Hess-Trucks: Leveraging Eigenspaces for Optimizing Deep Networks

Jayson P. Salkey, Greg W. Benton, Wesley J. Maddox

November 2019

1 Introduction

It has been shown empirically that the spectra of Hessian of neural networks concentrate and become heavy-tailed as they reach optima. (Pennington 2017 and Sagun 2018 and Ghorbani 2019).

However, the concentration of spectral properties and its relevance to generalization has been of interest in the Kernel community for some time. (Williams 2002 and Shawe-Taylor 2003).

Furthermore, such questions and mysteries around spectral clustering has been posed by works such as (Ng and Jordan 2002).

Parallels have been drawn between Kernel K-means and Spectral clustering (Welling Tech Note), Kernel-PCA and the Neural Tangent Kernel (Jacot 2019).

2 Hypothesis

- We have empirically witnessed and validated through some random matrix theory that some of the eigenvalues of the Hessian grow throughout training.
- We can decompose (e.g. Sagun et al, 2017 and others) $H = F + S$ where F is the Fisher information and S is another matrix.

  In the case where F is the Gauss-Newton, then elements of S roughly correspond to the residuals between model and truth.
- We observe the eigenvalues of F to increase through training explaining why the eigenvalues of H grow through training.
- In the infinite width limit, $tr(S) \to 0$ and decays through training (e.g. Jacot et al 2019).
- If we are wide enough for NTK style dynamics to hold, the posterior covariance will correspond to a null-space in $p - n$ directions.

- Shawe-Taylor et al. suggests that such a separation of eigenvalues can be induced over a random sample with kernel-PCA. The eigenvalues concentrate exponentially fast with the number of data-samples, or the right kernel.

  Perhaps such a kernel could be equivalent to the NTK or something like it.

3 Other Questions

- Does this suggest that more volume in solutions leads to easier way to widen the distance between eigenvalues?
- Connections with GPs?

  GPs do this naturally by selecting a decent kernel like an RBF which forces the decay of the eigenvalues to be fast (wide) and the log determinant of K drives the irrelevance to zero.

4 Modern Approaches: SWA, SWA-Gaussian

5 Ultra-Modern Approaches: SWA-Hess Trucks

In order to motivate our method, we must first discuss lay out some intuition about the Hessian corresponding to training a neural network.

6 Preliminaries

6.1 Assumptions

We will assume some standard technical assumptions for providing a stage for this problem, We will assume that $L(\theta)$ has Lipschitz continuous gradients

$$||\nabla L(\theta) - \nabla L(\theta')|| \leq C||\theta - \theta'||$$

We will assume that $L(\theta)$ is strongly convex near some local minimum.

$$L(\theta + h) \geq L(\theta) + \nabla L(\theta)'h + \frac{1}{2}h'\nabla^2 L(\theta)h$$

Despite using subsamples of the sampled training data, we will also assume that gradients and higher order quantities are computed without stochasticity.

6.2 Line Search

Typically in training deep networks, the learning rate of the optimizers is usually fixed to lay between, $\alpha \in (0, 1]$. This is chosen as it is usually very computationally expensive to perform a line search minimization in the direction of the computed search direction. **(MAYBE, MAYBE NOT)** However, in our experiments, to showcase the difference in learning these parameters of the network across different optimizers, network depths, and mini-batch sizes, we utilise the simple algorithm, backtracking line search, in order to enforce a sufficient decrease in the loss function by requiring that we adhere to the Armijo condition, $f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f_k' p_k$.

Algorithm 1: Backtracking Line Search

Result: α
$\alpha_0 \leftarrow 1.$;
$\rho \in (0, 1)$;
$c \in (0, 1)$;
$\alpha \leftarrow \alpha_0$;
while $L(\theta_k + \alpha p_k) > L(\theta_k) + c\alpha \nabla L(\theta_k)' p_k$ **do**
  $\alpha = \rho\alpha$;
end

For the Hessian computation, we make use of a finite differences approach where such that we can attain a Hessian vector product.

$$\nabla^2 f_k d \approx \frac{\nabla f(x_k + hd) - \nabla f(x_k)}{h},\ h \approx 10e - 8$$

6.3 Gradient Descent

We use Gradient descent as our baseline method for optimising our loss function. Gradient descent has been the standard approach to iteratively improving parameters in models for some time. We can define gradient descent as,

$$\theta_{k+1} = \theta_k - \alpha_k \nabla L(\theta_k)$$

This works due to the idea that the negated gradient direction gives the greatest reduction in the loss function, $L(\theta)$ per unit change of parameters, θ.

We can acquire some motivation from a 1-order taylor series for $L(\theta_k)$,

$$L(\theta_k + h) \approx L(\theta_k) + \nabla L(\theta_k)' h$$
$$\min_h\ L(\theta_k) + \nabla L(\theta_k)' h$$
$$\min_h\ ||h||||\nabla L(\theta_k)|| \cos(\phi),\ ||h|| = 1,\ \cos(\pi) = -1$$
$$h = -\frac{\nabla L(\theta_k)}{||\nabla L(\theta_k)||}$$

Problems with gradient descent can be seen when considering a 2-dimensional quadratic. With a fixed step length, α_k, for large step sizes, α_k, it is possible to easily overshoot the minimizer and effectively miss the optimal solution. For small step sizes, α_k, it is clear to see that as we reach the minimum, the steps taken will become continually small before reaching the minimizer.

We can show a more technical explanation of this failure by showing the gradient descent minimization to a quadratic approximation to $L(\theta)$.

$$L(\theta_k + h) \approx L(\theta_k) + \nabla L(\theta_k)'h + \frac{1}{2}h'\nabla^2 L(\theta_k)h$$
$$\approx L(\theta_k) + \nabla L(\theta_k)'h + \frac{1}{2}h'(mI)h = L(\theta_k) + \nabla L(\theta_k)'h + \frac{m}{2}||h||^2$$
$$\arg\min_h L(\theta_k) + \nabla L(\theta_k)'h + \frac{m}{2}||h||^2 = -\frac{1}{m}\nabla L(\theta)$$

Approximating $\nabla^2 L(\theta)$ with mI, where m is an arbitrary scalar, and I is an identity matrix, this gives an approximation to the Hessian that gives all directions the same very high curvature which reveals some insight into why this method struggles on problems where curvature varies greatly.

Finally, we can take a look at the convergence rate of the method to further motivate our study of higher order methods, let θ^* be a local minimizer, and a fixed step size $t \leq \frac{1}{C}$, and C was our Lipschitz constant defined above.

$$L(\theta_k) - L(\theta^*) \leq \frac{||\theta_0 - \theta^*||^2}{2tk},\ t \leq \frac{1}{C}$$

where C is the Lipschitz constant. This implies that gradient descent has a convergence rate of $\mathcal{O}(\frac{1}{k})$.

In summary, we know that advantages of gradient descent are that it is simple to implement and computing the gradient is a cheap operation at each iteration. However, it is often slow due to the fact that most problems are not strongly convex and its zig-zagging nature.

Algorithm 2: Gradient Descent with Line Search

Result: θ^*
while *not converged* **do**
  $p_k = -\nabla L(\theta_k)$;
  $\alpha_k \leftarrow lineSearch(\cdot)$;
  $\theta_k = \theta_k + \alpha_k p_k$;
end
$\theta^* = \theta_k$;

7 Quasi-Newton Approaches

7.1 Limited-Memory Broyden–Fletcher–Goldfarb–Shanno

We introduce a limited-memory variant of a Quasi-Newton method in order to potentially increase the speed of reducing the loss in terms of number of iterations.

In the original BFGS, each iteration update had the form, $x_{k+1} = x_k - \alpha_k H_k \nabla f_k$, where α_k was the step length and the inverse curvature matrix, H_k, was updated after every iteration, $H_{k+1} = V_k' H_k V_k + \rho_k s_k s_k'$. We define $\rho_k = \frac{1}{y_k' s_k}$ and $V_k = I - \rho_k y_k s_k'$, with $s_k = x_{k+1} - x_k$, $y_k = \nabla f_{k+1} - \nabla f_k$.

Clearly, the storage of the inverse Hessian and its manipulation is completely unreasonable for any practical deep neural network. In fact, any form of storage of a Hessian-like matrix is typically intractable for these problems and deters actual use of second-order optimisation methods.

The original BFGS algorithm consists of low rank (rank 2) updates to the constructed inverse approximate Hessian. By utilising LBFGS, we hope to also exploit some of this nature and attain a superlinear convergence rate in the optimisation. I expect that this method, while more computationally expensive than Gradient Descent, will perform better in terms of number of iterations.

The Limited-Memory Broyden–Fletcher–Goldfarb–Shanno(LBFGS) is a method that approximates the inverse Hessian by only considering a 'history', m, of previous differences in iterates, s_k, and gradients,y_k in order to compute just an approximate Hessian-vector product through summations and inner products alone. The time complexity of this algorithm is $\mathcal{O}(mn)$, where n is the number of variables, and m is the window size.

Algorithm 3: L-BFGS two-loop recursion

Result: $H_k \nabla L(\theta_k) = r$

$q \leftarrow \nabla L(\theta_k)$;
for $i = k-1,\ ...,\ k-m$ **do**
  $\alpha_i \leftarrow \rho_i s_i' q$;
  $q \leftarrow q - \alpha_i y_i$;
end
$r \leftarrow H_k^0 q$;
for $i = k-m,\ ...,\ k-1$ **do**
  $\beta \leftarrow \rho_i y_i' r$;
  $r \leftarrow r + s_i(\alpha_i - \beta)$;
end

Algorithm 4: L-BFGS

Result: θ^*

while *not converged* **do**
  $H_k^0 \leftarrow \frac{s'_{k-1}y_{k-1}}{y'_{k-1}y_{k-1}}I$;
  $p_k \leftarrow -H_k \nabla L(\theta_k)$;
  $\alpha_k \leftarrow lineSearch(\cdot)$;
  $\theta_k = \theta_k + \alpha_k p_k$;
  if $k > m$ **then**
    discard $s_{k-m},\ y_{k-m}$;
  end
  SAVE $s_k \leftarrow L(\theta_{k+1}) - L(\theta_k)$;
  SAVE $y_k \leftarrow \nabla L(\theta_{k+1}) - \nabla L(\theta_k)$;
end
$\theta^* = \theta_k$;

7.2 Newton Conjugate Gradient

The line search Newton conjugate gradient method is also known as the truncated Newton method. The search direction is computed by applying conjugate gradients to the Newton equations, $\nabla^2 L_k p_k = -\nabla L_k$. However, the Hessian may have negative eigenvalues, and if this is the case we terminate the CG iteration. This modification produces a guaranteed descent direction. If we replace, the Hessian with B_k, when it is positive definite, we should expect a newton-like step, and a gradient-step when there is negative curvature detected. With these conditions, this method has superlinear convergence and should perform better than Gradient Descent and similarly to LBFGS. The time complexity associated with this method is $\mathcal{O}(s\sqrt{k})$ where s is the number of non-zero entries in B_k and k is the condition number.

Algorithm 5: Line Search Newton-CG

Result: θ^*

while *not converged* **do**
  $\epsilon = \min(0.5, \sqrt{||\nabla L_k||})||\nabla L_k||$;
  $z_0 = 0,\ r_0 = \nabla L_k,\ d_0 = -r_0$;
  for *until MaxIterations* **do**
    if $d_j' B_k d_j \leq 0$ **then**
      if $j = 0$ **then**
        $p_k = -\nabla L_k$;
      else
        $p_k = z_j$;
      end
      $\alpha_j = \frac{r_j' r_j}{d_j' B_k d_j}$;
      $z_{j+1} = \alpha_j d_j$;
      $r_{j+1} = r_j + \alpha_j B_k d_j$;
      if $||r_{j+1}|| < \epsilon$ **then**
        $p_k = z_{j+1}$;
      end
      $\beta_{j+1} = \frac{r_{j+1}' r_{j+1}}{r_j' r_j}$;
      $d_{j+1} = -r_{j+1} + \beta_{j+1} d_j$;
    end
  end
  $\alpha_k \leftarrow lineSearch(\cdot)$;
  $\theta_k = \theta_k + \alpha_k p_k$;
end
$\theta^* = \theta_k$;

For the Hessian-vector product computation, we make use of a finite differences approach where such that we can attain a Hessian vector product.

$$\nabla^2 B_k d \approx \frac{\nabla f(x_k + hd) - \nabla f(x_k)}{h},\ h \approx 10e - 8$$

8 Experiments

9 Conclusions