MASK AND UNDERSTAND: EVALUATING THE IMPORTANCE OF PARAMETERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Influence functions are classic techniques from robust statistics based on first-order Taylor approximations that have been widely used in the machine learning community to estimate small perturbations of datasets accurately to the model. However, existing researches concentrate on the estimate the perturbations of the training or pre-training points. In this paper, we introduce the influence functions to predict the effects of removing features or parameters. It is worth emphasizing that our method can be applied to explore the influence of any combination of parameters disturbance on the model whether they belong to the same layer or whether are related. The validation and experiments also demonstrate that the influence functions for parameters can be used in many fields such as understanding model structure, model pruning, feature importance ranking, and any other strategies of masking parameters as you can imagine when you want to evaluate the importance of a group of parameters.

1 INTRODUCTION

With the continuous development of machine learning, it has shown great success in many application scenarios such as natural language processing, biology, finance, computer vision, etc (Goodfellow et al., 2016). In the process of building a model, we often want to know whether a particular feature is important or not, and how much this parameter contributes to the effect of the entire model, especially when the number of the feature or the parameters is huge. An intuitive method is to delete a feature or set a parameter to 0, retrain the model with the training set, and compare the changes between the new model and the initial model to understand the impact of a feature or parameter on the model. However, it is unrealistic to retrain the model for each judgment, which is an unacceptable cost. Fortunately, we have a simple and powerful tool called influence functions.

Influence functions are classic techniques from robust statistics (Cook & Weisberg, 1980; 1982) based on first-order Taylor approximations that can estimate small perturbations accurately to the model. As is shown in (Koh et al., 2019), there have been a large number of useful applications such as diagnosing batch effects, apportioning credit between different data sources, understanding effects of different demographic groups or in a multiparty learning setting since the introduction of influence functions into understanding black-box predictions (Koh & Liang, 2017). However, all the existing researches are focused on predicting the effects of removing training points on the model. What would happen if we use influence functions to predict the effects of removing parameters?

In this paper, we first introduce the influence functions to estimate the parameters' perturbations. Then we extend the method from linear conditions to non-convex deep learning problems. We also use the calculation method of inversion of the Hessian matrix based on Hessian vector product (HVP) (Pearlmutter, 1994). Validation experiments show that the predicted value obtained by our method has a strong linear correlation with the true value even when we mask 60% of the parameters. Finally, we cite a few practical applications. The parameter-based influence functions can help us understand the model structure, assist in pruning strategies and rank the importance of features.

In summary, the major contributions of the paper include:

- **We give a general framework and introduce the influence functions for parameters' perturbations.** To the best of our knowledge, All current work on influence functions in the field of machine learning is based on datasets, and we creatively use influence functions

to determine the influence of parameter disturbances on the model. Our framework can not only calculate the omission of a single parameter but also calculate the influence of a set of parameters on the model. We mentioned the difficulty of derivation and the main theoretical contributions in Subsection 3.3.

- **We propose the method which can be applied to explore the influence of any combination of parameters disturbance on the model whether they belong to the same layer or whether they are related.** We give the application of constructing a group with different granularity. For validation in Subsection 5.1, we mask parameters randomly selected. For understanding model structure in Subsection 5.2, we mask the parameters of each layer as a group. For model pruning in Subsection 5.3, we prune at the granularity of the channel. For feature importance ranking in Subsection 5.4, we use whether it is related to a feature as the grouping standard of the parameter. Our method can evaluate the impact of any combination of parameters on the model at any level, without ignoring the correlation between the parameters.
- **We apply our method to different kinds of tasks and various fields.** As shown in Section 5, our method has a wide range of applications in various fields and can be used as a theoretically guaranteed parameter importance evaluation tool.

The rest of the paper is organized as follows. We introduce the related work in Section 2. In Section 3, we introduce the parameter-based influence function framework. In Section 4, we extend the previous linear hypothesis to non-convex and non-convexity and non-convergence models and introduce the acceleration method based on the Hessian vector product (HVP). In Section 5, we conducted a large number of experiments. The verification experiments and application experiments show that our method has a good effect in calculating the influence of the parameter grouped by each granularity on the model. Finally, we end in Section 6 with some discussion and conclusion.

2 Related work

Influence functions were first introduced by (Cook & Weisberg, 1980; 1982) and first used in the machine learning community for interpretability by (Koh & Liang, 2017) to estimate the effects of upweighting or perturbing a training point and the loss for a particular test point. Since then, influence functions have been widely used in various machine learning tasks. For example, (Schulam & Saria, 2019) used influence functions to approximate the gradient to recover a counterfactual distribution and increase model fairness. (Brunet et al., 2019) used influence functions to understand the origins of bias in word-embeddings. (Chen et al., 2020) identified the effects of pre-training points using influence functions. From the perspective of machine learning methodology and theory, there also have been a lot of further studies in recent years. (Koh et al., 2019) gave the accuracy of influence functions for measuring group effects with experimental and theoretical analysis. (Basu et al., 2020) showed that second-order influence functions could be used with optimization techniques to improve the selection of the most influential group for a test sample. To the best of our knowledge, all the researches on the influence functions are limited to the perturbations of the training or pre-training points. Our work is the first time to use the influence functions to study the parameter or feature disturbance.

Model pruning have been widely used for model compression in neural networks (Reed, 1993; Aghasi et al., 2017; Han et al., 2015; Sun et al., 2016). Weight-based methods, such as (Han et al., 2015), deleted parameters based on the magnitude of their absolute values, and retrain the remaining ones to recover the original prediction performance. However, as (LeCun et al., 1990; Hassibi & Stork, 1993) was shown, parameters with a low magnitude of their absolute values can be necessary for low error. Gradient-based methods, such as (LeCun et al., 1990; Hassibi & Stork, 1993; Dong et al., 2017), proposed to minimize the least increase of error approximated by second-order derivatives to prune parameters. But they did not consider the possible cross-correlations that may exist between the two parameters, at the same time, the method they used to calculate the Hessian inverse is extremely expensive. In the past few years, (Frankle & Carbin, 2018) found that a standard pruning technique naturally uncovered subnetworks whose initializations made them capable of training effectively, which worked well in experiments but lacked sufficient theoretical support. Our work can be used in model pruning as a kind of gradient-based method, which considers the

possible cross-correlations between two parameters for the first time and gives a complete theoretical guarantee.

Feature selection is a long-standing issue that is mainly classified into three methods: filter methods, wrapper methods, and embedded methods. **Filter methods** first perform feature selection on the data set, and then train the model (Battiti, 1994; Song et al., 2007; Chen et al., 2017). The feature selection process has nothing to do with the subsequent learning process. **Wrapper methods** is a kind of greedy search that directly takes the final model performance to be used as the evaluation criterion for features (Verikas & Bacauskiene, 2002; Kabir et al., 2010; Roy et al., 2015). Therefore, it is better than filtered selection generally, but the model needs to be trained multiple times in the feature selection process, the cost is usually much larger than the filtered selection. **Embedded methods** is the integration of the feature selection process and the learner training process, both of which are completed in the same process. The Least Absolute Shrinkage and Selection Operator (LASSO) (Tibshirani, 1996) is one of the most famous embedded methods, which is aimed to minimize the loss while enforcing an ℓ_1 constraint on the weights of the features. Although there have been some researches (Lemhadri et al., 2021a;b) of Lasso into neural networks in recent years. This may not always work for complex neural networks. Therefore, in the context of neural networks, some gradient-based methods have been proposed, eg., the averaged input gradient (AvGrad) (Hechtlinger, 2016) that uses the average of all the saliency maps extracted from individual instances for feature selection and (Ribeiro et al., 2016; Škrlj et al., 2020) use different aggregation mechanisms to judge the importance of features. (Sundararajan et al., 2017) use integrated gradients to solve the problem of gradients tend to be 0 in deep learning. Most of the existing gradient-based methods simply consider the derivative of the loss w.r.t. the datasets x and attach importance to experiments. But they do not give theoretically the corresponding relationship between gradient and disturbance, which is exactly one of our contributions.

3 Parameter-based Influence Functions

In this section, we detailedly introduce the general form of the application of the influence function for the parameters and give the changes of the model and the evaluation function when the parameters have a small disturbance.

3.1 Background and preliminary

Consider a prediction problem with parameters $\boldsymbol{\theta} \in \Theta$ from an input space $\mathcal{X} \subset \mathbb{R}^d$ to an output space $\mathcal{Y} \subset \mathbb{R}$. Given a training set of n instances $\{(\mathbf{x_i}, y_i)\}_{i=1}^n$ and a loss function $\ell(\mathbf{x}, y, \boldsymbol{\theta})$ which is twice-differentiable and convex in $\boldsymbol{\theta}$. It's worth to note that we will give the extensions to relax these assumptions in Section 4. To train the model, we need to select the model parameters which can minimize the L_2-regularized empirical risk

$$\hat{\boldsymbol{\theta}}_{\mathbf{1}} = \underset{\boldsymbol{\theta}_{\mathbf{1}} \in \Theta}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{x_i}, y_i, \boldsymbol{\theta}_{\mathbf{1}}) + \frac{\lambda}{2} ||\boldsymbol{\theta}_{\mathbf{1}}||_2^2, \tag{1}$$

where $\lambda > 0$ is the parameter which controls the regularization strength.

We want to measure how the other parameters would change if a particular group of parameters is masked. In order to achieve our goal, we define a perturbed vector $\boldsymbol{w} \in \{0, 1\}^m$, where m is the dimension of $\boldsymbol{\theta}$. In this situation $\boldsymbol{\theta}_w = \boldsymbol{\theta}_{\mathbf{1}} - \boldsymbol{w} \odot \boldsymbol{\theta}_{\mathbf{1}}$ represents the value of parameters masked by $\boldsymbol{w}$, where $\odot$ is the Hadamard product (also known as the element-wise, entrywise or Schur product), $\boldsymbol{\theta}_{\mathbf{1}}$ means that all of the parameters are not masked. The number of masked parameters is $||\boldsymbol{w}||_1$. The a new potimal model parameters $\boldsymbol{\theta}_w$ can be given by solving the following optimization problem when $||\boldsymbol{w}||_1$ number of them is masked

$$\hat{\boldsymbol{\theta}}_w = \underset{\boldsymbol{\theta}_w \in \Theta}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{x_i}, y_i, \boldsymbol{\theta}_w) + \frac{\lambda}{2} ||\boldsymbol{\theta}_w||_2^2. \tag{2}$$

We can calculate the influence of the change $\hat{\boldsymbol{\theta}}_w - \hat{\boldsymbol{\theta}}_{\mathbf{1}}$ by retraining the model. However, retraining the model for each removed parameter is extremely slow. In addition, if we calculate the changes after removing one parameter separately, and use their sum as the change of removing multiple

parameters, this method completely ignores the relationship between the parameters. There may be a set of related parameters whose influence on the model as a whole is less than the sum of the influence of a single parameter on the model. To solve these problems, we introduce the popular influence functions to calculate the influence of parameter changes on the model.

3.2 PARAMETER-BASED INFLUENCE FUNCTIONS

Theorem 1. *Let $H_\lambda = \frac{1}{n}\sum_{i=1}^{n} \nabla^2_{\boldsymbol{\theta}} \ell(\mathbf{x_i}, y_i, \hat{\boldsymbol{\theta}}_\mathbf{1}) + \lambda I$ is the second derivative of the L_2-regularized empirical risk (defined in Equation 1), and assume it exists. $\boldsymbol{w} \in \{0,1\}^m$ where m is the dimension of $\boldsymbol{\theta}$. Then, we have*

$$\mathcal{I}_{\text{params}} \stackrel{\text{def}}{=} \left.\frac{d\hat{\boldsymbol{\theta}}_w}{d\boldsymbol{w}}\right|_{\boldsymbol{w}=\mathbf{0}} = -H_\lambda^{-1}[\boldsymbol{w} \odot \hat{\boldsymbol{\theta}}_\mathbf{1} \odot (H_\lambda^{-1}\boldsymbol{w})^{\odot -1}], \tag{3}$$

where the operator '$\odot -1$' is the Hadamard inverse. This means that $B = A^{\odot -1}$ is equal to $B_{ij} = A_{ij}^{-1}$. $\hat{\boldsymbol{\theta}}_\mathbf{1}$ is calculated in Equation 1.

Remark 1. *Equation 3 gives the effects of the disturbance on the parameter to the model. H_λ describes the local curvature of the function, $(H_\lambda^{-1}\boldsymbol{w})^{\odot -1}$ implies the possible hidden cross-correlations between multiple parameters and $\boldsymbol{w} \odot \hat{\boldsymbol{\theta}}_\mathbf{1}$ is the original value of the masked parameters, which is obvious since the masked parameters are removed and the values of changes are equal to the original ones. In particular, when we only remove one parameter, the form of the influencing functions for parameters will become $\mathcal{I}_{\text{params},p} = -\frac{\theta_p}{[H_\lambda^{-1}]_{pp}} H_\lambda^{-1}$, where θ_p is the value of the removed parameter and $[H_\lambda^{-1}]_{pp}$ is the inverse of the Hessian matrix at the corresponding position. We can easily measure the sensitivity of the parameters through this formula.*

Next, we can analyze the influence of parameters that have a small disturbance on loss.

Lemma 1. *Under the same assumptions as Theorem 1, we have*

$$\begin{aligned}\mathcal{L}_{\text{Sensitivity}} &\stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n}\left(\left(\ell(\mathbf{x_i}, y_i, \hat{\boldsymbol{\theta}}_w) + \frac{\lambda}{2}||\hat{\boldsymbol{\theta}}_w||_2^2\right) - \left(\ell(\mathbf{x_i}, y_i, \hat{\boldsymbol{\theta}}_\mathbf{1}) + \frac{\lambda}{2}||\hat{\boldsymbol{\theta}}_\mathbf{1}||_2^2\right)\right) \\ &= \frac{1}{2}([\boldsymbol{w} \odot \hat{\boldsymbol{\theta}}_\mathbf{1} \odot (H_\lambda^{-1}\boldsymbol{w})^{\odot -1}])^{\mathrm{T}} H_\lambda^{-1}[\boldsymbol{w} \odot \hat{\boldsymbol{\theta}}_\mathbf{1} \odot (H_\lambda^{-1}\boldsymbol{w})^{\odot -1}],\end{aligned} \tag{4}$$

where $\hat{\boldsymbol{\theta}}_\mathbf{1}$ is calculated in Equation 1.

Remark 2. *Equation 4 gives the effect of small disturbance of parameters on loss. Similarly, when we only remove one parameter, the form of the influencing functions for loss will become $\mathcal{L}_{\text{Sensitivity,p}} = \frac{1}{2}\frac{(\theta_p)^2}{[H_\lambda^{-1}]_{pp}}$, where θ_p is the value of the removed parameter and $[H_\lambda^{-1}]_{pp}$ is the inverse of the Hessian matrix at the corresponding position. We can measure the sensitivity of the loss with different parameter masks through this formula instead of retraining the model.*

In practical applications, we often use a function $f_{\text{eva}} : \Theta \to \mathbb{R}$ to evaluate the model, or we want to know the influence of parameters on test datasets. In this situation, the change of f_{eva} can be used to judge the influence of the parameters disturbance on the model. We apply the chain rule of the derivation to give the influence functions of f_{eva}

$$\begin{aligned}\mathcal{I}_{f,\mathrm{z}} &\stackrel{\text{def}}{=} \left.\frac{df_{\text{eva,z}}(\hat{\boldsymbol{\theta}}_w)}{d\boldsymbol{w}}\right|_{\boldsymbol{w}=\mathbf{0}} \\ &= \nabla_{\boldsymbol{\theta}} f_{\text{eva,z}}(\hat{\boldsymbol{\theta}}_\mathbf{1})^{\mathrm{T}} \left[\left.\frac{d\hat{\boldsymbol{\theta}}_w}{d\boldsymbol{w}}\right|_{\boldsymbol{w}=\mathbf{0}}\right] \\ &= -\nabla_{\boldsymbol{\theta}} f_{\text{eva,z}}(\hat{\boldsymbol{\theta}}_\mathbf{1})^{\mathrm{T}} H_\lambda^{-1}[\hat{\boldsymbol{\theta}}_\mathbf{1} \odot (H_\lambda^{-1}\boldsymbol{w})^{\odot -1} \odot \boldsymbol{w}].\end{aligned} \tag{5}$$

Generally, the form of the evaluation function is the same as the loss function. We can not only measure the change in the training loss but also in the test loss when point z is in test datasets. Through the method of influence function, we can intuitively understand the influence of parameter disturbance on the model, which is widely used in many fields.

3.3 DIFFICULTIES AND OUR CONTRIBUTION

It is worth emphasizing that the disturbance of parameters is much more complex than that of datasets. Theoretically, the parameter-based influence functions need Hadamard product and corresponding inverse operation, which is different from our common matrix multiplication, which brings the difficulty of the proof. In terms of practical calculation, the influence functions based on the dataset often need to calculate HVP once, while the parameter-based influence functions often need to be calculated several times, which depends on how many combinations of parameters we need to evaluate.

Besides, our method can be applied to explore the influence of any combination of parameters disturbance on the model. In other words, you can imagine as you can, masking the parameter j of the i-th layer of the neural network and the parameter n of the m-th layer, regardless of whether they belong to the same layer or whether they are related. You can get the sensitivity value of the model with any imaginative mask functions.

4 EXTENSIONS AND TECHNOLOGIES

In this section, we introduce some extensions to our method so that it can be used in deep learning. Besides this, we will introduce the acceleration method to calculate the results efficiently.

4.1 NON-CONVEXITY AND NON-CONVERGENCE

In Section 3, we assume that $\hat{\boldsymbol{\theta}}$ is the global minimum. In practice, we often use SGD to optimize parameters or the model we build is totally non-convex. In this situation, we get our parameters $\tilde{\boldsymbol{\theta}}$, where $\tilde{\boldsymbol{\theta}} \neq \hat{\boldsymbol{\theta}}$. $H_{\tilde{\boldsymbol{\theta}}}$ could have negative eigenvalues. We can form a convex quadratic approximation of the loss around $\tilde{\boldsymbol{\theta}}$, i.e. $\tilde{\ell}(\boldsymbol{\theta}) = \ell(\tilde{\boldsymbol{\theta}}) + \nabla_\theta \ell(\tilde{\boldsymbol{\theta}})^{\mathrm{T}}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}) + \frac{1}{2}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^{\mathrm{T}}(H_{\tilde{\boldsymbol{\theta}}} + \lambda \mathbf{I})(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})$. Here ψ is a damping term added to insure that $H_{\tilde{\boldsymbol{\theta}}} + \lambda \mathbf{I}$ do not have negative eigenvalues. Recall that $H_{\tilde{\boldsymbol{\theta}},\lambda} = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta^2 \ell(\mathbf{x_i}, y_i, \tilde{\boldsymbol{\theta}}) + \frac{\lambda}{2}||\tilde{\boldsymbol{\theta}}||_2^2$, we have $H_{\tilde{\boldsymbol{\theta}},\lambda} = H_{\tilde{\boldsymbol{\theta}}} + \lambda \mathbf{I}$. If we choose the appropriate value of λ, then we can guarantee that $H_{\tilde{\boldsymbol{\theta}},\lambda}$ is positive definite.

4.2 HESSIAN VECTOR PRODUCT

According to Section 3, it is difficult for us to calculate the H_λ and inverse it. With n training points and p-dimensions of $\boldsymbol{\theta}$, it costs $O(np^2)$ operations to calculate the Hessian matrices and $O(p^3)$ operations to calculate the inverse, which is impossible for us in large scale machine learning applications. We investigated the common methods of calculating the inverse of the Hessian matrix (Lorraine et al., 2020; Maclaurin et al., 2015; Larsen et al., 1996; Bengio, 2000). Luckliy we can efficiently approximate the inverse with the Neumann series:

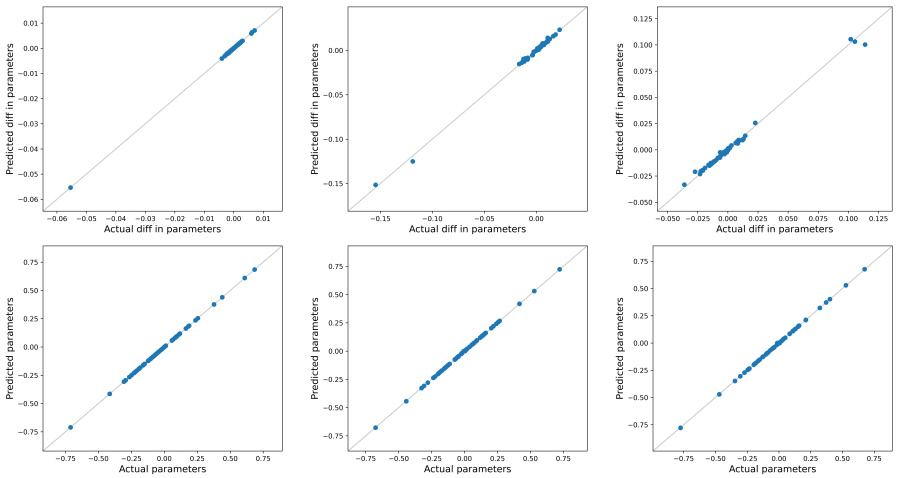
$$H_\lambda^{-1} = \lim_{i=\infty} \sum_{j=0}^{i} [I - H_\lambda]^j. \tag{6}$$

We then show how to do this approximation without instantiating any matrices by using efficient Hessian vector products. In particular, we can uniformly sample $(\mathbf{x}_i, y_i)$ and use $\nabla_{\tilde{\theta}}^2 \ell(\mathbf{x_i}, y_i, \boldsymbol{\theta})$ as an unbiased estimator of H. This gives us the following procedure: uniformly sample t points from the training data; define $\tilde{H}_0^{-1} v = v$; and recursively compute $\tilde{H}_j^{-1} v = v + (I - \nabla_{\tilde{\theta}}^2 \ell(\mathbf{x}_i, y_i, \boldsymbol{\theta}))\tilde{H}_{j-1}^{-1} v$, taking $\tilde{H}_t^{-1} v$ as our final unbiased estimate of $H^{-1} v$ when t is large enough and $\tilde{H}_t$ is stable.

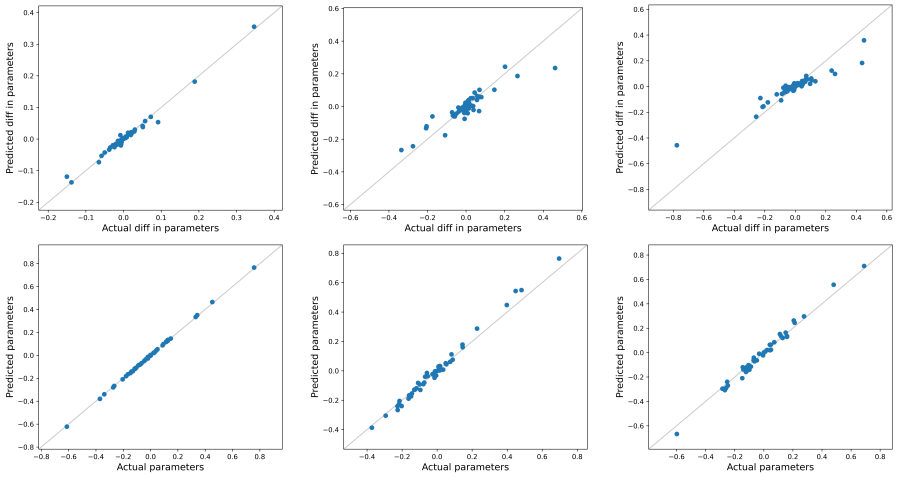
Compared with directly calculating the inverse of the Hessian matrix, using the Hessian vector product (Pearlmutter, 1994) can reduce the time complexity from $O(np^2 + p^3)$ to $O(np)$. At the same time, we do not need to store the matrix, the space complexity is also reduced from $O(p^2)$ to $O(p)$.

5 VALIDATION AND APPLICATIONS

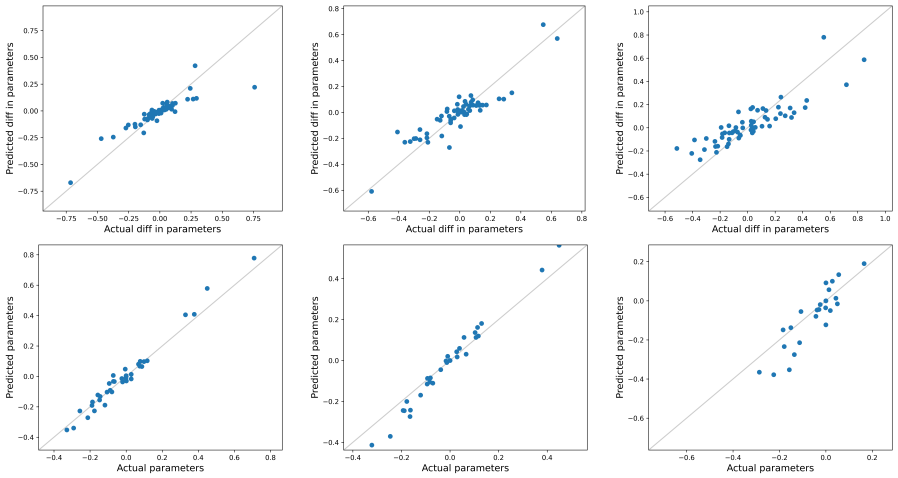
In this section, we will do some experiments in a single machine with eight cores (Intel Xeon Silver 4210@2.20 GHz), 128 GB of memory, and GeForce RTX 3090 with 24 GB graphics memory.



(a) mask one parameter (b) mask 5% parameters (c) mask 10% parameters



(d) mask 15% parameters (e) mask 20% parameters (f) mask 30% parameters



(g) mask 40% parameters (h) mask 50% parameter (i) mask 60% parameter

Figure 1: **Influence functions vs. retraining**. a, b, c, d, e, f, g, h, i represent that we randomly mask single, 5%, 10%, 15%, 20%, 30%, 40%, 50%, 60% of parameters.

To investigate the accuracy of our method, we firstly verify the accuracy of the approximation by comparing our influence functions with the retraining model. Then we give the applications of constructing a group with different granularity. These applications show that our method is effective and versatile.

5.1 INFLUENCE FUNCTIONS VS. RETRAINING

Influence functions in Section 3 give an approximate method for calculating the change of the model after masking a group of parameters. To prove the effectiveness of our method, we use a logistic regression model on the libsvm[1] dataset 'splice-scale' to compare the approximate results obtained by our algorithm with the results of model retraining.

To investigate the accuracy of our method, we compare **Actual parameters** which is calculated by retraining with **Predicted parameters** which is predicted using our method. To more accurately represent the role of our approximation algorithm, we calculate the difference between the parameters of the initial model and the parameters with a group of parameters masked. In this condition, **Actual diff in parameters** $= \hat{\boldsymbol{\theta}}_w - \hat{\boldsymbol{\theta}}_{\mathbf{1}}$, where $\hat{\boldsymbol{\theta}}_w$ is trained with $||w||_1$ number of parameters randomly masked and $\hat{\boldsymbol{\theta}}_{\mathbf{1}}$ is optimized without any mask. **Predicted diff in parameters** is calculated using Equation 3. We tested the difference between our method and the exact value of the parameters under different numbers of mask parameters using Logistic regression.

The result is shown in Figure 1. As the figure shows, especially when $||w||_1$ is relatively small, the parameter results predicted by our method are almost the same as the real results. When w is large, our prediction and the real value are still highly correlated. Our method still works very well even when we mask 60% of the parameters. Using our method, we only need to train once to be able to approximate any group of masks whether there is any relationship between them or not.

5.2 UNDERSTANDING MODEL STRUCTURE

The parameter-based influence functions can help us understand the structure of the model. In this subsection, we will show the contribution of neural network depth to the model. We trained a multi-layer convolutional Alex neural network on MNIST dataset (Deng, 2012), grouped the convolutional parameters by each layer, and calculate the impact of them on the model. The following results can be obtained. For a network with five layers of convolution kernels, the effects of these five layers were 0.2150, 0.3208, 0.5389, 0.9702, 0.9852. Then we increase the depth of the convolutional neural network. When the number of convolution kernels reached eleven, the effects of each convolutional layer were 0.0957, 0.1400, 0.2272, 0.3751, 0.2340, 0.0989, 0.0927, 0.1027, 0.1053, 0.1042, 0.1209.

The above results can help us understand the model structure. Firstly, when the depth of the convolutional neural network is relatively shallow, as the depth increases, the contribution of the deep convolutional layer to the model becomes larger and larger. When the depth increases to a certain level, the contribution of the convolutional layer to the model will decrease instead. This is completely consistent with the research results of other papers (He et al., 2016; Simonyan & Zisserman, 2014; Goodfellow et al., 2016). In addition, the last layer of the convolutional layer is followed by the classifier, so compared with the previous layers, its contribution is more important, as the depth of the model increases, the parameter scale of the model is constantly increasing, and each layer is the influence of the model is also diminishing.

5.3 MODEL PRUNING

Model pruning can help us reduce the size of the model while providing better generalization. Common model pruning is mainly divided into two categories, structured pruning, and unstructured pruning. Among them, structured pruning is mainly used to reduce the computational cost of the model, and unstructured pruning is more inclined to reduce the parameter scale, thereby reducing the size of the model. As a parameter-based influence function, our method can be used to evaluate the contribution of a certain parameter or a certain group of parameters to the model. Therefore, we

[1] http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Methods	Pruning rates								
	20%	30%	40%	50%	60%	70%	80%	90%	95%
random	0.944	0.869	0.786	0.488	0.207	0.116	0.113	0.113	0.113
random, retraining	0.989	0.989	0.988	0.988	0.988	0.986	0.983	0.113	0.113
our method	0.988	0.988	0.972	0.941	0.869	0.708	0.514	0.174	0.113
our method, retraining	0.989	0.990	0.991	0.991	0.989	0.989	0.989	0.980	0.975

Table 1: **The accuracy on the test set after pruning**. The accuracy of the model on the test dataset under the random channel pruning and the parameter-based influence function pruning. We respectively give the accuracy after pruning and the accuracy that is retrained after pruning.

can use the parameter-based influence function to perform model compression, whether it is a structured group of parameters, or a specific parameter, or even a random group of parameter clusters based on imagination, we can always get their influences on the model.

We used the parameter-based influence functions as the pruning rule. We trained an Alex network with five layers on the MNIST dataset. The accuracy of the full trained network on the test set is 0.989. We judged whether to prune the channel by calculating its contribution to the model. We cut the channels evenly on each layer according to the pruning rate. To illustrate the effectiveness of our method, we compared it with random pruning, the results are shown in Table 1.

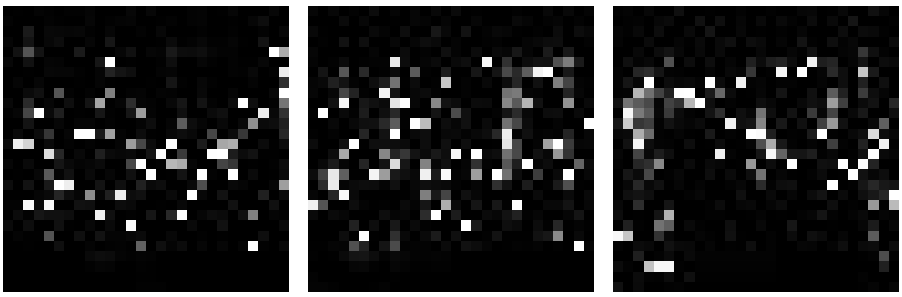
We can see that when the pruning rate is less than 30%, our method does not even need to be fine-tuned. This is also consistent with our theory. Our method essentially approximates the optimal training model after masking some parameters. When the pruning rate reaches 50%, even without retraining, our method can achieve an accuracy of 0.941. As the proportion of pruning increases, after using our method with retraining, the accuracy on the test set increases instead, which also shows that pruning can generalize the model. Finally, even if the pruning rate is as high as 95%, our method can still get a nice result after retraining. Random pruning can no longer recover the effect through retraining. This shows that our method can not only directly approximate the optimal model when the pruning rate is low but also can find the most important parameters after the pruning rate is high so that better results can be obtained after retraining.

5.4 Feature importance ranking

Feature importance ranking is very important in machine learning (Goodfellow et al., 2016). By sorting the importance of features, we can intuitively judge which features have a large contribution to the model and which features have a small contribution, which can further help us understand the model, reduce the dimensionality of the dataset, or reduce the size of the model. We show that the parameter-based influence functions can help us rank the importance of features so that we can better understand each feature of the dataset.

The key idea is to calculate the change of the loss after removing each feature, which is used as a criterion for evaluating feature sensitivity. To show the importance of features more intuitively, we have trained multiple binary classifiers on the MNIST dataset. The sensitivity is calculated separately for each input feature of different two-classifier models. Draw the result of the calculation in the form of a picture. The result is shown in Figure 2, Figure 2a is the feature importance distribution map of the binary classifier of 0 and 8. Figure 2b represents the two classifiers of 1 and 4, and Figure 2c represents the two classifiers of 1 and 7. The lighter the color, the more important this feature is.

First of all, it must be emphasized that we are judging the importance of the features in the binary classification model trained on 10800 training samples, not the importance of the model's features to a specific test point. Therefore, our feature importance distribution map only has a rough outline. In other words, it is to some extent the accumulation of the feature importance of a large number of training samples. We then analyze the feature importance distribution map. The edge features of the three images in Figure 2 are black which means that the characteristics of the edge are not important. This is consistent with our intuitive feeling. The handwritten fonts of the MNIST dataset are mainly concentrated in the middle of the image.



(a) $0, 8$ classifier (b) $1, 4$ classifier (c) $1, 7$ classifier

Figure 2: **Feature importance distribution map.** The importance of each feature in the binary classifier. The lighter the color, the more important the feature is. (a). The binary classifier of MNIST dataset 0 and 8. (b). The binary classifier of MNIST dataset 1 and 4. (c). The binary classifier of MNIST dataset 1 and 7.

Finally, let's analyze each picture carefully. For the binary classifiers with 0 and 8, the numbers 0 and 8 overlap in the upper and lower arcs to a certain extent. The main difference between them is in the middle. The result in Figure 2a is consistent with our guess, we can hardly find a half arc above and below. The middle part is a bit blurry due to the superposition of the importance of training samples, but we can still see some crossed contours. In the same way, we can see some acute angles like '$\angle$' corresponding to the number 4 in Figure 2b. In Figure 2c, we can see the horizontal line corresponding to the number 7. It is worth mentioning that many people will draw a dash on the vertical line when writing the number 7 to distinguish it from the number 1. This feature is also clearly shown in our feature sensitivity map. The part of the vertical line where the numbers 1 and 7 overlap is not visible in Figure 2c. This shows that the vertical line is not an important feature to distinguish between the numbers 1 and 7. The results in Figure 2 are completely consistent with our intuitive feelings.

6 Discussion and Conclusion

We start from the disturbance of the parameters and apply the influence function to them. Through a series of verification and application experiments, our method is effective and versatile. The test results on the libsvm dataset 'splic-scale' show that under linear conditions, our method still has a strong correlation with the true value for the 50% mask rate of the parameters. In a convolutional neural network, 30% of the parameters are masked in units of channels, and the accuracy can be basically maintained without retraining. Even if the mask is 95% of the parameters, it can be restored to an accuracy of 0.975 after retraining. Our method can also get good results in ranking the importance of the model. These validation and experiments show that our method has a good performance in various fields.

In fact, the core of the influence functions is to perturb the model. In order to explore the impact of disturbance on the model, existing work has focused on studying the disturbance of the dataset, and we deduce the influence of parameter disturbance on the model. Our method can be used to calculate the influence of any combination of parameters. The approximate calculation method based on Hessian vector product does not lose the correlation between the parameters, which makes our group of parameter influence functions valuable.

There are many possible applications of our method in other potential fields. For example, in some scenarios, we may want to know whether there is a correlation between multiple seemingly unrelated parameters in the model. The influence of each masked parameter on the model is calculated separately, and then they are regarded as a group to calculate the influence on the model. We can judge whether there is an internal correlation between these parameters by comparing them. In general, we give a new perspective from the parameter-based influence functions which provides a useful tool for us to understand the model from another angle.

REFERENCES

Alireza Aghasi, Afshin Abdi, Nam Nguyen, and Justin Romberg. Net-trim: convex pruning of deep neural networks with performance guarantee. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Samyadeep Basu, Xuchen You, and Soheil Feizi. On second-order group influence functions for black-box predictions. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

Roberto Battiti. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on neural networks*, 5(4):537–550, 1994.

Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.

Marc-Etienne Brunet, Colleen Alkalay-Houlihan, Ashton Anderson, and Richard Zemel. Understanding the origins of bias in word embeddings. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

Hongge Chen, Si Si, Yang Li, Ciprian Chelba, Sanjiv Kumar, Duane Boning, and Cho-Jui Hsieh. Multi-stage influence function. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Jianbo Chen, Mitchell Stern, Martin J Wainwright, and Michael I Jordan. Kernel feature selection via conditional covariance minimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

R Dennis Cook and Sanford Weisberg. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics*, 22(4):495–508, 1980.

R Dennis Cook and Sanford Weisberg. *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Xin Dong, Shangyu Chen, and Sinno Jialin Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems (NeurIPS)*. Morgan Kaufmann, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the 29th IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

Yotam Hechtlinger. Interpretation of prediction models using the input gradient. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Md Monirul Kabir, Md Monirul Islam, and Kazuyuki Murase. A new wrapper feature selection approach using neural network. *Neurocomputing*, 73(16-18):3273–3283, 2010.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

Pang Wei Koh, Kai-Siang Ang, Hubert HK Teo, and Percy Liang. On the accuracy of influence functions for measuring group effects. *arXiv preprint arXiv:1905.13289*, 2019.

Jan Larsen, Lars Kai Hansen, Claus Svarer, and M Ohlsson. Design and regularization of neural networks: the optimal use of a validation set. In *Neural Networks for Signal Processing VI. Proceedings of the 1996 IEEE Signal Processing Society Workshop*, pp. 62–71. IEEE, 1996.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1990.

Ismael Lemhadri, Feng Ruan, Louis Abraham, and Robert Tibshirani. Lassonet: A neural network with feature sparsity. *Journal of Machine Learning Research (JMLR)*, 22(127):1–29, 2021a.

Ismael Lemhadri, Feng Ruan, and Rob Tibshirani. Lassonet: Neural networks with feature sparsity. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021b.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, 2020.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Russell Reed. Pruning algorithms-a survey. *IEEE transactions on Neural Networks*, 4(5):740–747, 1993.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016.

Debaditya Roy, K Sri Rama Murty, and C Krishna Mohan. Feature selection using deep neural networks. In *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015.

Peter Schulam and Suchi Saria. Can you trust this prediction? auditing pointwise reliability after learning. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (ICAIS)*, 2019.

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

Blaž Škrlj, Sašo Džeroski, Nada Lavrač, and Matej Petkovič. Feature importance estimation with self-attention networks. In *Proceedings of 24th European Conference on Artificial Intelligence (ECAI)*, 2020.

Le Song, Alex Smola, Arthur Gretton, Karsten M Borgwardt, and Justin Bedo. Supervised feature selection via dependence estimation. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, 2007.

Yi Sun, Xiaogang Wang, and Xiaoou Tang. Sparsifying neural network connections for face recognition. In *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Antanas Verikas and Marija Bacauskiene. Feature selection with neural networks. *Pattern recognition letters*, 23(11):1323–1335, 2002.

A PROOF OF THEOREM 1

Proof. Recall that the L_2-regularized empirical risk is as follows

$$R(\boldsymbol{\theta}) \overset{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{x_i}, y_i, \boldsymbol{\theta}) + \frac{\lambda}{2}||\boldsymbol{\theta}||_2^2. \tag{7}$$

We assume that R is twice-differentiable and strictly convex in $\boldsymbol{\theta}$, i.e.,

$$H_\lambda \overset{\text{def}}{=} \nabla_\theta^2 R(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\nabla_\theta^2\ell(\mathbf{x_i}, y_i, \boldsymbol{\theta}) + \lambda\mathbf{I}. \tag{8}$$

Note that the masked dimensions in $\hat{\boldsymbol{\theta}}_\mathbf{1}$ is $\mathbf{0}$, we have the obvious condition that $\boldsymbol{w}\odot\Delta\boldsymbol{\theta}+\boldsymbol{w}\odot\hat{\boldsymbol{\theta}}_\mathbf{1} = 0$, where $\Delta\boldsymbol{\theta} = \boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_\mathbf{1}$. Thus our goal is to solve the following optimization problem

$$\begin{aligned}\hat{\boldsymbol{\theta}}_w = \arg\min_{\boldsymbol{\theta}\in\Theta} \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{x_i}, y_i, \boldsymbol{\theta}_w) + \frac{\lambda}{2}||\boldsymbol{\theta}_w||_2^2,\\ \text{s.t.} \quad \boldsymbol{w}\odot\Delta\boldsymbol{\theta} + \boldsymbol{w}\odot\hat{\boldsymbol{\theta}}_\mathbf{1} = \mathbf{0}.\end{aligned} \tag{9}$$

Use the Largrange multipliers method, we can form the Lagrangian function

$$\mathcal{L} = R(\boldsymbol{\theta}_w) + \boldsymbol{\psi}^{\mathrm{T}}(\boldsymbol{w}\odot\Delta\boldsymbol{\theta} + \boldsymbol{w}\odot\hat{\boldsymbol{\theta}}_\mathbf{1}), \tag{10}$$

where $\boldsymbol{\psi}$ and $\boldsymbol{\theta}$ are the same dimension. Now we can calculate the gradient

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}} &= \nabla_\theta R(\boldsymbol{\theta}_w) + \boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{w}\\ &= \nabla_\theta R(\hat{\boldsymbol{\theta}}_\mathbf{1}) + \frac{1}{n}H_\lambda\Delta\boldsymbol{\theta} + \boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{w}\\ &= \frac{1}{n}H_\lambda\Delta\boldsymbol{\theta} + \boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{w} = \mathbf{0}\\ \frac{\partial\mathcal{L}}{\partial\boldsymbol{\psi}} &= \Delta\boldsymbol{\theta} + \boldsymbol{w}\odot\hat{\boldsymbol{\theta}}_\mathbf{1} = \mathbf{0}.\end{aligned} \tag{11}$$

For the first formula above, we use Taylor expansion at $\hat{\boldsymbol{\theta}}_\mathbf{1}$. Since $\hat{\boldsymbol{\theta}}_\mathbf{1}$ minimzes $R(\boldsymbol{\theta})$. we have $\nabla R(\hat{\boldsymbol{\theta}}_\mathbf{1}) = \mathbf{0}$. We can solve the equation that

$$\Delta\boldsymbol{\theta} = -H_\lambda^{-1}[\hat{\boldsymbol{\theta}}_\mathbf{1}\odot(H_\lambda^{-1}\boldsymbol{w})^{\odot-1}\odot\boldsymbol{w}]. \tag{12}$$

Thus

$$\mathcal{I}_{\text{params}} \overset{\text{def}}{=} \left.\frac{d\hat{\boldsymbol{\theta}}_w}{d\boldsymbol{w}}\right|_{\boldsymbol{w}=\mathbf{0}} = -H_\lambda^{-1}[\boldsymbol{w}\odot\hat{\boldsymbol{\theta}}_\mathbf{1}\odot(H_\lambda^{-1}\boldsymbol{w})^{\odot-1}], \tag{13}$$

which finishs the proof. □

B PROOF OF LEMMA 1

Proof. From the definition we have

$$\begin{aligned}\mathcal{L}_{\text{Sensitivity}} &\overset{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n}\left(\left(\ell(\mathbf{x_i}, y_i, \hat{\boldsymbol{\theta}}_w) + \frac{\lambda}{2}||\hat{\boldsymbol{\theta}}_w||_2^2\right) - \left(\ell(\mathbf{x_i}, y_i, \hat{\boldsymbol{\theta}}_\mathbf{1}) + \frac{\lambda}{2}||\hat{\boldsymbol{\theta}}_\mathbf{1}||_2^2\right)\right)\\ &= \nabla_\theta R(\hat{\boldsymbol{\theta}}_\mathbf{1})d\hat{\boldsymbol{\theta}}_w + \frac{1}{2}d\hat{\boldsymbol{\theta}}_w^{\mathrm{T}}H_\lambda d\hat{\boldsymbol{\theta}}_w + O(||d\hat{\boldsymbol{\theta}}_w||^3)],\end{aligned} \tag{14}$$

Since $\nabla_\theta R(\hat{\boldsymbol{\theta}}_\mathbf{1}) = \mathbf{0}$, input Equation 13 into Equation 14 we have

$$\begin{aligned}\mathcal{L}_{\text{Sensitivity}} &= \frac{1}{2}d\hat{\boldsymbol{\theta}}_w^{\mathrm{T}}H_\lambda d\hat{\boldsymbol{\theta}}_w\\ &= \frac{1}{2}([\boldsymbol{w}\odot\hat{\boldsymbol{\theta}}_\mathbf{1}\odot(H_\lambda^{-1}\boldsymbol{w})^{\odot-1}])^{\mathrm{T}}H_\lambda^{-1}[\boldsymbol{w}\odot\hat{\boldsymbol{\theta}}_\mathbf{1}\odot(H_\lambda^{-1}\boldsymbol{w})^{\odot-1}],\end{aligned} \tag{15}$$

which finishs the proof. □