A survey of deep learning optimizers-first and second order methods

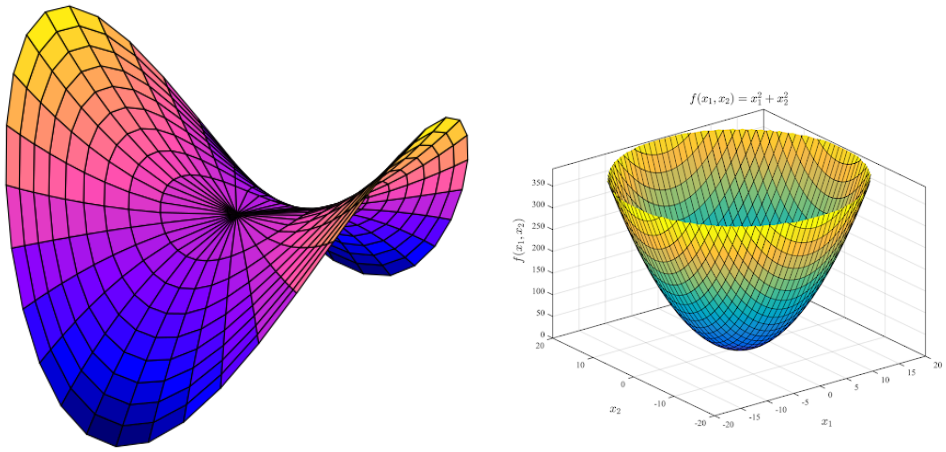
Deep Learning optimization involves minimizing a high-dimensional loss function in the weight space which is often perceived as difficult due to its inherent difficulties such as saddle points, local minima, ill-conditioning of the Hessian and limited compute resources. In this paper, we provide a comprehensive review of 12 standard optimization methods successfully used in deep learning research and a theoretical assessment of the difficulties in numerical optimization from the optimization literature.

CCS Concepts: • **Neural Networks**; • **Numerical Optimization**; • **Theory of Computation**;

Additional Key Words and Phrases: Numerical Optimization, Deep Learning

ACM Reference Format:
. 2021. A survey of deep learning optimizers-first and second order methods. *Proc. ACM Meas. Anal. Comput. Syst.* 37, 4, Article 111 (December 2021), 19 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION



Optimization encompasses a central role in machine learning, statistical physics, pure mathematics, random matrix theory and also scientific research. Deep learning involves optimization of non-convex loss functions over continuous, high-dimensional spaces. Neural network training involves solving a loss function (piecewise) that is differentiable and continuous using gradient-based optimization techniques that utilize the first order or second order information to update its weight parameters and thus seek to find the critical point (global minima.

Author's address:

2476-1249/2021/12-ART111 $15.00
https://doi.org/10.1145/1122445.1122456

Gradient descent, quasi-Newton, BFGS are commonly used to perform such minimizations and find the optimal solution (global minima). The goal of a machine learning algorithm is to reduce the expected generalization error. We do this to minimize the Kullback-Leibler (KL) divergence- a measure of how similar the two distributions are, between the empirical data distribution and the true underlying data distribution. Since,the true distribution is largely unknown we minimize the expected loss on the training set- a quantity called empirical risk and this ensures that the model generalizes to unseen examples in the test samples. Although this method, in theory is highly prone to overfitting since models with high capacity can simply memorize the training set, empirical results surprisingly enough dictate the opposite and prove the model's capability to interpolate in the convex hull of the training data points and achieve local generalization power within its capacity. Thus, high-dimensional non-convex optimization techniques comes at the cost of no theoretical guarantees yet obtains state-of-the-art results on numerous tasks such as image classification, text processing and representation learning. Also deep nets are composed of layers of affine transformations followed by a layer of nonlinearity. Thus,the choice of the nonlinear function becomes more important to avoid vanishing and exploding gradients problems.
Hochreiter and Schmidhuber[1997] conjectured that generalization of deepnets is related to the curvature of the loss at the converged solution. Their algorithm the "flat minima search" utilizes the Hessian information to find a large region in the weight space called the "flat minima" where all the solutions leads to a small error function value. It is also noted that high error local minima traps do not appear when the model is overparameterized and Keskar et al.2016 demonstrated that the large batch methods always tend to converge to sharp mimizers (large number of positive eigenvalues) and thus generalize a bit worse than the small batch methods that are known to converge to flat minimizers which are characterized by having numerous small eigenvalues though they have comparable training accuracies. This is attributed to the empirical evidence that large batch methods are attracted to regions of sharp minima while the basins found by small batch methods are wider and since are unable to escape these basins of attractions,they generalize better. Although a larger batch-size provide a more accurate approximation of the true gradient with a lot less fluctuations in the update step, it is computationally very expensive and leads to fewer update steps while a smaller batch-size offers a regularizing effect due the noise they add during the learning process. However,this might lead the high variance in the estimate of the gradient and thus requires a smaller learning rate to maintain stability and converge to an optimal solution. It turns out that most directions in the weight space with similar loss values since the Hessian consists of a large number of eigenvalues which are close to zero even at random initialization points.
Goodfellow et.al[2014] shows the existence of a linear subspace during neural network training with no barriers and a smooth path connecting initialization and the final minima. Also they show that poor conditioning of the Hessian and variance in the gradient estimate are major obstacles in the SGD training process and these could possibly deviate the algorithm completely from the desired basin of attraction. Thus the final minima or the final region in the parameter space and the width of the final minima that SGD converges to depends largely on the learning rate, batch size and gradient covariance, with different geometries and generalization properties. Felix Draxler et.al[2019] constructed continuous paths between minima for RNN's and conjectured that the loss minima are not isolated points but essentially form a continuous manifold. Also since these paths are flat this implies the existence of a single connected manifold of low loss and not a distinct basin of attraction. More precisely, the part of the parameter space when the loss remains below a certain low threshold form a connected region of low error valleys. These reflect the generalization capabilities of the network since this provides evidence that large neural nets have enough parameters to produce accurate predictions even though they undergo huge structural changes. This is quite a deviation from the current literature where the minima is typically depicted as points at the bottom of a strictly convex valley of a certain width with network parameters given by the location of the minimum. Stanislaw Jastrzębski et.al[2018] conjectured that the SGD process depends on the ratio of the learning rate to batch size which thus determines the width of the endpoint and its generalization capabilities. First, as we approach the local minima, the loss surface can be approximated by a quadratic bowl,

with the minimum at zero loss (reflecting the ability of neural nets to overfit the training data). Thus, the training can be approximated by an Ornstein-Unhlenbeck process. Second,the covariance of the gradients and the Hessian of the loss function are approximately equal and this closeness is possibly the reason behind SGD escaping the global minima. Since the loss functions of DNNs are typically non-convex functions of the parameters, with

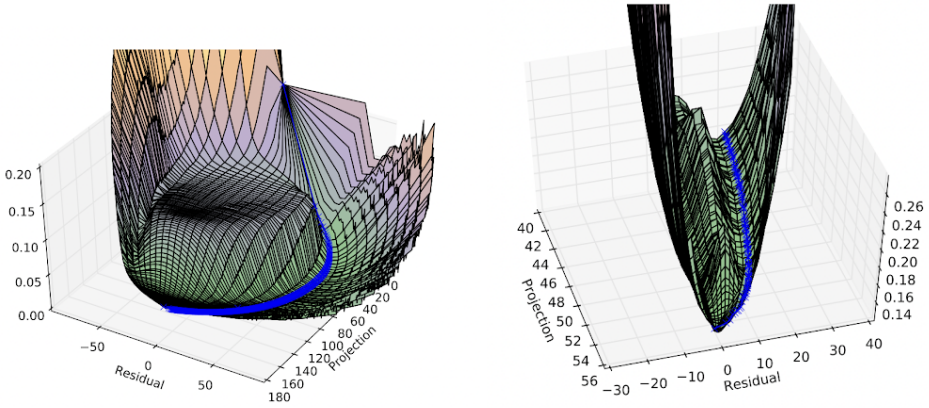


Fig. 1. The 3-D visualization of the SGD trajectory illustrating how SGD avoids the plateau regions. Source: Ian J. Goodfellow[35]

complex structure and potentially multiple minima and saddle points, SGD generally converges to different regions of parameter space, with different geometries and generalization properties, depending on optimization hyper-parameters and initialization. The reason for the success of SGD on a wide variety of tasks is now clear: these tasks are relatively easy to optimize. Finding a good direction in a high-dimensional space is a difficult problem, but it is not nearly as difficult as navigating an error surface that has complicated obstacles within multiple different low-dimensional subspaces. It is also highly likely that the non-convex loss landscape is such that it contains regions of local minima at high energies which full-batch methods couldn't have possibly avoided.But,the stochastic nature of SGD due its inherent noise and a smaller batch size doesn't suffer from this problem and is not trapped in irrelevant basins of attraction. With non-convex functions, such as neural nets, it is possible to have many local minima. Indeed, nearly any deep model is essentially guaranteed to have an extremely large number of local minima. However, this is not necessarily a major problem. It is empirically confirmed that as the dimensionality N increases, local minima with high error relative to the global minimum occur with a probability that is exponentially small in N. This is mainly because neural nets exhibit model identifiability problem which results in an extremely large amount of local minima but all are equivalent in their cost function value. They do not have very high error values relative to the global minima and are thus good solutions that stochastic gradient descent (SGD) often finds itself at during the training process.
Critical points with high cost are far more likely to be saddle points. We are not exactly concerned with finding the exact minimum of a function, but seek only to reduce its value significantly to obtain a good generalization error. In the gradient descent method,the main problem is not the direction but its step size along each eigenvector because the update step is directly proportional to the corresponding eigenvalue in that direction. Thus, we towards the optimal point if the eigenvalue is positive else we move away from it if it is negative along those directions and this slows down the learning process since now we might to circumnavigate a tall mountain or a flat region of high error or might simply take a very long time to move away from the saddle points.In a

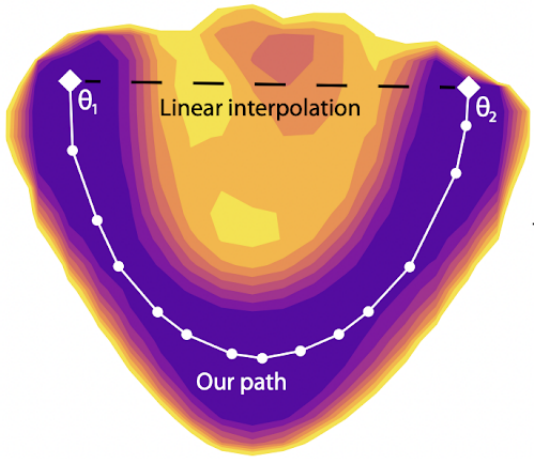


Fig. 2. Left: A slice through the one million-dimensional training loss function of DenseNet-40-12 on CIFAR10 and the minimum energy path found by our method. The plane is spanned by the two minima and the mean of the nodes of the path. Source: Felix Draxler [13]

convex problem,such a flat region corresponds to the global minima, but in non-convex optimization problem, it could correspond to a high value of the loss function.Although the Newton's method solves this slowness issue by rescaling the gradients in each direction with the inverse of Hessian,it can often result in taking the wrong direction and can be applied iteratively as long as the Hessian remains positive definite. Thus for a locally quadratic function (with positive definite H), the Newton's method jumps directly to the minimum but if the eigenvalues are not all positive (near a saddle point) then the update can lead us to move in the wrong direction. Also since second-order methods are computational intensive since the computational complexity of computing the Hessian is $O\left(N^3\right)$ and thus practically only networks with a small number of parameters can be trained via Newton's method. Several methods such as the Levenberg-Marquardt algorithm are proposed to overcome this problem in non-convex settings specifically when the Hessian has negative eigenvalues. This includes adding a constant along the diagonal the Hessian and this largely offsets the negative eigenvalues.Recently Luke Metz et.al[2021] established the connections between gradient explosion and the initialization point by measuring the recurrent Jacobian. They found that a random initialization (unstable initialization) the NN policy is poorly behaved and has many eigenvalues with norm greater than length 1 and thus the gradient norm grows. However, for a stable initialization (shrinks the initialization by multiplying by 0.01) which was picked specifically to prevent the gradients to explode, we find that many eigenvalues close to 1 and thus the gradients do not explode. Sutskever et al.[2013] demonstrated that poorly initialized and the absence of momentum perform worse than well-initialized networks with momentum.

The main challenges for neural net optimization in these high-dimensional spaces can be ill-conditioning of the Hessian, poor condition number, local minima, plateaus and flat regions, proliferation of saddle points and poor correspondence between local and global structure. The existence of a large number of saddle points can significantly slow down learning since they are regions of high error plateaus and give the impression of existence of a local minimum. Also since these critical points are surrounded by plateaus of small curvature, second-order methods such as the Newton methods are attracted to saddle points and thus slows down the learning process. Also a common observation is that the norm of the gradients (a good indication to check if the optimizer has

converged to any critical point) is not necessarily small and the Hessian has negative eigenvalues at the end of training process which is very small indicating that the algorithm did not converge to any local minima. It is also possible that models with high capacity has a large number of local minima which can increase the number of interactions and cause the Hessian to be ill-conditioned. There are also wide, flat regions of constant value where the gradients and Hessian are all zero. Such degenerate locations pose a major problem to optimization algorithms. We conjecture that large neural nets often converges to a local minima most of which are equivalent and yield similar generalization performance on the test set. Also, the probability of finding a high error local minima is exponentially small in high-dimensional structures and decreases quickly with network size.

2 FIRST ORDER METHODS

2.1 Momentum

Polyak[Polyak,1964] introduced the momentum method to accelerate the learning process of first-order gradient based methods especially in regions of high curvature and it also addresses the poor conditioning of the Hessian matrix and the variance in the stochastic gradient descent process. This algorithm computes an update step by accumulating an exponentially decaying moving average of the past gradients and continues to move in their direction. The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradient change directions and we gain a faster convergence. Although it can be applied to both full-batch and mini batch learning methods, stochastic gradient descent combined with mini-batches and a momentum term has been golden rule in the deep learning community to get excellent results on perception tasks. It dampens oscillations in directions of high curvature by combining gradients of opposite signs and also yields a larger learning rate in regions of low curvature. It builds but speed in directions of consistent gradient and thus traverse a lot faster than the steepest descent due to its accumulated velocity.
If the momentum is close to 1 then this algorithm is a lot faster and it is equally important to ensure a smaller momentum value (close to 0.5) to the start of the learning process. At the beginning, since the points are randomly initialized in the weight space,there maybe large gradients and the velocity term can lead in not entirely beneficial. Thus, once these large gradients disappears and we find the desired basin of attraction we can increase the momentum term smoothly to its final value which is 0.9 to 0.99 to ensure that learning is not stuck and wastes computational time in traversing flat regions where the gradients are almost zero.

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L \left(\boldsymbol{f} \left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} \right), \boldsymbol{y}^{(i)} \right) \right)$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$$

The momentum algorithm accelerated convergence to a local minimum, requiring fewer iterations than the steepest descent method ,precisely by $\sqrt{R}$ times fewer iterations, where R is the condition number of the curvature at the local minimum. The velocity term plays the role of velocity as in physical analogy in which it accelerates the particle through the parameter space in reinforced directions and thus the velocity vector is also known as imparting momentum to the particle. The hyperparameter determines how quickly the contributions of previous gradients exponentially decay. Previously in gradient descent the step size was proportional to the norm of the gradient. Now, it depends on how large and aligned a sequence of gradients are. The step size is largest when many gradients point in exactly the same direction. We can think of the analogy of a ball rolling down a curved hill and whenever it descends a steep part of the surface,it gathers speed and continues sliding in that direction.

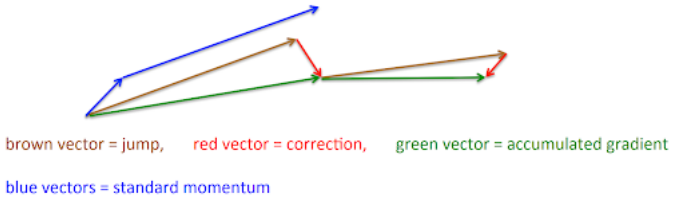


Fig. 3. Nesterov momentum first makes a big jump in the direction of the previous accumulated gradient and then compute the gradient where you end up and thus make a correction. Source: Geoffrey Hinton [29]

2.2 Nesterov Momentum

Sutskever et al.[2013] introduced a variant of the classical momentum (CM) that aligns in line with the work of Nesterov's accelerated gradient (NAG) method for optimizing convex functions. The update rule is given by:-

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L \left(f \left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v} \right), \boldsymbol{y}^{(i)} \right) \right]$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$$

From the formula it is clear that NAG is similar to CM except where the gradient is evaluated. With Nesterov momentum, the gradient is evaluated after the current velocity is applied. This means that in NAG we first make a big jump in the direction of the previous accumulated gradient and then compute the gradient where you end up and thus make a correction. This gradient based correction factor is important for a stable gradient update especially for higher values of μ. The gradient correction to the velocity responds much quicker in NAG and if μv_t is indeed a poor update and an inappropriate velocity, then NAG will point back towards t more strongly than $\nabla f(\theta_t)$ does, thus providing a larger and more timely correction to v_t than CM. Thus, it able to avoid oscillations and is much more effective than CM (effectiveness compound over the course of learning) along high-curvature vertical directions and it thus more tolerant to larger values of μ compared to CM. For convex functions, Nesterov momentum achieves a convergence rate of $O\left(1/K^2\right)$ where k is the number of steps.

2.3 Adaptive Learning Rates

The delta-bar-delta algorithm (Jacobs 1988) introduced the concept of having separate adaptive learning rates for each individual connection (weight) which is set empirically by observing its gradient after each iteration. The idea is that: if the gradient stays consistent i.e. remains the same sign we tune up the learning rate, else if the gradient reverses its sign then we decrease its learning rate. The intuition is that,for deep neural nets the appropriate learning rate varies widely for each of its weights. The magnitude of the gradients are often different for different layers and if the weight initialization is small then the gradients are small for early layers than the later ones for very deep neural nets. We start with a local gain of 1 for each weight and use small additive increases or multiplicative decreases depending on the sign of the gradient (mini-batch). This ensures that the big gains decay rapidly when the oscillations start. It is important to ensure that we limit the gains to lie within a reasonable range and use bigger mini-batches to ensure that the sign of the gradient is not due to the sampling error of the mini-batches.
Jacobs (1989) proposed the method for combining adaptive learning rates with momentum.He conjectures that instead of using the sign agreement between the current gradient and the previous gradient we use the current gradient and the velocity of the weight and thus combine the advantages of momentum and adaptive learning rates .Since momentum doesn't care about axis-aligned effects it can deal with diagonal ellipses and can traverse in the diagonal directions quickly.

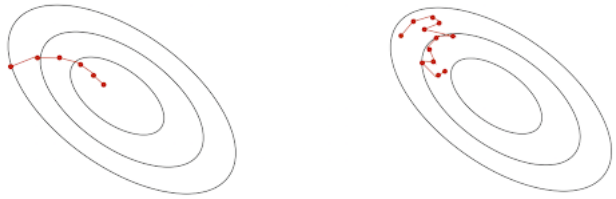


Fig. 4. Batch gradient descent moves directly downhill.SGD takes steps in a noisy direction, but moves downhill on average. Source: Roger Grosse [21]

2.4 Adagrad

The Adagrad algorithm individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of the historical squared values of the gradient (Duchi et al.[2011]). For weight parameter with the largest gradient has the largest decrease in its learning rate while the parameter with the smallest gradient has the smallest decrease in its learning rate. This results in faster progress in the more gently sloped regions of parameter space and works well when the gradient is sparse. For non-convex optimization problems the accumulation of the squared gradients can result in a premature and excessive decrease in the learning rate.

2.5 RMSProp

The RMSProp algorithm (Hinton[2012]) modifies the Adagrad by changing the accumulated squared gradient into an exponentially weighted moving average. In the nonconvex setting,since Adagrad decreases the learning rate rapidly by using the history of the squared gradient it is possible that the learning halts before arriving at the desired basin of attraction or locally convex bowl structure. RMSProp uses an exponentially decaying average of the squared gradients to discard the past history and can converge rapidly after finding a convex bowl it behaves like an instance of the AdaGrad algorithm, since the AdaGrad method is designed to converge rapidly for a convex function. If the eigenvectors of the Hessian are axis-aligned, then RMSprop can correct the curvature. Since, RMSProp lacks the bias-correction term it can often lead to large step sizes and divergence with sparse gradients. If the eigenvectors of the Hessian are axis-aligned (dubious assumption), then RMSProp can correct the curvature.

2.6 Adam

Adam (adaptive moments) aligns along the works of RMSProp and momentum with a few distinctions such as invariance to the diagonal rescaling of the gradients. First, in Adam, we incorporate momentum by computing the biased first-order moment of the gradient as an exponentially decaying moving average. Second,similar to the RMSProp algorithm we incorporate the second-order moment of the gradient as an exponentially decaying average. This combination has theoretical guarantees in convex settings.Since the moving averages are initialized to a vector of all zeros, the moment estimates are biased during zero during the initial steps especially when beta1 and beta2 are close to 1. Thus, we introduce bias-corrections to these estimates to account for their initialization for both the momentum term and the second-order moment (uncentered variance). Since, RMSProp lacks the correction factor it has high-bias in the early stages of training while Adam is robust to the choice of hyperparameters. Since the accumulated squared values of the gradients term it is an approximation to the diagonal of the Fisher information matrix it is more adaptive and leads to faster convergence.

3 SECOND-ORDER METHODS

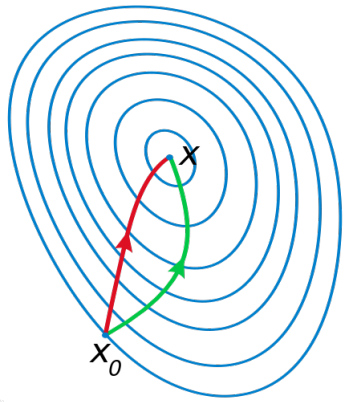


Fig. 5. Illustration of the gradient descent and Newton's method for a quadratic function. The Newton's method converges in a single step (direct route) by exploiting the curvature information contained in the Hessian matrix.

Second-order optimization methods utilize the second derivatives for weight update.Newton's method arises naturally from the second-order Taylor series approximation of the loss function, ignoring the derivatives of higher order. For a locally quadratic function (with positive definite H), by rescaling of the gradients with the inverse of the Hessian the Newton's method jumps directly to the minimum and thus converges in a single step. If the function is nearly quadratic, then this is a very good estimate of the minimizer of f. Since f is twice differentiable, the quadratic model of f will be very accurate when x is near the local minima. When it is not truly quadratic (there are higher-order terms) but can be locally approximated as a positive definite quadratic,then this update can be iterated to reach the minima which is much faster than gradient descent.Therefore,for the quadratic case the order of convergence is infinity for any initial point x_0.
Given a starting point, we construct a quadratic approximation to the objective function that matches the first and second derivative values at that point. We then minimize the approximate (quadratic function) instead of the original objective function. The minimizer of the approximate function is used as the starting point in the next step and repeat the procedure iteratively. The second derivative measures how quickly the first derivative changes w.r.t. the input and tells us how well the gradient descent will perform. The Hessian H is a measure of curvature or concavity of the function. For vector-valued functions when both the input and other are vectors then the matrix containing all the first-order derivatives is known as the Jacobian matrix. For a scalar-valued function the matrix containing the second-order derivatives is known as the Hessian matrix. Suppose $f : R^n : R$ (input is a vector and output is scalar) then the Hessian matrix of f is given as follows:-

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Convex optimization algorithms have strong theoretical guarantees because convex functions (epigraph is a convex set) are well-behaved with nice properties.They lack saddle points and all their local minima corresponds to the global minima specifically because the Hessian is positive semi-definite. The Hessian matrix is real and symmetric with $H_{ij} = H_{ji}$. At a critical point,where the gradient is zero we can examine the eigenvalues of the Hessian matrix to determine whether the critical point is a local maximum, local minimum or a saddle point. In multiple dimensions the eigendecomposition (decompose into a set of real eigenvalues and an orthogonal basis of eigenvectors) of the Hessian matrix can be used to test the critical point.Convex optimization algorithms have strong theoretical guarantees because convex functions (epigraph is a convex set) are well-behaved with nice properties. They lack saddle points and all their local minima corresponds to the global minima specifically because the Hessian is positive semi-definite.When the Hessian is positive definite (all eigenvalues are positive), the point is a local minimum. Likewise,when the Hessian is negative definite (all its eigenvalues are negative), the point is a local maximum. A saddle point has both positive and negative eigenvalues (both positive and negative curvatures), where it is a local minima across one cross section and a local maxima within another cross section (the function curves upward along one direction and downward across another cross section).

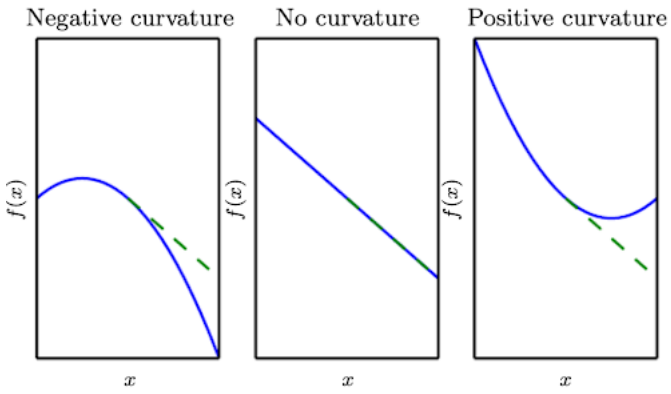


Fig. 6. A quadratic function with various curvature.The dashed line indicates the expected descent direction and the value of the cost function downhill performed by gradient descent.With negative curvature,the cost function decreases faster than the gradient predicts while with positive curvature the function decreases much slowly than expected and thus begins to increase.With no curvature the gradient correctly predicts the decrease. Source: Ian Goodfellow [23]

In the case when the energy surface is in the shape of an elongated quadratic function or an ellipsoid, then we can change the ellipsoidal shape to a spherical shape (the inverse of the Hessian spheres out the error surface locally) using a whitening transform and we can now perform normal gradient descent in the new coordinate system. In newton's optimization algorithm to find the local minimum x^* with iteration sequence of $x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots\ldots\ldots \rightarrow x$ the $\nabla f^2\,(x_k)$ must be positive definite otherwise the search direction might not be the descent direction. Since the Hessian is symmetric we can determine the Newton step (update) using Cholesky's algorithm. If the Hessian is positive semidefinite with at least one zero eigenvalue then there is nothing that confirms that the iterator has converged to a minimizer since the higher derivatives of f(x) are unknown. If the Hessian is

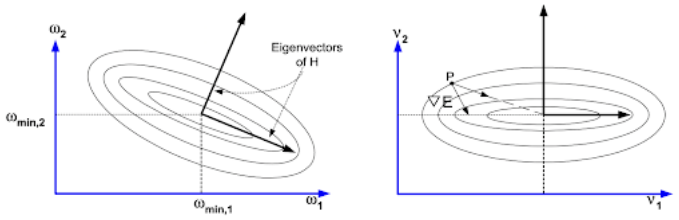


Fig. 7. For two dimensions, the lines of constant of the error surface E are oval in shape.The eigenvectors of H along the major and minor axes. The eigenvalues measure the steepness of E along each of the eigendirection. Source:Yann LeCun [1]

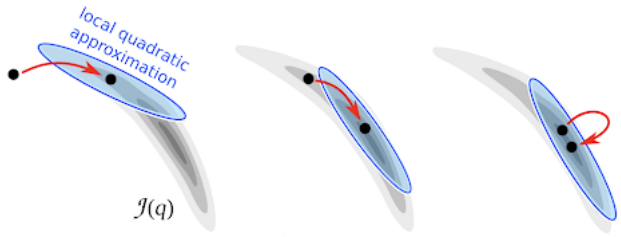


Fig. 8. Newton method constructs a local quadratic approximation of the function about the current point,moves towards the minimizer of this quadratic model and then iteratively progresses towards the global minima. Source: Nicholas Vieau Alger [17]

not positive definite (with zero eigenvalues or negative eigenvalues indicating flat regions or some directions are curved downward) then the algorithm will diverge. For nonlinear function approximators such as neural nets (affine transformation+nonlinear activation function) then the Hessian is not positive definite everywhere. Also,an important feature of Newton's method is that it is independent of linear changes of coordinates.
While minimizing a Lipschitz continuous convex function,the rate of convergence of the gradient descent method depends on the condition number of the Hessian,where a high condition number leads to slow convergence. The condition number of the Hessian being the ratio of the largest eigenvalue to the smallest eigenvalue is the ratio of the steepest ridge's steepness to the shallowest ridge's steepness. If the condition number of a set is small (near one) it means that the set has approximately same width in all directions, i.e., it is nearly spherical. If the condition number is large,it means that the set is wider in some directions than others (ellipsoidal). Thus, if the level sets were circular, then the steepest descent direction would converge to the minima is a single point whereas if they were elliptical curves, then it will not point straight to the minimum and it zigzags as it approaches the minimum. Since, SGD takes large steps in directions of high curvature and smaller steps in directions of low curvature, for a very badly conditioned problem (i.e. high condition number), the SGD can point in nearly orthogonal direction to that of the optimal direction (towards minimum) causing slow progress towards the minimum since the method must now zigzag many times to get close to the minimum. Even with an "exact line search" that minimizes along that steepest descent direction, the method ends up zigzagging. Recall that the Hessian matrix is symmetric with real eigenvalues and there is an orthogonal basis of eigenvectors.The eigendecomposition of the matrix is given as follows:-
where Q is an orthogonal matrix with the ith column corresponding to the eigenvectors and is a diagonal matrix (whose diagonal elements are eigenvalues). To understand the ill-conditioned curvature (when the Hessian is ill-conditioned gradient is changing rapidly), we illustrate it with the following example as studied in Yuchen

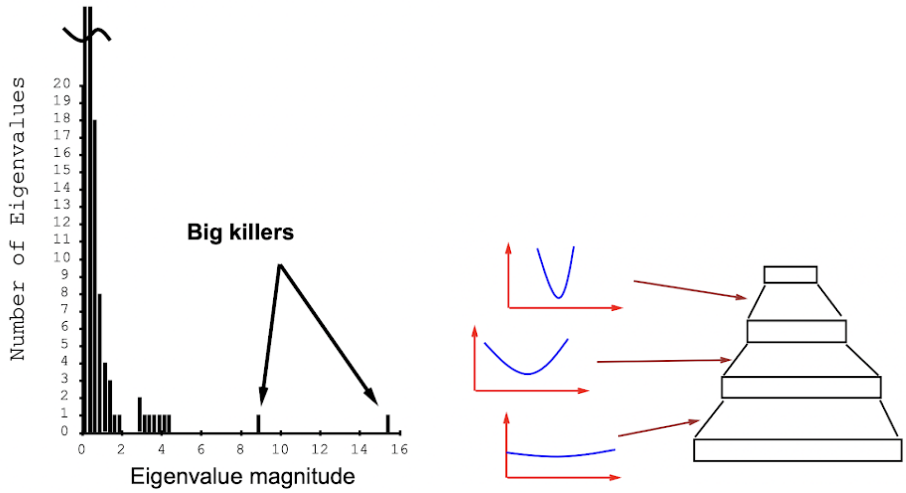


Fig. 9. Eigenvalue spectrum in a 4 layer shared weights network. Source:Yann LeCun [1]

Wang[2013]. Suppose H has some large positive eigenvalues (i.e. high-curvature directions) and some eigenvalues close to 0 (i.e. low-curvature directions), then gradient descent progresses in a rather ineffective zig-zag pattern of progress in high curvature directions and makes no significant progress in other directions with less curvature (the gradient is perpendicular to the contours and has no guarantees to preserve the minimum).
For a convex quadratic objective:-

$$\mathcal{J}(\theta) = \frac{1}{2}\theta^T A\theta$$

where A is a PSD symmetric matrix. Then we would update using gradient descent:-

$$\begin{aligned}
\theta_{k+1} &\leftarrow \theta_k - \alpha\nabla\mathcal{J}(\theta_k) \\
&= \theta_k - \alpha A\theta_k \\
&= (I - \alpha A)\theta_k \\
\implies \theta_k &= (I - \alpha A)^k\theta_0 \qquad \text{(Q orthogonal} \implies Q^TQ = I\text{)} \\
&= \left(I - \alpha Q\Lambda Q^T\right)^k\theta_0 \\
&= \left[Q(I - \alpha\Lambda)Q^T\right]^k\theta_0 \\
&= Q(I - \alpha\Lambda)^kQ^T\theta_0 \qquad \text{(by Spectral Decomposition)}
\end{aligned}$$

For a quadratic function the Hessian is constant and is well-conditioned. For non-convex functions the Hessian being ill-conditioned matters more near the minima (if the Hessian changes too fast then we follow a zigzag path). The cases:-
- $0 < \alpha\lambda_i \leq 1$: decays to 0 at a rate that depends on $\alpha\lambda_i$.
- $1 < \alpha\lambda_i \leq 2$: oscillates.
- $\alpha\lambda_i > 2$: unstable (diverges).

Hence, we need the learning rate to be bound by $\alpha < 2/\lambda_{\max}$ to prevent instability. This ensures that we avoid overshooting the minima and not going uphill in directions with strong positive curvature. This also bounds the rate of progress in other directions:-$\alpha\lambda_i < 2\lambda_i/\lambda_{\max}$.

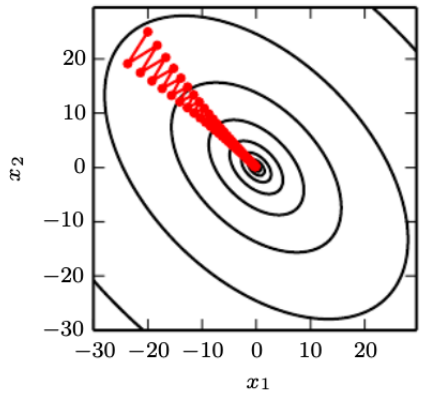


Fig. 10. The first-order gradient descent method fails to exploit the curvature information contained in the Hessian matrix.The condition number has a large influence on the SGD trajectory and we illustrate it using a quadratic function f(x) whose Hessian has a condition number 5. This means that the direction of most curvature has five times more curvature than the direction of least curvature. In this case, the most curvature is in the direction [1, 1] and the least curvature is in the direction [1, 1]. The red lines indicate the path followed by gradient descent.This results in an ellipsoidal quadratic function and gradient descent follows a zig-zag path often bouncing off the canyon walls (since this is the gradient is perpendicular to the surface) and we observe that it spends a lot of descending down the canyon walls because they are the steepest feature.It indicates that gradient descent takes larger number of steps to converge and near the minima it oscillates back and forth untils it converges to the local minima.If the step size is large then it can overshoot and reach the opposite canyon wall on the next iteration and hence not the best search direction.The large positive eigenvalue of the Hessian corresponding to the eigenvector pointed in this direction indicates that this directional derivative is rapidly increasing, so an optimization algorithm based on the Hessian could predict that the steepest direction is not actually a promising search direction in this context. Source: Ian Goodfellow [23]

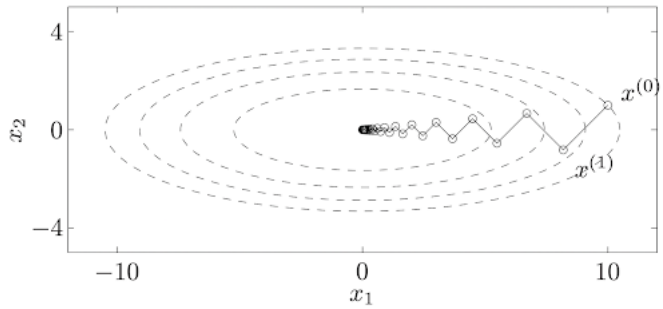


Fig. 11. Some contour lines of the function f(x)=(½)(x21+10x22). The condition number of the sublevel sets, which are ellipsoids, is exactly 10. The figure shows the iterates of the gradient method with exact line search, started at x(0)=(10,1). Source: Stephen Boyd [22]

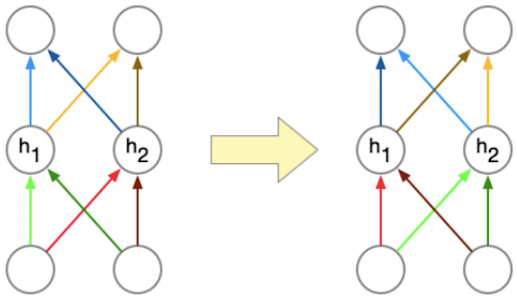


Fig. 12. Permutation symmetries i.e. reordering the hidden units in a way that preserves the function in neural nets.Thus we have k! permutations of the hidden units which all compute the same function. Source: Roger Grosse [21]

3.1 Gauss-Newton and Levenberg-Marquardt algorithm

The Gauss-Newton method utilizes the square of the Jacobian and work only for mean squared error loss functions.They have $O\left(N^3\right)$ complexity and can only minimize nonlinear least square problems.

$$\Delta w = \left(\sum_p \frac{\partial f\left(w, x_p\right)}{\partial w}^T \frac{\partial f\left(w, x_p\right)}{\partial w} \right)^{-1} \nabla E(w)$$

The Levenberg-Marquardt algorithm is an approximation to Newton's method that uses a diagonal approximation to the Hessian and takes care of extreme directions of curvature and thus prevents the update from moving in the wrong direction.

$$\Delta w = \left(\sum_p \frac{\partial f\left(w, x_p\right)}{\partial w}^T \frac{\partial f\left(w, x_p\right)}{\partial w} + \mu I \right)^{-1} \nabla E(w)$$

The regularization parameter is used to address the issue when the Hessian is not positive definite and has directions that correspond to small negative eigenvalues.

3.2 Conjugate Gradients

The conjugate gradient optimization is an iterative method that utilizes the conjugate directions i.e. orthogonal directions to perform a gradient update step and thus avoids the explicit computation of the inverse of the Hessian. This approach is used to overcome the weakness of gradient descent method and it does this by aligning the current search direction along the current gradient step with a memory component which is given by the linear combination of previous directions. For a quadratic function,the gradient descent method fails to exploit the curvature information present in the Hessian matrix wherein descends down the canyon iteratively by following a zig-zag pattern and wastes most of its time by repeatedly hitting the canyon walls and makes no significant progress. This happens because each line search direction is orthogonal to the previous direction since the minimum of the objective along a given direction corresponds to the steepest descent direction at that point. Thus, this descent direction does not preserve the progress in the previous direction and will undo the progress in that direction and we need to reiterate to reminimize the progress made in the previous direction. Thus, we obtain an ineffective zig-zag pattern of progress towards the optimum and because of the large step size it often overshoots the minima. The conjugate gradients address this problem by making the search directions A-orthogonal i.e.the current search direction is conjugate to the previous line search direction and it will not spoil the progress made in the previous iteration.

note:-This method is only suitable for solving positive definite systems i.e. when the Hessian has all positive eigenvalues.The orthogonality condition rightly finds the minimum along the direction d_i since the eigenvector e_{i+1} is orthogonal to d_i.Two directions are defined as conjugate if it satisfies the condition given as follows:-

$$\boldsymbol{d}_t^\top \boldsymbol{H}\boldsymbol{d}_{t-1} = 0$$

where H is the Hessian matrix.Thus,this method can be viewed as an example of Gram-Schmidt orthonormalization and it converges in at most k line searches.At training iteration t,the current search direction dt is given by:-

$$\boldsymbol{d}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \beta_t \boldsymbol{d}_{t-1}$$

The parameter β_t controls how much of the previous direction should contribute to the current search direction. Since,this line search method utilizes the eigenvectors of H to choose β_t which could be computationally expensive, we mainly use two popular methods for computing β_t and they are as follows:-
1.The Fletcher-Reeves:-

$$\beta_t = \frac{\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}_t\right)^\top \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}_t\right)}{\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}_{t-1}\right)^\top \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}_{t-1}\right)}$$

2.The Polak-Ribiere:-

$$\beta_t = \frac{\left(\nabla_\theta J\left(\theta_t\right) - \nabla_\theta J\left(\theta_{t-1}\right)\right)^\top \nabla_\theta J\left(\theta_t\right)}{\nabla_\theta J\left(\theta_{t-1}\right)^\top \nabla_\theta J\left(\theta_{t-1}\right)}$$

For nonlinear conjugate gradient we set this parameter to zero (restarts at each step by forgetting the past search directions with the current direction given by the unaltered gradient at that point) to ensure that it is locally optimal and ensures a faster convergence. Also, the initial point has a strong influence on the number of the steps taken for convergence and it is observed that the Fletcher-Reeves method converges only if the initial point is sufficiently close to the desired minima and thus the Polak-Ribiere method converges much more quickly.

3.3 BFGS

The Quasi-Newton Broyden-Fletcher-Goldfarb-Shanno (BFGS) method iteratively builds an approximation of the inverse of the Hessian at each step.T he BFGS incorporates the second-order information using a low-rank approximation of the Hessian and is thus similar to the method of conjugate gradients without the computational burden. A feature of this method is that the explicit calculation of the Hessian is not required since we now use the secant equation to construct the approximation matrix by successively analyzing the gradient vector. The Jacobian of g is the Hessian of f. We build the matrix M using the gradient information and the update rule is given as follows:-
1. We chose a positive definite matrix M such as the Identity matrix. We typically use a diagonal approximation such as I to ensure that successive updates of H will be positive definite.
2. Then the search direction is determined by $\rho_t = M_t g_t$.
3. A line search is performed along this direction to determine the step size ϵ^* and the final update:-

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^* \boldsymbol{\rho}_t$$

4. The update rule for the estimate of the inverse of the Hessian is:-

$$M(t) = M(t-1)\left(1 + \frac{\phi^T M \phi}{\delta^T \phi}\right)\frac{\delta\delta^T}{\delta^T\phi} - \left(\frac{\delta\phi^T M + M\phi\delta^T}{\delta^T\phi}\right)$$

where we define the abbreviations as:-

$$\phi = \nabla E(w(t)) - \nabla E(w(t-1))$$
$$\delta = w(t) - w(t-1)$$

We perform a series of line searches along this direction and since it takes more steps to converge due to the lack of precision in the approximation of the true inverse Hessian,the BFGS is a $O\left(N^2\right)$ method. This is a rank-two approximation and can be thought of a "diagonal plus low-rank" approximation to the Hessian. Since, it does not utilizes only the second-order information it is sometimes more efficient than Newton's method on difficult problems. It is estimated that BFGS has self-correcting properties i.e. when the previous estimate is bad and slows down the convergence then BFGS well correct itself in the next few steps. We mainly compute the approximation to the Hessian matrix using 1) finite difference method 2) square Jacobian approximation 3) computation of the diagonal of the Hessian 4) a product of the Hessian and a vector.

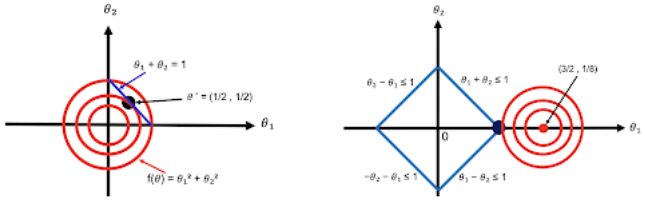


Fig. 13. Illustration of L1 and L2 regularization. Source: Kevin P. Murphy [44]

3.4 L-BFGS

Since,the memory costs of storing and manipulating the inverse of the Hessian (dense matrices) is prohibitively large for deep neural nets, we utilize the L-BFGS method to circumvent this issue. It maintains a simple and compact approximations of the Hessian matrices:we construct the Hessian approximation by using only the most recent iteration to incorporate the curvature information and surprisingly enough leads to optimal rate of convergence for difficult problems. Thus,we maintain a fixed memory system which iteratively discards the curvature information from the past since it not likely to influence the behavior of the Hessian in the current iteration, thus saving storage. Thus, we add and delete information at each stage and discards the past correction pairs and starts the process anew.Intuition:- Instead of storing the fully dense n^*n approximators, we modify M by implicitly storing only a certain number of correction pairs (say,m) and thus include the curvature information from only the m recent iteration (we store only a set of vectors of length n). The main weakness of L-BFGS is that it converges slowly on ill-conditioned problems-specifically, on problems where the Hessian has a wide distribution of eigenvalues.

3.5 ADAHESSIAN

ADAHESSIAN is a second-order stochastic approximation method that incorporates the curvature information present in the Hessian for faster convergence and reduced memory overhead using:-
1) The Hessian diagonal approximation using Hutchinson's method 2) a root-mean square exponential moving average to reduce the variations in the diagonal approximation of the Hessian 3)a block diagonal averaging to reduce the variance of Hessian diagonal elements. As discussed earlier, second-order methods involve preconditioning the gradient with the inverse of the Hessian which is designed to accelerate learning for ill-conditioned problems i.e. flat curvatures in some directions and sharp curvature in other directions by rescaling the gradient vector along each of its directions. It handles extreme directions of curvature by appropriate gradient rotation and scaling and ensures convergence to a critical point of any sort (local minima) and is thus more reliable than other adaptive first-order methods. But this comes at a prohibitive cost, since it is computationally infeasible

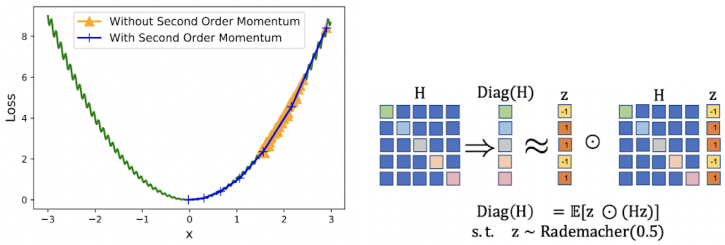


Fig. 14. Illustration of the local and global curvature wherein the local curvature information is avoided using an exponential moving average.ADAHESSIAN converges in 1000 iterations without moving average while it takes only 7 iterations with the moving average enabled. Source: Zhewei Yao [34]

to compute the Hessian at each iteration and not scalable for modern neural network architectures.Thus,we approximate the Hessian matrix as a diagonal operator given as follows:-

$$\Delta w = \text{diag}(\mathbf{H})^{-k}\mathbf{g}$$

where Δw is the weight update and $0 \leq k \leq 1$ is the Hessian power. The Hessian diagonal (diag(H)) denoted as D is efficiently computed using the Hutchinson's method. To obtain the Hessian matrix H without explicit computation, they utilize the Hessian-free approximation method (also randomized linear algebra) given as:-

$$\frac{\partial \mathbf{g}^T z}{\partial \theta} = \frac{\partial \mathbf{g}^T}{\partial \theta} z + \mathbf{g}^T \frac{\partial z}{\partial \theta} = \frac{\partial \mathbf{g}^T}{\partial \theta} z = \mathbf{H}z$$

where z is a random vector with Rademacher distribution.The Hessian diagonal is given as the expectation of $z \odot Hz$ which corresponds to optimal performance on various tasks.

$$\boldsymbol{D} = \text{diag}(\mathbf{H}) = \mathbb{E}[z \odot (\mathbf{H}z)]$$

This has the computational overhead equivalent to a gradient backpropagation step. They employ a moving-average term (spatial averaging) and Hessian momentum (update rule is similar to Adam) to smooth out stochastic variations in the curvature information per-iteration and this leads to faster convergence for smooth and strictly convex functions.

$$D^{(s)}[ib + j] = \frac{\sum_{k=1}^{b} D[ib + k]}{b}, \text{ for } 1 \leq j \leq b, 0 \leq i \leq \frac{d}{b} - 1$$

where $D^{(s)}$ is spatially averaged Hessian diagonal and b is the block size.

3.6 Newton-CG method

The line search Newton-CG method utilizes the Hessian-vector products to form the Hessian which has attractive global convergence properties with superlinear convergence and results in optimal search directions when the Hessian is indefinite. The Hessian-free method doesn't require the explicit knowledge of the Hessian but uses the finite-difference method to access the actual Hessian at each point. Here, we compute the product of the Hessian ($\nabla^2 f_k$) and the vector d using finite-differencing technique to get the approximation:-

$$\nabla^2 f_k d \approx \frac{\nabla f\,(x_k + hd) - \nabla f\,(x_k)}{h}$$

For some small differencing interval h. Also using the Gauss-Newton matrix instead of the Hessian is known to result in better search directions since G is always positive definite and thus avoids the problem of negative curvature. CG is designed for positive-definite systems and the Hessian is indefinite for points far away from the

solution,we can terminate early or use directions of negative when they are detected to account for this. Since, the HF approach has access to the true Hessian we may run across directions of extremely low curvatures or even negative curvature which and thus the search directions can result only in small reductions in the functions even after many function evaluations in the line search. If we are not careful, such a direction could result in a very large CG step, possibly taking the outside of the region where the Newton approximation is even valid. One solution is to add a damping parameter to the Hessian and consider B = H + λI. This reconditions H and controls the length of each CG step,mimicking a trust region method.

$$\begin{aligned} \text{if } \rho_k < \tfrac{1}{4}, \text{ then } \lambda \leftarrow \alpha\lambda. \\ \text{elseif } \rho_k > \tfrac{3}{4}, \text{ then } \lambda \leftarrow \alpha^{-1}\lambda. \end{aligned}$$

$$\rho_k = \frac{f\left(x_k + p_k\right) - f\left(x_k\right)}{q_k\left(p_k\right) - q_k(0)}.$$

3.7 Lookahead

Lookahead optimizer is orthogonal to the previous approaches and proposes a distinct way for weight updates in which it maintains two sets of weights:slow weights and fast weights. It maintains an inner loop of optimization for k steps using standard methods such as the SGD or Adam (mini-batches) for iterative fast weights update and follows it up with a linear interpolation of the slow weights along the direction which is given as $\theta - \phi$ in the weight space. Thus, after each slow weights update, we reset it as the new fast weights and thus results in making rapid progress in low curvature directions with reduced oscillations and faster convergence. Intuition:- The slow weights trajectory is characterized as an exponential moving average (Polyak averaging has strong theoretical guarantees) of the fast weights. After k inner-loop steps,t he slow weights is given as follows:-

$$\begin{aligned} \phi_{t+1} &= \phi_t + \alpha\left(\theta_{t,k} - \phi_t\right) \\ &= \alpha\left[\theta_{t,k} + (1-\alpha)\theta_{t-1,k} + \ldots + (1-\alpha)^{t-1}\theta_{0,k}\right] + (1-\alpha)^t\phi_0 \end{aligned}$$

For a choice of optimizer A and a mini-batch of data d the update rule for fast weights:-

$$\theta_{t,i+1} = \theta_{t,i} + A\left(L, \theta_{t,i-1}, d\right)$$

4 CONCLUSION

We summarize first-order and second-order optimization methods from a theoretical viewpoint using the optimization literature. It is also noted that high error local minima traps do not appear when the model is overparameterized and local minima with high error relative to the global minimum occur with a probability that is exponentially small in N. Critical points with high cost are far more likely to be saddle points. Also since second-order methods are computational intensive since the computational complexity of computing the Hessian is $O(N^3)$ and thus practically only networks with a small number of parameters can be trained via Newton's method. We hope that our paper will stimulate the progressive step in the examination of these methods for training deep learning architectures.

5 ACKNOWLEDGMENTS

We thank the anonymous reviewers for their review and suggestions.

REFERENCES

[1] Yann LeCun, Leon Bottou, Genevieve B. Orr, Klaus-Robert Muller "Efficient BackProp", http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

[2] Jason D. Lee, Max Simchowitz, Michael I. Jordan, and Benjamin Rech "Gradient Descent Converges to Minimizers", https://arxiv.org/pdf/1602.04915.pdf

[3] Levent Sagun , Ugur Guney, Gerard Ben Arous, Yann LeCun “Explorations on High Dimensional Landscapes”, https://arxiv.org/pdf/1412.6615.pdf
[4] Laurent Dinh, Razvan Pascanu, Samy Bengio, Yoshua Bengio “Sharp Minima Can Generalize For Deep Nets”, https://arxiv.org/pdf/1703.04933.pdf
[5] Yann N. Dauphin, Razvan Pascanu , Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli,Yoshua Bengio “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”, https://arxiv.org/pdf/1406.2572.pdf
[6] James Martens “Deep learning via Hessian-free optimization”, https://www.cs.toronto.edu/ jmartens/docs/Deep$_{H}essianFree.pdf$
[7] Razvan Pascanu , Yann N. Dauphin, Surya Ganguli, Yoshua Bengio “On the saddle point problem for non-convex optimization”, https://arxiv.org/pdf/1405.4604.pdf
[8] Stanislav Fort, Stanislaw Jastrzebski “Large Scale Structure of Neural Network Loss Landscapes”, https://arxiv.org/pdf/1906.04724.pdf
[9] Ilya Sutskever, James Martens, George Dahl, Geoffrey Hinton “On the importance of initialization and momentum in deep learning”,https://www.cs.toronto.edu/ fritz/absps/momentum.pdf
[10] Jorge J. More’ and D. C. Sorensen, Newton’s method
[11] R. Fletcher, Practical Methods of Optimization, Second
[12] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gerard Ben Arous, Yann LeCun “The Loss Surfaces of Multilayer Networks”, https://arxiv.org/pdf/1412.0233.pdf
[13] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, Fred A. Hamprecht “Essentially No Barriers in Neural Network Energy Landscape”, https://arxiv.org/pdf/1803.00885.pdf
[14] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski “Measuring the Intrinsic Dimension of Objective Landscapes”, https://arxiv.org/pdf/1804.08838.pdf
[15] Stanislaw Jastrzębski, Zachary Kenton, Devansh Arpit , Nicolas Ballas, Asja Fischer5, Yoshua Bengio, Amos Storkey “Three Factors Influencing Minima in SGD”, https://arxiv.org/pdf/1711.04623.pdf
[16] Yurii Nesterov,Introductory Lectures on Convex Optimization(2004)
[17] Nicholas Vieau Alger ‘Data-Scalable Hessian Preconditioning for Distributed Parameter PDE-Constrained Inverse Problems”, https://repositories.lib.utexas.edu/bitstream/handle/2152/75559/ALGER-DISSERTATION-2019.pdf?sequence=1
[18] Diederik P. Kingma, Jimmy Lei Ba “Adam:A Method for Stochastic Optimization”, https://arxiv.org/pdf/1412.6980.pdf
[19] Jonathan Richard Shewchuk “An Introduction to the Conjugate Gradient Method Without the Agonizing Pain”,https://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient.pdf
[20] Walter Murray “Newton-type Methods”, https://web.stanford.edu/class/cme304/docs/newton-type-methods.pdf
[21] Roger Grosse and Jimmy Ba “CSC 421/2516 Lectures 7–8: Optimization”, https://www.cs.toronto.edu/ rgrosse/courses/csc421$_{2}$019/$slides/lec$07$.pdf$
[22] Stephen Boyd, Lieven Vandenberghe “Convex Optimization”, https://web.stanford.edu/ boyd/cvxbook/bv$_{c}vxbook.pdf$
[23] Ian Goodfellow, Yoshua Bengio, Aaron Courville “Deep Learning”
[24] David G. Luenberger, Yinyu Ye “Linear and Nonlinear Programming”
[25] Jorge J. More and Danny C. Sorensen “On the Use of Directions of Negative Curvature in a Modified Newton Method”, https://www.osti.gov/servlets/purl/5331130
[26] Philip E. Gill Walter Murray “Newton-type methods for unconstrained and linearly constrained optimization”
[27] Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate O(1/k2). Soviet Mathematics Doklady
[28] Nesterov, Y. (2004). Introductory lectures on convex optimization : a basic course. Applied optimization. Kluwer Academic Publ., Boston, Dordrecht, London
[29] Geoffrey Hinton, Kevin Swersky “Neural Networks for Machine Learning”
[30] Jorge Nocedal Stephen J. Wright “Numerical Optimization”
[31] Yoshua Bengio, Aaron Courville, and Pascal Vincent “Representation Learning: A Review and New Perspectives”, https://arxiv.org/abs/1206.5538
[32] Razvan Pascanu and Tomas Mikolov and Yoshua Bengio ‘On the difficulty of training Recurrent Neural Networks”, https://arxiv.org/abs/1211.5063
[33] Y. Bengio, P. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," in IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157-166, March 1994, doi: 10.1109/72.279181
[34] Zhewei Yao and Amir Gholami and Sheng Shen and Mustafa Mustafa and Kurt Keutzer and Michael W. Mahoney “ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning”, https://arxiv.org/abs/2006.00719
[35] Ian J. Goodfellow and Oriol Vinyals and Andrew M. Saxe “Qualitatively characterizing neural network optimization problems”,https://arxiv.org/abs/1412.6544
[36] Hong Hui Tan and King Hann Lim 2019 IOP Conf. Ser.: Mater. Sci. Eng. 495 012003 “Review of second-order optimization techniques in artificial neural networks backpropagation”
[37] Yinyu Ye “Second Order Optimization Algorithms I Lecture Notes 13”

[38] Roberto Battiti; First-and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method. Neural Comput 1992; 4 (2): 141–166. doi: https://doi.org/10.1162/neco.1992.4.2.141
[39] Randall Balestriero and Jerome Pesenti and Yann LeCu "Learning in High Dimension Always Amounts to Extrapolation", https://arxiv.org/abs/2110.09485
[40] Stanislaw Jastrzebski and Maciej Szymczak and Stanislav Fort and Devansh Arpit and Jacek Tabor and Kyunghyun Cho and Krzysztof Geras "The Break-Even Point on Optimization Trajectories of Deep Neural Networks",https://arxiv.org/abs/2002.09572
[41] Stanislav Fort and Adam Scherli "The Goldilocks zone: Towards better understanding of neural network loss landscapes", https://arxiv.org/abs/1807.02581
[42] Robin M. Schmidt and Frank Schneider and Philipp Hennig "Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers", https://arxiv.org/abs/2007.01547
[43] Lin Xia, Xudong Huang, Guanpeng Wang, Tao Wu "Positive-Definite Sparse Precision Matrix Estimation", https://www.scirp.org/pdf/APM$_2$017012314391669.pdf
[44] Kevin P. Murphy , "Probabilistic Machine Learning: An Introduction".