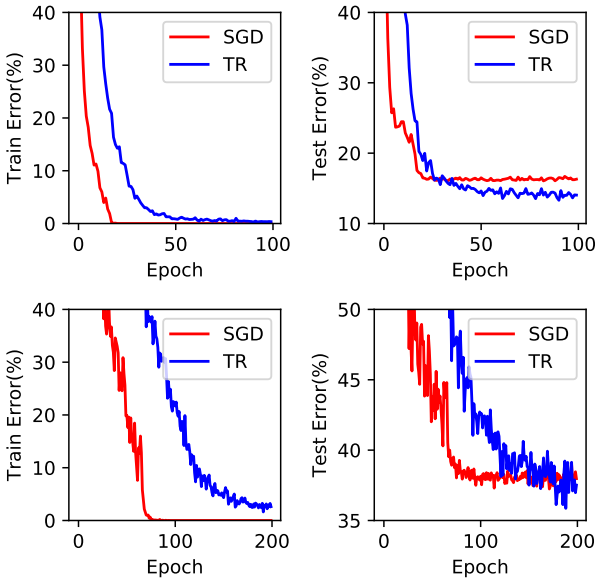


Figure 2. Training/Testing error with epoch. The first row is CIFAR10 dataset(batch size $B = 512$) and the second row is STL10 dataset(batch size $B = 1024$). We find TR method has better test accuracy on both datasets.

To do so, we design an experiment that compares testing accuracy under different batch sizes in Fig. 3. We observe that our method (TR) converges to solutions with much better test error when batch size is larger than 128. We postulate this is because SGD has not enough noise to escape from a sharp local minimum, especially when the large batch is used. For smaller batch size, TR method is not as good as SGD method, this is because the approximation of gradient and Hessian is not good enough by only a few samples. However, our Hybrid method combines the benefits of TR and SGD methods to make it both fast and accurate, for both small and large batch size.

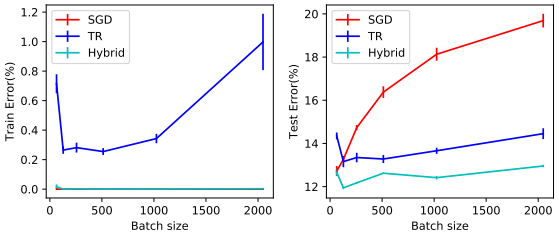


Figure 3. We choose $B \in \{64, 128, 256, 512, 1024, 2048\}$, for each batch size and each algorithm, we independently run 5 times to gather mean accuracy and deviation (in order to show significance). The experiment is conducted on VGG16+CIFAR10.

4.4. Loss landscape of different solutions

To explore whether TR/Hybrid methods find the wide local minimum, we follow the experiment techniques in (Keskar et al., 2016) to draw the loss curve that connects two models. Specifically, if we use $w^\star_{\text{SGD}}$, $w^\star_{\text{TR}}$ and $w^\star_{\text{Hybrid}}$ to denote models of SGD, trust region and hybrid method respectively, then the interpolated model in between is

$$w^\star = \alpha w^\star_A + (1-\alpha) w^\star_B, \quad A, B \in \{\text{SGD}, \text{TR}, \text{Hybrid}\}. \tag{10}$$

The loss and accuracy can be evaluated on both training set and test set by parameters $w^\star$, and the result is displayed in Fig. 4. We also calculate the ℓ_2-distance between $w^\star_A$ and $w^\star_B$, shown in titles of sub-figures.

Fig. 4 clearly shows the width and relative location of local minima: models of TR and hybrid methods lie in the same wide local minimum, while SGD finds a much sharper local minimum. Moreover, we notice Dist(TR, SGD) $\approx$ Dist(Hybrid, SGD) $\gg$ Dist(Hybrid, TR), implying that hybrid method is a refinement of TR method (both of them are close to the exact wide local minimum, but hybrid method is even closer). In contrast, SGD converges to a very different solution, where the loss curve is sharp and the model does not generalize as good as the other two.

4.5. Properties of large batch SGD vs TR

By far, we can only conclude that trust region method generalize better than SGD, especially when batch size is large. In what follows, we design a novel experiment to help us identifying the differences between those two methods. Rather than running a homogeneous algorithm, we interleave SGD and TR as two tracks, for simplicity we denote SGD(n) as running n epochs SGD, and TR(n) as running n epochs TR. In the first track, we run SGD(50) at the beginning, then we use the last iteration in SGD(50) to initialize TR(100), lastly based on the latest iterate, run SGD(50) again. In the second track we change the order: begin with TR(100), then SGD(50), and end with TR(50).

The purpose of switching between TR and SGD methods is to detect the character of respective solutions they find. Before running experiment we do such "thought experiment": **Imagine** if large batch SGD converges to a sharp local minimum, and TR method can successfully escape it, then initialized by the solution that SGD finds, TR method will "climb over" the loss hill and down to a wide minimum, during this process the loss function (as well as accuracy) first increases and then decreases. On the contrary if TR method already finds a wide local minimum, then initialized by that solution SGD shall not escape to other minima, so we don't expect a sharp change in either loss or accuracy. This thought experiment is illustrated in Fig. 5.

Our experimental results are presented in Table 1, where we run VGG16+CIFAR10 and choose batch size $B = 2048$. The overall running process is shown in the picture, and we further extract some important data into the table in order to see the differences more clearly.

We explain the results from following angles:

1. As expectation, there is a sharp raise in both train loss and test error at the 50th epoch in SGD(50)-TR(100)-SGD(50) track, meaning trust region method does escape from the solution that SGD(50) converges to. Meanwhile both loss function and test error change smoothly at the 100th epoch in TR(100)-SGD(50)-TR(50) track, which is understandable because SGD can't escape the basin of wide minimum obtained by TR.
2. Warm-started by TR(100), SGD can reach the best testing accuracy, moreover, the accuracy merely change by enlarging batch size. This sets the foundation of our hybrid method in Fig. 3, where the testing accuracy only drops 0.35% from $B = 64$ to $B = 1024$, in comparison SGD testing accuracy drops 6.97%.
3. Due to subsampled Hessian and gradient, TR method can only fluctuate around the minimizer, this causes a 1.6% drop in testing accuracy compared to the best result. Another clue is the 1.0% drop in testing accuracy and raise of training loss at 150th epoch in TR(100)-SGD(50)-TR(50) track, when SGD(50)$\rightarrow$TR(50) happens.

We want to emphasize that even though we fixed the step size in SGD while in practice we should damp the step size after every few epochs, this won't change the results above, since a smaller step size makes it even harder to escape from sharp minima. In appendix we have supplementary experiments by using other networks combined with other datasets to show that these findings are not specific to one network.

4.6. Another possibility

One may argue that our trust region method can escape from sharp local minima purely because it adopts a larger step size. We thus design an ablation experiment. Specifically we follow the standard way to run SGD: at the early stage the largest possible step size is used, and then divide the step size by a factor of 5 when the accuracy plateaus. The result is presented in appendix due to space limit, we find that SGD still doesn't generalize as good as trust region method.

5. Discussion

In this paper we first show that a simple gradient based method can effectively find the global minimum of trust region subproblem, even if it is nonconvex. Using our effi-

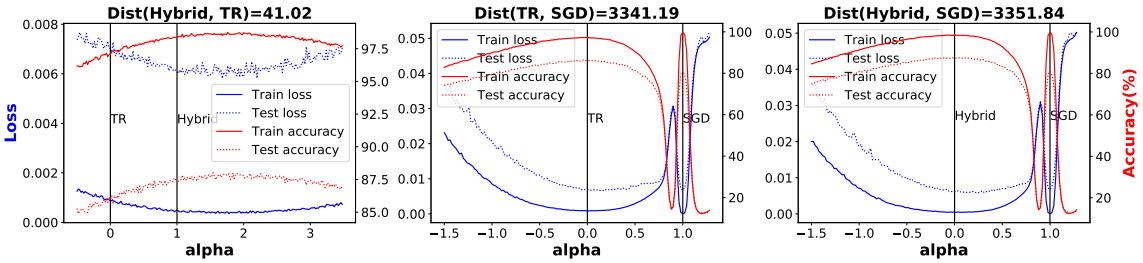


Figure 4. Loss(blue lines) and accuracy(red lines) curve of interpolated models, the x-label is α defined in (10), y-label in the left is the averaged loss and y-label in the right is accuracy. We also calculate the distance of models between $\alpha = 0$ and $\alpha = 1$ in the titles.

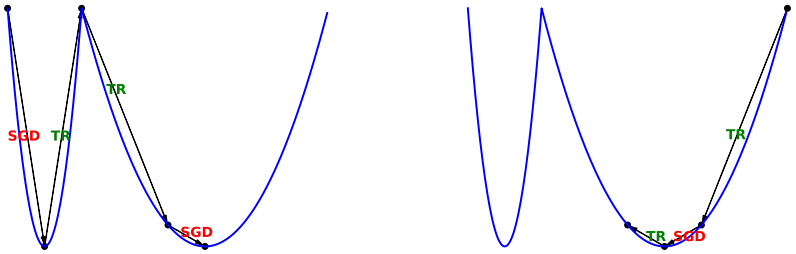
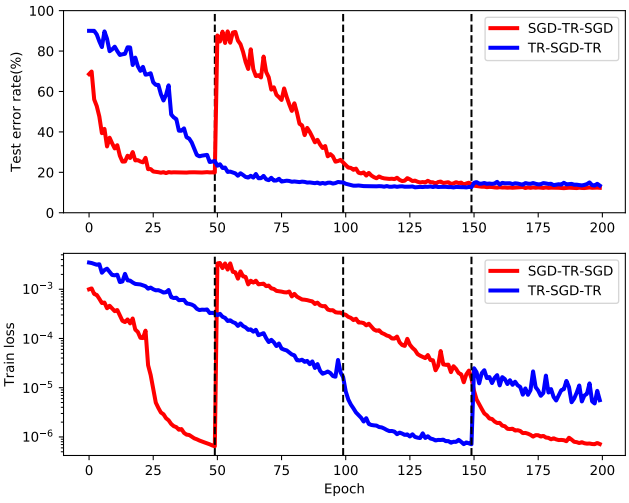


Figure 5. Illustration of **imaginary process** of Track 1(left) and Track 2(right). Note that we use subsampled Hessian and gradient, so the iterate of trust region method will fluctuate around local minimum.



	Track 1	Track 2
Stage I	SGD(50) (80.1%)	TR(100) (85.6%)
Stage II	TR(100) (86.2%)	SGD(50) (87.5%)
Stage III	SGD(50) (87.8%)	TR(50) (86.5%)
Baseline	ADAM(100) on small batch (86.6%)	

Table 1. Testing accuracy in different stages. In Track 1 we run SGD(50)→TR(100)→SGD(50), in Track 2 we change the order to TR(100)→SGD(50)→TR(50), so both tracks run 200 epochs in total. Testing accuracy is reported by the end of each stage. For both tracks we choose batch size $B = 2048$ and CIFAR-10 dataset. As a reference, we also run Adam algorithm on small batch ($B = 64$).

cient subproblem solver, subsampled trust region method is faster (in terms of running time) than state-of-the-art second order methods. Furthermore, we observe that our algorithm is comparable to SGD with respect to running time, but can converge to a solution with better generalization error. We suggest to combine trust region method with SGD to enjoy both fast and accurate properties. As an important future direction, it would be interesting to see why trust region based algorithm can escape sharp local minima and try to establish convergence results on subsampled trust region method.

References

Absil, P-A, Mahony, Robert, and Sepulchre, Rodolphe. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

Blanchet, Jose, Cartis, Coralia, Menickelly, Matt, and Scheinberg, Katya. Convergence rate analysis of a stochastic trust region method for nonconvex optimization. *arXiv preprint arXiv:1609.07428*, 2016.

Botev, Aleksandar, Ritter, Hippolyt, and Barber, David. Practical gauss-newton optimisation for deep learning. *arXiv preprint arXiv:1706.03662*, 2017.

Boyd, Stephen and Vandenberghe, Lieven. *Convex optimization*. Cambridge university press, 2004.

Carmon, Yair and Duchi, John C. Gradient descent efficiently finds the cubic-regularized non-convex newton step. *arXiv preprint arXiv:1612.00547*, 2016.

Chen, Ruobing, Menickelly, Matt, and Scheinberg, Katya. Stochastic optimization using a trust-region method and random models. *Mathematical Programming*, pp. 1–41, 2015.

Conn, Andrew R, Gould, Nicholas IM, and Toint, Philippe L. *Trust region methods*. SIAM, 2000.

Curtis, Frank E, Robinson, Daniel P, and Samadi, Mohammadreza. A trust region algorithm with a worst-case iteration complexity of\mathcal {O}(\epsilonˆ{-3/2}) for nonconvex optimization. *Mathematical Programming*, 162(1-2):1–32, 2017.

Do Carmo, Manfredo Perdigao and Flaherty Francis, J. *Riemannian geometry*, volume 115. Birkhäuser Boston, 1992.

Du, Simon S, Jin, Chi, Lee, Jason D, Jordan, Michael I, Poczos, Barnabas, and Singh, Aarti. Gradient descent can take exponential time to escape saddle points. *arXiv preprint arXiv:1705.10412*, 2017.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12 (Jul):2121–2159, 2011.

Erway, Jennifer B and Gill, Philip E. A subspace minimization method for the trust-region step. *SIAM Journal on Optimization*, 20(3):1439–1461, 2009.

Erway, Jennifer B, Gill, Philip E, and Griffin, Joshua D. Iterative methods for finding a trust-region step. *SIAM Journal on Optimization*, 20(2):1110–1131, 2009.

Ge, Rong, Huang, Furong, Jin, Chi, and Yuan, Yang. Escaping from saddle pointsonline stochastic gradient for tensor decomposition. In *Conference on Learning Theory*, pp. 797–842, 2015.

Gould, Nicholas IM, Lucidi, Stefano, Roma, Massimo, and Toint, Philippe L. Solving the trust-region subproblem using the lanczos method. *SIAM Journal on Optimization*, 9(2):504–525, 1999.

Goyal, Priya, Dollár, Piotr, Girshick, Ross, Noordhuis, Pieter, Wesolowski, Lukasz, Kyrola, Aapo, Tulloch, Andrew, Jia, Yangqing, and He, Kaiming. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Griewank, Andreas. The modification of newtons method for unconstrained optimization by bounding cubic terms. Technical report, Technical report NA/12, 1981.

Grosse, Roger and Martens, James. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pp. 573–582, 2016.

Hazan, Elad and Koren, Tomer. A linear-time algorithm for trust region problems. *Mathematical Programming*, 158 (1-2):363–381, 2016.

Hinton, Geoffrey E, Osindero, Simon, and Teh, Yee-Whye. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Ho-Nguyen, Nam and Kilinc-Karzan, Fatma. A second-order cone based approach for solving the trust-region subproblem and its variants. *SIAM Journal on Optimization*, 27(3):1485–1512, 2017.

Howard, Andrew G, Zhu, Menglong, Chen, Bo, Kalenichenko, Dmitry, Wang, Weijun, Weyand, Tobias, Andreetto, Marco, and Adam, Hartwig. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Jin, Chi, Ge, Rong, Netrapalli, Praneeth, Kakade, Sham M, and Jordan, Michael I. How to escape saddle points efficiently. *arXiv preprint arXiv:1703.00887*, 2017.

Kawaguchi, Kenji, Kaelbling, Leslie Pack, and Bengio, Yoshua. Generalization in deep learning. *arXiv preprint arXiv:1710.05468*, 2017.

Keskar, Nitish Shirish, Mudigere, Dheevatsa, Nocedal, Jorge, Smelyanskiy, Mikhail, and Tang, Ping Tak Peter. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kiros, Ryan. Training neural networks with stochastic hessian-free optimization. *arXiv preprint arXiv:1301.3641*, 2013.

Lee, Jason D., Simchowitz, Max, Jordan, Michael I., and Recht, Benjamin. Gradient descent only converges to minimizers. In Feldman, Vitaly, Rakhlin, Alexander, and Shamir, Ohad (eds.), *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pp. 1246–1257, Columbia University, New York, New York, USA, 23–26 Jun 2016. PMLR. URL http://proceedings.mlr.press/v49/lee16.html.

Martens, James. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 735–742, 2010.

Martens, James and Grosse, Roger. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417, 2015.

Nesterov, Yurii. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

Powell, Michael JD. A new algorithm for unconstrained optimization. *Nonlinear programming*, pp. 31–65, 1970.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Steihaug, Trond. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

Tripuraneni, Nilesh, Stern, Mitchell, Jin, Chi, Regier, Jeffrey, and Jordan, Michael I. Stochastic cubic regularization for fast nonconvex optimization. *arXiv preprint arXiv:1711.02838*, 2017.

Udriste, Constantin. *Convex functions and optimization methods on Riemannian manifolds*, volume 297. Springer Science & Business Media, 1994.

Wilson, Ashia C, Roelofs, Rebecca, Stern, Mitchell, Srebro, Nathan, and Recht, Benjamin. The marginal value of adaptive gradient methods in machine learning. *arXiv preprint arXiv:1705.08292*, 2017.

Wright, Stephen J and Nocedal, Jorge. Numerical optimization. *Springer Science*, 35(67-68):7, 1999.

Xu, Peng, Roosta-Khorasan, Farbod, and Mahoney, Michael W. Second-order optimization for non-convex machine learning: An empirical study. *arXiv preprint arXiv:1708.07827*, 2017a.

Xu, Peng, Roosta-Khorasani, Farbod, and Mahoney, Michael W. Newton-type methods for non-convex optimization under inexact hessian information. *arXiv preprint arXiv:1708.07164*, 2017b.

You, Yang, Gitman, Igor, and Ginsburg, Boris. Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 2017a.

You, Yang, Zhang, Zhao, Hsieh, Cho-Jui, and Demmel, James. 100-epoch imagenet training with alexnet in 24 minutes. *arXiv preprint arXiv:1709.05011*, 2017b.

Yuan, Ya-xiang. A review of trust region algorithms for optimization. In *ICIAM*, volume 99, pp. 271–282, 2000.

Zeiler, Matthew D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Zhang, Chiyuan, Bengio, Samy, Hardt, Moritz, Recht, Benjamin, and Vinyals, Oriol. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.