Using Hessians as a Regularization Technique

Adel Rahimi, Tetiana Kodliuk, and Othman Benchekroun

Dathena Science Pte. Ltd., #07-02, 1 George St., Singapore, 049145
{adel.rahimi, tania.kodliuk, othman.benchekroun}@dathena.io
http://www.dathena.io

Abstract. In this paper we present a novel, yet simple, method to regularize the optimization of neural networks using second order derivatives. In the proposed method, we calculate the Hessians of the last n layers of a neural network, then re-initialize the top k percent using the absolute value. This method has shown an increase in our efficiency to reach a better loss function minimum. The results show that this method offers a significant improvement over the baseline and helps the optimizer converge faster.

1 Introduction

Neural Networks are heavily dependent on derivatives as a means of optimization given that they are differentiable end to end. For example, Gradient Descent [1] and its variants [2] [3] [4] minimize the network's loss function, $J(\theta)$, through its first-order partial derivatives. These derivatives, which are stored in the Jacobian matrix, represent the rates of the change in $J(\theta)$ with respect to each of the network's parameters. Still, gradient descent can have some difficulty optimizing these parameters only through the loss function's first-order derivatives as some of them can be far from their optimum.

Avoiding such issues can be done by computing the partial derivatives of the Jacobian matrix. These derivatives give us an understanding of the rate at which the gradient itself is changing, which is really useful when dealing with multiple parameters, especially when updating them is expensive. This gradient of gradients, i.e. $J(\theta)$'s second-order derivatives, is stored in the Hessian matrix as the gradient of the gradients. The Hessian is formulated in equation 1.

$$\mathbf{H}(f(x)) = \mathbf{J}(\nabla f(x))^T \tag{1}$$

As explained above, the Hessian matrix is based on the Jacobian's partial derivatives with respect to each network parameter. This calculation is shown in equation 2.

$$\mathbf{H}(f(x)) = \frac{\partial \mathbf{J}}{\partial x_1}, \cdots, \frac{\partial \mathbf{J}}{\partial x_n} \tag{2}$$

As can be seen in equation 2, the dimensions of the Hessian matrix can be quite large, requiring $\theta(n^2)$ memory to store it. As most of the modern neural network architectures have millions of parameters, storing all of this data can be tremendously hard, thus limiting the use of second-order derivatives when training neural networks.

2 Proposed Method

To solve the limitations introduced in the first part of this paper, we propose a simple method to find and re-initialize weights that are far from their optimal point. While some alternative methods only compute estimates as a way of overcoming the computation and storage challenges posed by Hessians, they are not accurate as they are entirely based on value approximations.

Instead, we propose calculating the Hessians solely for the last n layers in the neural network as these last layers have an immediate effect on the task at hand. Large hessian values for the n layers are the indicators of an extreme slope, and therefore of high fluctuations in the gradient. The parameters corresponding to these large values are far from their optimal weight, and subsequently will not have any impact in the network.

The training process with the proposed method is as follows:

```
for Epoch in Training:
    Forward pass
    Calculate loss
    Update weights using optimizer
    for n last layers:
        Calculate Hessian matrix
        Get absolute value of each element
        Sum/Average alongside dimensions
        Get the top Hessian magnitudes (absolute values)
        Re-initialize weights
```

Weight re-initialization can be defined by users, for instance by setting all weights to 0, by using the random uniform distribution, or by using the Xavier initialization [5]. Moreover, we can either re-initialize a whole unit — corresponding to a column of weights — or a single weight.

The experiments ran on the test data allows us to conclude that summing and averaging the hessian values does not make any difference as use absolute values; therefore, either aggregation method can be used depending on the network's architecture and the task at hand.

3 Experiment

To evaluate the proposed method, we construct a classifier on the MNIST dataset [6] with 4 hidden layers, each with a ReLU activation function. The number of hidden units per layer are respectively: 1000, 1000, 500, and 200. We then follow the algorithm presented above to get the Hessian matrix (which was averaged on the additional dimensions) and re-initialize the top k percent of the weights. To analyze the effects of k on the results, we tried different values on our data — namely $k = 10, 20, 30$.

4 Results

Experiments have shown that the proposed method significantly improves the convergence time and overall accuracy in comparison to normal training. Using a small k, $k = 10$ in our case, will bring less divergence between the normal training and the proposed method. Figure 1 shows the accuracy of our method in comparison to the baseline model, which was trained without any regularization. Our model reached a higher accuracy and has a bigger area below the curve, meaning our model reaches a higher accuracy in less epochs. Setting a larger k — with $k = 20$ for example — will divert the network as shown in figure 2 (left plot). The difference between the plots is smaller than the previous results, i.e. for $k = 10$. Finally, using a very high k ($k = 30$) will make the training process unstable, and the network will likely perform even worse, as shown in the figure 2 (right plot).

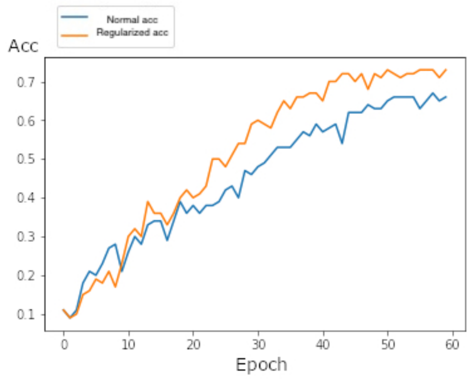

Fig. 1. Comparison of accuracy between a normal training and our proposed method with k value of 10.

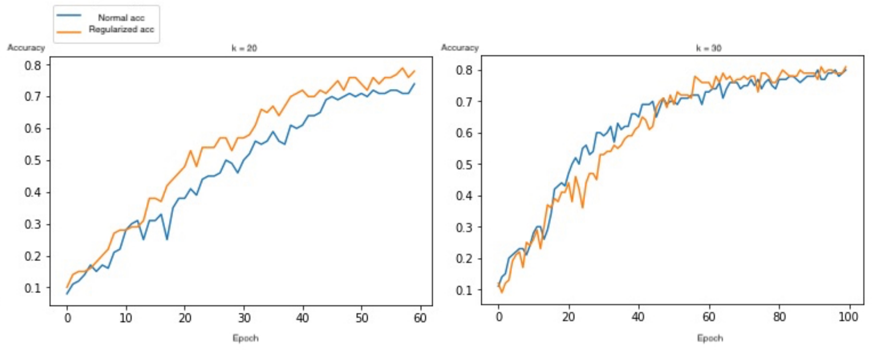

Fig. 2. Comparison of accuracy between anormal training and our proposed method with different k values (for $k = 20$ and $k = 30$).

5 Conclusion

Using the proposed method, calculating the Hessians of the last n layers of the network to re-initialize the top k percent, we can speed up the convergence of our training and tune the network's parameters that are hard to optimize. This method is more trustworthy than the alternative methods presented as it uses exact Hessians instead of simple estimations. Moreover, using only the last n layers in the network makes it less costly than conventional methods that use Hessians for the whole network. In the end, this method can be used as a simplified robust method for Neural Network regularization.

References

1. David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
2. Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
3. Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.
4. Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.
5. Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.
6. Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.