Neuron with Steady Response Leads to Better Generalization

Qiang Fu *†
Microsoft Research Asia
Beijing, China
qifu@microsoft.com

Lun Du *
Microsoft Research Asia
Beijing, China
lun.du@microsoft.com

Haitao Mao *‡
Michigan State University
Michigan, U.S.
haitaoma@msu.edu

Xu Chen *
Microsoft Research Asia
Beijing, China
xu.chen@microsoft.com

Wei Fang *‡
Tsinghua University
Beijing, China
fw17@mails.tsinghua.edu.cn

Shi Han
Microsoft Research Asia
Beijing, China
shihan@microsoft.com

Dongmei Zhang
Microsoft Research Asia
Beijing, China
dongmeiz@microsoft.com

Abstract

Regularization can mitigate the generalization gap between training and inference by introducing inductive bias. Existing works have already proposed various inductive biases from diverse perspectives. However, none of them explores inductive bias from the perspective of class-dependent response distribution of individual neurons. In this paper, we conduct a substantial analysis of the characteristics of such distribution. Based on the analysis results, we articulate the Neuron Steadiness Hypothesis: the neuron with similar responses to instances of the same class leads to better generalization. Accordingly, we propose a new regularization method called Neuron Steadiness Regularization (NSR) to reduce neuron intra-class response variance. Based on the Complexity Measure, we theoretically guarantee the effectiveness of NSR for improving generalization. We conduct extensive experiments on Multilayer Perceptron, Convolutional Neural Networks, and Graph Neural Networks with popular benchmark datasets of diverse domains, which show that our Neuron Steadiness Regularization consistently outperforms the vanilla version of models with significant gain and low additional computational overhead.

1 Introduction

Deep Neural Network (DNN) achieves state-of-the-art results in a wide range of areas and has various applications across industries, including self driving cars [37], virtual assistants [38], intelligent healthcare [32], personalized recommendation [34], etc. DNN's success usually relies on a plenty amount of training data. However, DNN's generalization is often hampered in many domains where training data is insufficient because data annotation is labor-intensive and expensive.

*These authors contributed equally to the work.
†Corresponding Author
‡Work performed during the internship at MSRA.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Regularization is a popular method that helps to improve generalization through introducing inductive bias. Regularization is one of the key elements of machine learning, particularly of deep learning [8]. Specifically, inductive bias represents assumptions about the model properties other than the consistency of outputs with targets. There have been tremendous efforts in identifying such desired properties, which results in a series of widely used regularization methods. For example, L2 Regularization [36, 21] penalizes large norms of model weights, which puts constraints on “parameter scale". L1 regularization improves “sparseness" by rewarding zero weight or neuron response. Jacobian regularization [43, 13] minimizes the norm of the input-output Jacobian matrix to improve the “smoothness" of the learned mapping function. Orthogonal regularization [4, 1] enlarges “weight diversity" to reduce the feature redundancy. Batch Normalization [15] promotes “training dynamics stability" by reducing the internal covariate shift.

Although the existing works have already proposed various inductive biases from diverse perspectives, including the aforementioned “parameter scale", “sparseness", “smoothness", “weight diversity" and “training dynamics stability", to the best of our knowledge, there is no work to explore inductive bias from the perspective regarding the characteristics of neuron response distribution on each class. From another point of view, the existing works leverage the information related to weights (parameter scale regularization), weight correlations (orthogonal regularization), derivatives of mapping function (smoothness regularization), collective neurons responses (sparseness regularization, Batch Normalization), but none of them considers the intra-class response distribution of individual neurons.

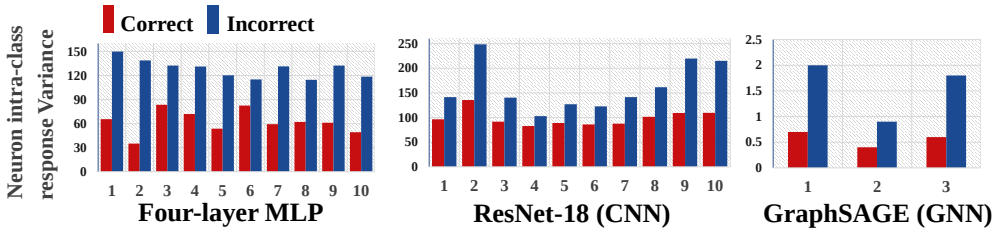
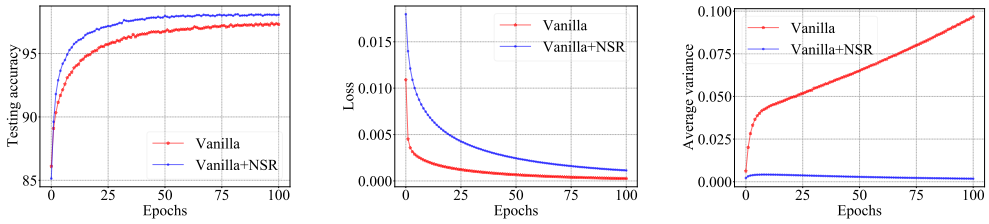


Figure 1: Comparison of the intra-class response variance of correctly and incorrectly classified testing samples for different architectures: four-layer MLP for MNIST, ResNet-18 for CIFAR-10, and GraphSAGE for PubMed. The horizontal axis and the vertical axis represent class indexes and the value of intra-class response variance, respectively. Each bar represents the intra-class response variance aggregated from all neurons in the penultimate layer.



(a) Testing accuracy (b) Cross entropy loss (c) Average intra-class variance

Figure 2: Training procedure of the vanilla four-layer MLP and the four-layer MLP with NSR on MNIST. (a) represents the testing accuracy; (b) and (c) illustrate the corresponding cross entropy loss and average intra-class response variance of these two models on the training set.

In this paper, we study the characteristics of the intra-class response distribution of each individual neuron to identify the new regularization method. In more detail, for each individual neuron, we analyze the variance of its response to samples of the same class, which is called **neuron intra-class response variance**. We find that such intra-class response variance has an obvious correlation with classification correctness. As shown in Figure 1, we find the correctly classified samples usually have smaller intra-class response variance compared to the misclassified samples. Besides, it can be observed that the vanilla model with cross entropy as the optimization target usually could not control

intra-class response variance well, as shown in Figure 2, which leaves a potential improvement space for the regularization. The details of experiments and observations are explained in Section 2.

Based on these observations, we articulate a **Neuron Steadiness Hypothesis**: neuron with similar responses to instances of the same class, i.e., smaller neuron intra-class response variance can lead to better generalization. Accordingly, we propose the regularization method called **Neuron Steadiness Regularization (NSR)** to improve generalization by penalizing large neuron intra-class response variance.

Based on the Complexity Measure theory in Section 3.3, we conduct theoretical analysis of the effectiveness of NSR for improving generalization. In addition, our regularization method shows significant improvements in various network architectures, including Graph Neural Networks (GNN), Convolution Neural Networks (CNN), and Multilayer Perceptron (MLP).

To sum up, our contributions are as follows:

- We articulate a Neuron Steadiness Hypothesis and demonstrate its validity. It provides a new regularization perspective based on the neuron-level class-dependent response distribution.
- We propose a new regularization method, Neuron Steadiness Regularization, to improve generalization ability. The method is computationally efficient and general enough to be applied to various architectures and tasks. Theoretical analyses guarantee its effectiveness.
- Extensive experiments are conducted on multiple types of datasets like images, citation graphs and product graphs, with various network architectures including GNN, CNN and MLP. Significant improvements evidently verify the effectiveness of Neuron Steadiness Regularization.

2 Observations

In this section, we verify the Neuron Steadiness Hypothesis by experiments with the following identified observations.

2.1 Correlation between neuron intra-class response variance and classification correctness

The neuron intra-class response variance is derived from neuron response distribution which could be obtained by recording neuron responses when we feed input samples to the model. In this paper, for the neuron with ReLU as its activation function, we do not take its zero-response into account for calculation, because zero-response corresponds to the inactivated state where the neuron does not respond at all. In other words, for neurons with ReLU as the activation function, we only record its non-zero responses to represent its response distribution.

With the obtained response distribution of any given neuron, it is relatively straightforward to calculate our discussed statistics, i.e., intra-class response variance, which is the mean squared deviation of the neuron response from the mean of the intra-class response. Then, for each neuron, we calculate intra-class response variance corresponding to the correctly classified samples and misclassified samples, respectively. Finally, we aggregate such two respective variances of all neurons in the penultimate layer separately and present them in Figure 1.

Figure 1 shows that for different networks, the average intra-class response variance of correctly classified samples is smaller than that of misclassified ones on arbitrary class. It indicates the strong correlation between classification correctness and neuron intra-class response variance.

2.2 Dynamics of neuron intra-class response variance during training procedure

We investigate the tendency of neuron intra-class response variance along with the training procedure. For comparison, we perform the analysis on the vanilla model and the model with our proposed neuron steadiness regularization. We calculate the intra-class response variance of each neuron on the entire training set after each training epoch. Then, the intra-class response variance of all neurons is averaged and denoted as average variance in Figure 2 (c). We also show the testing accuracy and the training cross entropy loss in Figure 2 (a) and (b), respectively. Other architectures demonstrate similar tendencies and can be found in Appendix D.

From Figure 2, we could see that the cross entropy objective keeps being optimized during the training procedure which leads to increasing classification accuracy for both models. However, for the vanilla version, the average neuron intra-class response variance is growing larger because the model training does not impose constraints on neuron intra-class response variance. For the model trained with our proposed regularization, the neuron intra-class response variance is controlled and decreases after a few epochs. More importantly, the learned model with well-controlled intra-class response variance has higher testing accuracy than the vanilla version although its corresponding cross entropy loss is even larger. In addition, we could also see that regulating neuron intra-class response variance may also help the optimization procedure to achieve higher accuracy in earlier epochs.

To conclude all the above observations, it is reasonable to design the regularization based on the neuron steadiness hypothesis, i.e., reducing neuron intra-class response variance, for better generalization.

3 Method

In this section, we first describe the proposed **Neuron Steadiness Regularization**. Then we further introduce several techniques for computational efficiency.

3.1 Definition of Neuron Steadiness Regularization

Neuron Steadiness Regularization (NSR) for a specific neuron is defined as the summation of intra-class response variances of different classes. The NSR for the n-th neuron can be formulated as:

$$\sigma_n = \sum_{j=1}^{J} \alpha_j \cdot \mathrm{Var}\left(X_{n,j}\right) = \sum_{j=1}^{J} \alpha_j \cdot \mathbb{E}\left[\left(X_{n,j} - \mathbb{E}\left[X_{n,j}\right]\right)^2\right] \tag{1}$$

where $X_{n,j}$ is a random variable denoting the n-th neuron's response for a sample belonging to the j-th class. J is the number of classes. $\alpha_j = \frac{z_j}{\sum_i z_i}$ is the prior probability of the j-th class where z_j is the sample amount of j-th class. Notice that α_j is not a hyper-parameter, and it only presents the importance of different classes. With the NSR term defined for each individual neuron, the overall regularization is derived by applying NSR term to all neurons in the network as follows:

$$\mathcal{L}_S = \sum_{n=1}^{N} \lambda_n \sigma_n \tag{2}$$

where $\mathcal{L}_S$ represents the NSR term for the entire network, and N is the number of neurons in the network. λ_n is the hyper-parameter to control the regularization intensity. In this paper, for practical simplicity, λ is set as the same value for all neurons. Adding the overall regularization term to the main training target, i.e., the cross entropy loss, the final regularized loss function can be written as:

$$\mathcal{L} = \mathcal{L}_C + \mathcal{L}_S = \mathcal{L}_C + \lambda \sum_{n=1}^{N} \sigma_n, \tag{3}$$

where $\mathcal{L}_C$ represents cross entropy loss.

3.2 Practical Implementation

3.2.1 Mini-batch Training

In order to use mini-batch training, we adapt our NSR method to do forward and backward propagation based on mini-batches of samples. To be specific, we first transform Eq. (1) as:

$$\begin{aligned} \sigma_n &= \sum_{j=1}^{J} \alpha_j \left(\mathbb{E}\left[X_{n,j}^2\right] - \mathbb{E}^2\left[X_{n,j}\right]\right) \\ &= \mathbb{E}\left[\sum_{j=1}^{J} \alpha_j X_{n,j}^2\right] - \sum_{j=1}^{J} \alpha_j \mathbb{E}^2\left[X_{n,j}\right]. \end{aligned} \tag{4}$$

The first term has the same form as typical loss functions, i.e., an expectation of the function of data samples, which can be easily generalized to a mini-batch training setting via estimating the expectation with a batch of samples. Although the second term can be estimated in the same way, the square operation magnifies the estimation error, especially when the batch size is not large enough.

To alleviate the problem while not introducing much computing overhead, we propose a memory queue-based estimation method that allows us to leverage more history samples for estimation without additional sampling and forward/backward computation. For each class, we record the number of samples and the summation values of the neurons' response within each batch. To further reduce the storage overhead, we only maintain these values of the latest M batches by two M-length queues.

More specifically, an element $c_{m,j}$ in the first memory queue is the count number of j-th class instances in the m-th batch, represented as:

$$c_{m,j} = \sum_{y_i \in Y_m} \delta\left(y_i = j\right), \tag{5}$$

where Y_m is the set of labels in the m-th batch, y_i is the label of i-th sample, and δ is a characteristic function, i.e., $\delta(condition) = 1$ if $condition$ is satisfied, otherwise $\delta(condition) = 0$. An element $s_{m,j}^{(n)}$ in the second memory queue is the summation value of n-th neuron's response for the samples belonging to the j-th class within the m-th batch, represented as:

$$s_{m,j}^{(n)} = \sum_{x_i^{(n)} \in \mathcal{X}_m^{(n)}} \delta\left(y_i = j\right) \cdot x_i^{(n)}, \tag{6}$$

where $\mathcal{X}_m^{(n)}$ is the set of the n-th neuron's response within the m-th batch and $x_i^{(n)}$ is the n-th neuron's response of the i-th sample. When a new batch is fed, the estimation of expectation $\mathbb{E}\left[X_{n,j}\right]$ can be updated by the following steps:

$$\begin{gathered} C_j := C_j - c_{0,j} + c_{*,j}, \quad S_j^{(n)} := S_j^{(n)} - s_{0,j}^{(n)} + s_{*,j}^{(n)} \\ \hat{\mathbb{E}}\left[X_{n,j}\right] := S_j^{(n)} / C_j, \end{gathered} \tag{7}$$

where $\hat{\mathbb{E}}\left[X_{n,j}\right]$ is the estimation of expectation $\mathbb{E}\left[X_{n,j}\right]$, $C_j = \sum_m c_{m,j}$, $S_j^{(n)} = \sum_m s_{m,j}^{(n)}$, and $c_{*,j}$, $s_{*,j}^{(n)}$, as new elements appended to the queues, represent the count number and the summation for the new batch. Based on this dynamic update method, the additional memory overhead is negligible, and we give a space complexity analysis in Appendix E.

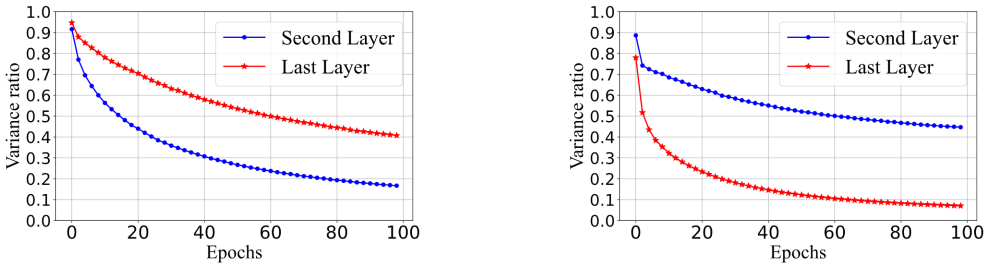
3.2.2 Layer Selection criterion for Applying NSR

We find that there is a correlation or redundancy among the steadiness constraints of different layers, meaning that applying NSR on different layers of the network has an overlapping effect on neuron steadiness control. In more detail, from Figure 3, we can see that every layer's variance ratio decreases even only one specific layer is applied with NSR. Such steadiness correlation or redundancy among layers is not surprising because different layers are served as input/output of each other.

Due to the aforementioned redundancy, considering the trade-off between performance gain and computational overhead, we select only one layer to apply NSR. As our NSR is used to reduce neuron intra-class response variance, naturally, the intuitive criterion is to apply NSR to the layer with the largest aggregated neuron intra-class response variance. Detailed experiment results in RQ4 of the experiment section show such a criterion works well. In this paper, if not otherwise specified, we apply NSR to only one particular layer determined by this layer selection criterion.

3.3 Theoretical Analysis

In this section, we theoretically analyze the difference in model generalization ability with or without our proposed regularization method NSR based on Complexity Measure [35]. The complexity Measure is one of the mainstream methods to measure the generalization ability of a deep learning model. A **lower complexity measure** means a **better generalization ability**. Formally, a Complexity Measure is a measure function $\mathcal{M} : \{\mathcal{H}, \mathcal{S}\} \rightarrow \mathbb{R}^+$ where $\mathcal{H}$ is a class of models and $\mathcal{S}$ is a training set. According to the definition, if a complexity measure of a given model tends to be 0, the model



(a) NSR only applied on the second layer (b) NSR only applied on the last layer

Figure 3: The trend of variance ratio along with the training procedure on MNIST. The variance ratio is the neuron intra-class response variance of the three-layer MLP applied with NSR, (a) only on the second layer and (b) only on the last layer, divided by the corresponding variance of the vanilla three-layer MLP.

should have the best generalization ability. Note that, several different complexity measures have been proposed.

We select Consistency of Representations [33] as the concrete Complexity Measure used in our theoretical analysis. This measure is designed based on Davies-Bouldin Index [5] and is the Winning Solution of the NeurIPS 2020 Competition on Predicting Generalization in Deep Learning. Mathematically, for a given dataset and a model, we define the following statistics:

$$S_i = \left(\frac{1}{n_i} \sum_{\tau}^{n_i} |\mathcal{O}_i^{(\tau)} - \mu_{\mathcal{O}_i}|^p \right)^{1/p} \quad \text{for } i = 1 \cdots J \tag{8}$$

$$M_{i,j} = ||\mu_{\mathcal{O}_i} - \mu_{\mathcal{O}_j}||_p \qquad \text{for } i, j = 1 \cdots J, \tag{9}$$

where i and j are two different classes, and n_i is the number of samples belonging to the class i. $\mathcal{O}_i^{(\tau)}$ is the output representation of the τ-th sample belonging to class i for the final layer, and $\mu_{\mathcal{O}_i}$ is the cluster centroid of the representations of class i. S_i is a measure of scatter within representations of class i, and $M_{i,j}$ is a measure of separation between representations of classes i and j. Then, the complexity measure based on the Davies-Bouldin Index is defined as:

$$\mathcal{C} = \frac{1}{J} \sum_{i=1}^{J} \max_{i \neq j} \frac{S_i + S_j}{M_{i,j}}. \tag{10}$$

Based on the definition of the complexity measure $\mathcal{C}$, we have the following lemma:

Lemma 3.1. *For a multi-class classification problem, when (1) a deep learning model utilizing the Cross Entropy loss with an NSR regularization term on any of its intermediate layers is optimized via gradient descent and (2) the capacity of the model is sufficiently large, the Consistency of Representations Complexity Measure on this model $\mathcal{C}$ will tend to be 0. Under the same condition, when the deep learning model is only optimized by the Cross Entropy loss without an NSR regularization term, there will be infinite local minima where the complexity measure $\mathcal{C}$ will be a positive number.*

The proof is shown in Appendix A. Intuitively, this lemma shows that in an ideal condition (i.e., the model is sufficiently large), if the definition of the Consistency of Representations Complexity Measure is utilized, a deep learning model with NSR regularization term will be guaranteed to converge to local minima with the best generalization ability. In contrast, the model without NSR will not have the same guarantee.

4 Experiment

In this section, we conduct experiments over a variety of datasets to examine the performance of our neuron steadiness regularization on three extensively used neural network architectures. We design a series of experiments to answer the following research questions. **RQ1**: How does NSR perform on various datasets and different neural architectures? **RQ2**: Does NSR outperform other classical

regularization methods? **RQ3**: What is the effect of combining NSR with other popular methods like Batch Normalization or Dropout? **RQ4**: What if we apply NSR to multiple layers instead of one particular layer?

4.1 Experiment Setup

Network Architectures and Datasets. Three architectures utilized in experiments are Multilayer Perceptron (MLP), Convolutional Neural Network (CNN) and Graph Neural Network (GNN). For vanilla models used as our baselines in different architectures, we adopt ResNet-18 [11], VGG-19 [26] and ResNet-50 for CNN, GCN [17] and GraphSAGE [10] for GNN. We run five MLP models which are denoted as MLP-L, L indicates the number of layers including the input layer. We elaborate on the details of these vanilla models in Appendix C.2. As for benchmark datasets used in our experiments, MLP and CNN are applied to image recognition task on **MNIST** [22], **CIFAR-10** [19] and **ImageNet** [6] datasets, respectively. GNN is applied to node classification on four real-world graph datasets: **WikiCS** [30], **PubMed** [48], **Amazon-Photo** and **Amazon-Computers** [41]. Notice that ImageNet is a large benchmark dataset with 1000 classes. Details are shown in Appendix C.4.

Table 1: Error rate of applying our NSR on five MLP models for MNIST.

Model	MLP-3	MLP-4	MLP-6	MLP-8	MLP-10
Vanilla (%)	3.09 ± 0.10	2.29 ± 0.07	2.44 ± 0.09	2.87 ± 0.09	3.06 ± 0.06
Vanilla+NSR (%)	$\mathbf{2.80 \pm 0.08}$	$\mathbf{1.64 \pm 0.04}$	$\mathbf{1.76 \pm 0.06}$	$\mathbf{1.98 \pm 0.09}$	$\mathbf{1.72 \pm 0.14}$
Gain	9.39%	28.38%	27.87%	30.87%	43.79%

Experiment Settings. We follow the typical implementation settings to conduct our experiments. For ResNet-18 and VGG-19 on CIFAR-10, we follow the detailed setting of [49, 9]. For ResNet-50 on ImageNet, we follow the official implementation provided by torchversion library [4]. For GraphSAGE and GCN, we follow the implementation setting of [28]. For MNIST dataset, we divide 60000 training images into the training set with 50000 samples and the validation set with the remaining 10000 samples to select hyper-parameter. For each of the four graph datasets, it is randomly split into training, validation, and testing sets with a ratio of 6:2:2. Note that, SGD [40] is used to optimize MLP, and Adam [16] for other models except for ResNet-18 that is optimized by Momentum [40] according to the implementation setting of [49]. We use the typical setting of batch size as 100 for all experiments. To ensure the model convergence, training epochs are set as 100 for both MLP and GNN, 200 for ResNet and 500 for VGG.

Table 2: Error rate of applying our NSR on ResNet-18 and VGG-19 for CIFAR-10, and top-5 error rate on ResNet-50 for ImageNet.

Model	ResNet-18	VGG-19	ResNet-50
Vanilla	4.22 ± 0.07	9.19 ± 0.18	7.82 ± 0.07
Vanilla+NSR	$\mathbf{3.74 \pm 0.08}$	$\mathbf{8.09 \pm 0.17}$	$\mathbf{6.98 \pm 0.08}$
Gain	11.37%	11.97%	10.74%

In all experiments except for RQ4, we apply NSR to only one particular layer with the same λ for each neuron. For this only one hyper-parameter λ, like most other regularization methods, we apply a random search strategy to find its proper value ranging from 1e-2 to 10. The error rate is the evaluation metric and each result is averaged over 5 runs with different random seeds. The hardware environments are detailed in Appendix B.

4.2 Experiment Results

RQ1: Performance of NSR: Here, we discuss the performance of NSR over different models and present the results on Tab. 1 - 3 "Gain" is the percentage of relative reduction in the error rate. Tab. 1 demonstrates that NSR can improve the performance of MLP with different layers: the relative error rate is reduced by 9.39% at least and 43.79% at most. Besides, as the number of network layers increases, the gain of NSR shows a roughly upward trend. The four-layer MLP achieves the lowest classification error rate among all vanilla version baselines, and the accuracy becomes worse as the networks grow deeper. It indicates that deep MLP encounters a severe overfitting problem. The

[4] https://pytorch.org/hub/pytorch_vision_resnet/

success of our regularization in addressing such a problem reveals the importance of stabilizing the response of each individual neuron to instances from the same class.

For CNN models, Tab. 2 demonstrates that NSR can reduce the relative error rate for CIFAR-10 by 11.97% on VGG-19 and 11.37% on Resnet-18, and reduce the relative error rate (top-5) for ImageNet by 10.74% on ResNet-50. It is worth mentioning that VGG-19, ResNet-18 and ResNet-50 have already adopted Batch Normalization, Dropout regularization and Weight Decay in vanilla models, the accuracy gain of our method reveals that NSR can have extra benefits to ulteriorly enhance generalization ability when combined with Batch Normalization, Dropout and Weight Decay. We will show more evidence about this in RQ3.

Table 3: Error rate of applying our NSR on GCN and GraphSAGE over four graph datasets.

Dataset	Layers	GraphSAGE (%)	GraphSAGE+NSR (%)	GCN (%)	GCN+NSR (%)
PubMed	2	10.73 ± 0.06	$\mathbf{9.89 \pm 0.08}$	12.02 ± 0.00	$\mathbf{11.92 \pm 0.00}$
	3	10.20 ± 0.25	$\mathbf{9.48 \pm 0.12}$	12.76 ± 0.18	$\mathbf{12.19 \pm 0.11}$
	4	10.43 ± 0.17	$\mathbf{9.79 \pm 0.19}$	14.01 ± 0.07	$\mathbf{12.96 \pm 0.08}$
Amazon-Photo	2	5.82 ± 0.00	$\mathbf{4.54 \pm 0.10}$	6.73 ± 0.00	$\mathbf{6.27 \pm 0.00}$
	3	5.20 ± 0.14	$\mathbf{4.86 \pm 0.13}$	8.00 ± 0.11	$\mathbf{7.96 \pm 0.10}$
	4	6.37 ± 0.30	$\mathbf{5.62 \pm 0.59}$	10.24 ± 0.14	$\mathbf{9.03 \pm 0.25}$
Amazon-Computers	2	11.37 ± 0.55	$\mathbf{10.47 \pm 0.05}$	12.17 ± 0.07	$\mathbf{10.86 \pm 0.03}$
	3	11.88 ± 1.05	$\mathbf{10.22 \pm 0.54}$	14.90 ± 0.25	$\mathbf{13.66 \pm 0.12}$
	4	15.49 ± 0.90	$\mathbf{12.86 \pm 0.82}$	18.07 ± 0.74	$\mathbf{16.02 \pm 0.23}$
WikiCS	2	16.81 ± 0.21	$\mathbf{16.06 \pm 0.33}$	18.41 ± 0.06	$\mathbf{17.99 \pm 0.05}$
	3	15.97 ± 0.18	$\mathbf{15.27 \pm 0.21}$	18.66 ± 0.23	$\mathbf{18.10 \pm 0.27}$
	4	16.63 ± 0.31	$\mathbf{15.43 \pm 0.24}$	19.21 ± 0.31	$\mathbf{18.84 \pm 0.26}$

The number of layers of two GNN models varies from 2 to 4. It is a typical setting following empirical experiences as nodes will capture similar information from neighbors and result in over-smoothness when GNNs grow deeper. Tab. 3 shows that GraphSAGE and GCN applied with NSR outperform the vanilla model with different layer depths on all datasets. Specifically, GraphSAGE and GCN achieve an average improvement of 8.6% and 5.8%, respectively, and 17.0% improvement at most.

RQ2: Comparison with Other Regularization: The vanilla ResNet18 here does not use any regularization term and learning rate decay methods for fairness. We first compare our NSR method with several classical regularization methods shown in Tab. 4. Notice that, both our NSR and the other regularization methods for comparison have only one hyper-parameter to tune, and we utilize the same hyper-parameter searching strategy to select the best hyper-parameter values for all of them. As shown in Tab. 4, our NSR performs best among these regularization methods on all three models with different architectures, i.e., MLP-4 (MLP), ResNet-18 (CNN), GraphSAGE (GNN). The results indicate that our NSR has remarkable improvement and is general for various architectures.

RQ3: Combination with Other Regularization: We further investigate the effect of combining our NSR with other popular methods like Batch Normalization and Dropout. We conduct the experiments on MLP-4 and the result are organized in Tab. 5. From Tab. 5 we can find that both Batch Normalization and Dropout can reduce error rate compared with vanilla baseline, and adding our NSR on top of them can further promote the performance significantly. It indicates that NSR can provide complementary regularization benefits with Batch Normalization and Dropout.

Table 4: Error rate comparison of different regularization methods on different models.

	MLP-4	ResNet-18	GraphSAGE
Vanilla	2.29 ± 0.07	7.96 ± 0.12	11.37 ± 0.55
L1	2.27 ± 0.05	7.83 ± 0.23	10.81 ± 0.13
L2	2.27 ± 0.05	7.67 ± 0.18	10.68 ± 0.35
Jacobian	2.21 ± 0.04	7.90 ± 0.07	11.27 ± 0.45
NSR	$\mathbf{1.64 \pm 0.04}$	$\mathbf{7.20 \pm 0.09}$	$\mathbf{10.52 \pm 0.22}$

Table 5: Combination of our NSR with Batch Normalization (BN) and Dropout (DO) for training.

MLP-4	Vanilla	+ BN	+ BN&NSR
Error rate	2.29 ± 0.07	2.22 ± 0.04	$\mathbf{1.62 \pm 0.08}$
MLP-4	Vanilla	+ DO	+ DO&NSR
Error rate	2.29 ± 0.07	2.19 ± 0.04	$\mathbf{1.64 \pm 0.04}$

Table 6: Effect of applying NSR on different layer(s) of MLP-4. The subscript number indicates which layer(s) NSR is applied on.

	MLP	MLP_2	MLP_3	MLP_4	$\text{MLP}_{3,4}$
Error rate	2.29 ± 0.07	2.22 ± 0.08	1.90 ± 0.13	1.64 ± 0.04	$\mathbf{1.63} \pm \mathbf{0.08}$

Table 7: Effect of applying NSR on different layer(s) of GraphSAGE-2. The subscript number indicates which layer(s) NSR is applied on.

	GraphSAGE	GraphSAGE_1	GraphSAGE_2	$\text{GraphSAGE}_{1,2}$
Error rate	11.37 ± 0.55	10.47 ± 0.05	10.52 ± 0.22	$\mathbf{10.30} \pm \mathbf{0.17}$

RQ4: Layer selection for applying NSR: We have introduced how to select one particular layer to apply NSR in section 3.2.2. To evaluate such layer selection criterion, we compare the performances trained by applying NSR to multiple layers or to only one selected layer, respectively. Taking MLP-4 on MNIST and GraphSAGE-2 on Amazon-Computers as two examples, we apply NSR to different layer(s), and their corresponding error rates are listed in Tab. 6 $\sim$ Tab. 7, where model_l means NSR is applied to the l_{th} layer of the model. Notice that, except the first layer of MLP-4, the input layer, for the second, third, and last layer of MLP-4, their neuron intra-class response variance are 409, 510, and 1660, respectively. For GraphSAGE-2, its neuron intra-class response variance are 4.15 and 2.68, for the first and last layer, respectively. The variance of MLP-4 and GraphSAGE-2 are quite different because of the data characteristic difference between MNIST and Amazon-Computers.

The results in Tab. 6 - 7 show that, no matter which layer(s) is applied with NSR, it could always improve the accuracy compared with the vanilla baseline for both MLP-4 and GraphSAGE-2. In addition, according to our layer selection criterion, applying NSR to the layer with the biggest variance, i.e., the last layer for MLP-4 and the first layer for GraphSAGE-2, could achieve the most significant gain compared with applying NSR to other individual layers. Also, it could achieve similar accuracy compared with the best one obtained by applying NSR to multiple layers. This is the empirical evidence to demonstrate the rationality of our layer selection criterion.

5 Related Work

Regularization improves generalization ability by introducing the inductive bias based on prior knowledge. According to the type of prior knowledge, existing works can be roughly categorized into Domain Specific Regularization and Model Generic Regularization.

Domain Specific Regularization shows success in various domains including Face Recognition [46, 24, 2, 25], Knowledge Graph Completion [50, 20, 31], Graph Neural Network [3, 14, 39, 47, 27]. In Face Recognition, [2] points out the existence of external factors, such as different situations of environmental illumination, head poses, and facial expressions, which brings the challenge that faces images from the same person may have even larger differences than face images from different persons. To address specific challenges in face recognition, [24] imposes the carefully-designed regularization method which enforces the representations of face instances from the same person to be similar. In Graph Neural Network, the popular model usually suffers from the over-smoothing problem due to the six degrees of separation [18]. [3] statistically analyzes the high correlation between smoothness and the mean average distance (MAD) among node representation. Then a regularization called MADGap is proposed which punishes over smoothness by minimizing MAD.

Model Generic Regularization can be categorized into network-wise Regularization, layer-wise Regularization, and neuron-wise Regularization, according to the granularity of the studied properties. Network-wise Regularization encodes the desired property of the entire network, like the sparseness of the network and the smoothness of the mapping function between input and output. L2 Regularization [36, 21] encourages small sum squared magnitude of model weights. [13] introduces an efficient framework to minimize the norm of the input-out Jacobian matrix for noise robustness. [7] encourages model parameters to be uniformly low at some local regions of the loss function. Layer regularization [45, 36] becomes popular due to the success achieved by Batch Normalization [15]. Neuron-wise Regularization is the method with the most fine-grained granularity. Dropout [12, 44] is the typical example belonging to this category. It randomly removes neurons with a certain probability to avoid

strong co-adaption between neurons. Activation Regularization [29] adds the L2 Regularization on masked activations of neurons and Temporal Activation Regularization restricts the difference between RNN outputs at adjacent timesteps. To minimize the variance of sample variance and obtain a few distinct modes, Variance Constancy Loss [23] is applied on all neurons before the activation as regularization. Our proposed Neuron Steadiness Regularization belongs to the neuron-wise regularization that leverages information of individual neuron response distribution.

6 Conclusion and Future Work

We explore the inductive bias from the new perspective of class-dependent response distribution of individual neurons. Based on experimental observations, we articulate the Neuron Steadiness Hypothesis and propose the Neuron Steadiness Regularization that penalizes large intra-class neuron response variance. Based on the Complexity Measure, we provide a theoretical analysis of its effectiveness for improving generalization. Additionally, we conduct extensive evaluations on diverse datasets with various network architectures to demonstrate its power. Especially, we demonstrate its effectiveness on the classification task with large models like ResNet-50, and large datasets with many classes like ImageNet with 1000 classes. We systematically consider the border impact, and No risk is found.

References

[1] Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. In *Proceedings of International Conference on Learning Representations*, 2017.

[2] Jie Cai, Zibo Meng, Ahmed-Shehab Khan, Zhiyuan Li, James O'Reilly, and Yan Tong. Island loss for learning discriminative features in facial expression recognition. In *IEEE International Conference on Automatic Face & Gesture Recognition*, 2018.

[3] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3438–3445, 2020.

[4] Shuhao Cui, Shuhui Wang, Junbao Zhuo, Liang Li, Qingming Huang, and Qi Tian. Towards discriminability and diversity: Batch nuclear-norm maximization under label insufficient situations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3941–3950, 2020.

[5] David L Davies and Donald W Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2):224–227, 1979.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[7] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press Cambridge, 2016.

[9] Henry Gouk, Bernhard Pfahringer, Eibe Frank, and Michael J Cree. Maxgain: Regularisation of neural networks by constraining activation magnitudes. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 541–556. Springer, 2018.

[10] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of NuerIPS*, 2017.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[12] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[13] Judy Hoffman, Daniel A Roberts, and Sho Yaida. Robust learning with jacobian regularization. *arXiv preprint arXiv:1908.02729*, 2019.

[14] Yifan Hou, Jian Zhang, James Cheng, Kaili Ma, Richard TB Ma, Hongzhi Chen, and Ming-Chang Yang. Measuring and improving the use of graph information in graph neural networks. In *Proceedings of International Conference on Learning Representations*, 2019.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of International Conference on Machine Learning*, pages 448–456, 2015.

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations*, 2015.

[17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *In Proceedings of International Conference on Learning Representations*, 2017.

[18] Judith Kleinfeld. Could it be a big world after all? the six degrees of separation myth. *Society, April*, 12:5–2, 2002.

[19] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[20] Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In *International Conference on Machine Learning*, pages 2863–2872, 2018.

[21] Kevin J Lang and Geoffrey E Hinton. Dimensionality reduction and prior knowledge in e-set recognition. In *Advances in Neural Information Processing Systems*, pages 178–185, 1990.

[22] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[23] Etai Littwin and Lior Wolf. Regularizing by the variance of the activations' sample-variances. *Advances in Neural Information Processing Systems*, 31, 2018.

[24] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of CVPR*, 2017.

[25] Xiaofeng Liu, BVK Vijaya Kumar, Jane You, and Ping Jia. Adaptive deep metric learning for identity-aware facial expression recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 20–29, 2017.

[26] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *Proceedings of International Conference on Learning Representations*, 2018.

[27] Qingqing Long, Yiming Wang, Lun Du, Guojie Song, Yilun Jin, and Wei Lin. Hierarchical community structure preserving network embedding: A subspace approach. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 409–418, 2019.

[28] Xiaojun Ma, Junshan Wang, Hanyue Chen, and Guojie Song. Improving graph neural networks with structural adaptive receptive fields. In *Proceedings of the Web Conference 2021*, page 2438–2447, 2021.

[29] Stephen Merity, Bryan McCann, and Richard Socher. Revisiting activation regularization for language rnns. *arXiv preprint arXiv:1708.01009*, 2017.

[30] Péter Mernyei and Cătălina Cangea. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*, 2020.

[31] Pasquale Minervini, Luca Costabello, Emir Muñoz, Vít Nováček, and Pierre-Yves Vandenbussche. Regularizing knowledge graph embeddings via equivalence and inversion axioms. In *Proceedings of Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 668–683, 2017.

[32] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics*, 19(6):1236–1246, 2018.

[33] Parth Natekar and Manik Sharma. Representation based complexity measures for predicting generalization in deep learning. *arXiv preprint arXiv:2012.02775*, 2020.

[34] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

[35] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. Exploring generalization in deep learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5949–5958, 2017.

[36] David C Plaut et al. Experiments on learning by back propagation. 1986.

[37] Qing Rao and Jelena Frtunikj. Deep learning for self-driving cars: Chances and challenges. In *Proceedings of IEEE/ACM International Workshop on Software Engineering for AI in Autonomous Systems*, pages 35–38, 2018.

[38] Reza Rawassizadeh, Taylan Sen, Sunny Jung Kim, Christian Meurisch, Hamidreza Keshavarz, Max Mühlhäuser, and Michael Pazzani. Manifestation of virtual assistants and robots into daily life: Vision and challenges. *CCF Transactions on Pervasive Computing and Interaction*, 1(3):163–174, 2019.

[39] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *Proceedings of International Conference on Learning Representations*, 2020.

[40] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[41] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

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

[43] Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65(16):4265–4280, 2017.

[44] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[45] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[46] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In *Proceedings of European Conference on Computer Vision*, pages 499–515, 2016.

[47] Shuwen Yang, Guojie Song, Yilun Jin, and Lun Du. Domain adaptive classification on heterogeneous information networks. In *IJCAI*, pages 1410–1416, 2020.

[48] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of International Conference on Machine Learning*, pages 40–48, 2016.

[49] Michael R Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. Lookahead optimizer: k steps forward, 1 step back. *arXiv preprint arXiv:1907.08610*, 2019.

[50] Zhanqiu Zhang, Jianyu Cai, and Jie Wang. Duality-induced regularizer for tensor factorization based knowledge graph completion. *Proceedings of Advances in Neural Information Processing Systems*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We share the corresponding link in Appendix C
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See in Appendix C.4
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Yes, each results is calculated from five random seeds: [1, 3, 5, 7, 9]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See in Appendix B
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] See in Appendix C.4
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proof

A.1 Lemma 3.1

Lemma 3.1. For a multi-class classification problem, when (1) a deep learning model utilizing the Cross Entropy loss with an NSR regularization term on any of its intermediate layer is optimized via gradient descent and (2) the capacity of the model is sufficiently large, the Consistency of Representations Complexity Measure on this model $\mathcal{C}$ will tend to be 0. Under the same condition, when the deep learning model is only optimized by the Cross Entropy loss without an NSR regularization term, there will be infinite local minima where the complexity measure $\mathcal{C}$ will be a positive number.

Proof. Let us consider the original Cross Entropy loss first:

$$\mathcal{L}_C = -\sum_{\tau=1}^{\mathcal{N}} \log q(y = y_\tau | x_\tau; \Theta), \tag{11}$$

where $\mathcal{N}$ is the number of training samples, y_τ is the label of the τ_{th} sample, x_τ is the feature of the τ_{th} sample, Θ is the parameters of the model, and $q(y|x_\tau;\Theta)$ is the predicted probability modeled by the deep learning model. Considering a single sample, we have the Cross Entropy loss for the τ_{th} sample:

$$\mathcal{L}_C^{(\tau)} = -\log q(y = y_\tau | x_\tau; \Theta) = -\log q_\tau(y), \tag{12}$$

where $q_\tau(y) \triangleq q(y = y_\tau | x_\tau; \Theta)$ for brevity. Because the deep learning models for the classification task always have a normalization operation (e.g., Softmax) to make the final unconstrained representation be a probability form, $q_\tau(y)$ has the following form:

$$q_\tau(y) = \frac{a_y^{(\tau)}}{\sum_{j=1}^{J} a_j^{(\tau)}}, \tag{13}$$

where J is the number of classes, $\mathbf{a}^{(\tau)} = [a_1^{(\tau)}, ..., a_J^{(\tau)}]$ is the final representation for the τ-th sample. The loss can be rewritten as:

$$\mathcal{L}_C^{(\tau)} = -\log \frac{a_y^{(\tau)}}{A_\tau} \tag{14}$$

where $A_\tau = \sum_{j=1}^{J} a_j^{(\tau)}$ is the normalization factor. Since we will use gradient descent to optimize the objective function, we calculate the partial derivatives of the objective for the representations $h_j^{(\tau)}$:

$$\begin{aligned} \frac{\partial \mathcal{L}_C^{(\tau)}}{\partial h_y^{(\tau)}} &= \frac{a_y^{(\tau)}}{A_\tau} - 1 \\ \frac{\partial \mathcal{L}_C^{(\tau)}}{\partial h_j^{(\tau)}} &= \frac{a_j^{(\tau)}}{A_\tau}, \text{ for } j : j \neq y. \end{aligned} \tag{15}$$

Let the gradient be zero, then we obtain the sufficient conditions of the local minima:

$$\begin{aligned} a_j^{(\tau)} &= 0 \text{ for } j : j \neq y, \\ A_\tau &= a_y, \\ a_y^{(\tau)} &> 0, \\ \sum_j a_j^{(\tau)} &= A_\tau \end{aligned} \tag{16}$$

In that case, under the assumption that the model is sufficiently large, the converged local minima for the τ-th sample are the vectors where only the y_τ-th entry is an arbitrary positive real number and the other entries are 0.

According to the definition of the Consistency of Representations Complexity Measure Eq. (10), it is easy to see that we have an infinite number of local minima where the measures $\mathcal{S}_i$ and $\mathcal{M}_{i,j}$ for arbitrary classes i and j are finite positive numbers. Thus, the Consistency of Representations Complexity Measure $\mathcal{C}$ will be a positive number in those local minima.

Let us take our proposed NSR regularization term $\mathcal{L}_S$ (refer to Eq. (3)) into consideration. When adding NSR to k-th layer, the loss of NSR can be rewritten as:

$$\mathcal{L}_S = \sum_{i}^{J} \sum_{\tau}^{n_i} ||\mathcal{O}_{i,k}^{(\tau)} - \mu_{\mathcal{O}_{i,k}}||^2, \tag{17}$$

where n_i is the number of samples belonging to the class i, $\mathcal{O}_{i,k}^{(\tau)}$ is the k-th layer output representation of the τ-th sample belonging to class i, $\mu_{\mathcal{O}_{i,k}}$ is the cluster centroid of the k-th layer representations of class i. If we set $\mathcal{O}_{i,k}^{(\tau)}$ as optimization variables, it is easy to see that the global minima is $\mathcal{O}_{i,k}^{(\tau)} = \mu_{\mathcal{O}_{i,k}}$. It means that the intermediate representations of the same class in layer k will be the same vector. Combined with the conditions of the local minima of Cross Entropy loss, we have the optimal final representations of the overall loss, i.e.,

$$\mathbf{a}_i^{\tau} = [\ \underbrace{0 \ldots 0}_{i-1 \text{ times}}\ C_i\ \underbrace{0 \ldots 0}_{J-i \text{ times}}\ ] \text{ for } i = 1 \cdots J, \tag{18}$$

where $\mathbf{a}_i^{\tau}$ is the final representation vector of the τ-th sample belonging to the class i, and C_i is a positive real number that satisfies $C_i \neq C_j$ if $i \neq j$. It means that the final representations for the samples belonging to the same class are the same, while the representations are different for the samples belonging to different classes. It is obvious that in that case, $\mathcal{S}_i$ is zero and $\mathcal{M}_{i,j}$ for arbitrary classes i and j is a finite positive number. Thus, the Consistency of Representations Complexity Measure $\mathcal{C}$ will be 0. The Lemma is proven. □

B Hardware and Software Environment

The experiments are performed on two Linux servers (CPU: Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz, Operation system: Ubuntu 16.04.6 LTS). For GPU resources, two NVIDIA Tesla V100 cards are used for MLP and GNN and ResNet-50 on ImageNet experiments while two Titan V cards are used for other CNN models. The python libraries we use to implement our experiments are PyTorch 1.7.1 and PyG 1.6.3.

C Details of Reproduction

C.1 Open Source Code

We publish our code in Github (i.e., `https://github.com/lundu28/NSR`).

C.2 Details of Baseline Methods

In this subsection, we detailed the network architectures and the baseline methods used in our experiments.

MLP Multilayer Perceptron is a fundamental deep neural network architecture that is composed of an input layer, several hidden layers and an output layer. Five MLP models with different numbers of hidden layers are used to test the performance of neuron steadiness regularization on MNIST for handwritten digit classification. ReLU is used as an activation function in the experiments. We denote MLP-L as an L-layer MLP network. For example, MLP-4 is a 4-layer MLP with 2 hidden layers.

CNN Convolutional neural networks are widely used in the computer vision (CV) field. Two famous ones among them, ResNet-18 [11], VGG-19 [26], are chosen as the baseline methods in our paper for the image recognition task on CIFAR-10 data set. ResNet-50 [11] is chosen for the ImageNet dataset [6]. **ResNet-18** [11] is a 20-layer convolutional neural network with residual connections to avoid the problem of vanishing gradient. **ResNet-50** is similar to ResNet-18 except that it is a more expressive

50-layer convolutional neural network. **VGG-19** [42] is another classic deep convolutional neural network architecture. We utilize a variant of VGG-19 adapted for CIFAR-10 [26] classification.

GNN Graph neural networks achieve fantastic results on graph structure data. GCN [17] and GraphSAGE are two classical methods in classifying nodes on real network data. **GCN** [17] is inspired by CNN and introduces spectral graph convolutions as the layer-wise propagation on graphs. **GraphSAGE** [10] is the first inductive GNN framework that samples neighbor nodes on graphs.

The detailed settings of different architectures of MLP and GNN are shown in Tab. 8. Notice that we only show the architecture of hidden layers. The dimensions of the input and output layer rely on the input feature size and the number of categories of different datasets respectively.

Table 8: The detail architectures of MLP and GNN.

Model	Hidden layer dimension
MLP-3	[100]
MLP-4	[256, 100]
MLP-6	[256, 128, 64, 32]
MLP-8	[256, 128, 64, 32, 32, 16]
MLP-10	[256, 128, 64, 64, 32, 32, 16, 16]
GNN-2	[100]
GNN-3	[256, 100]
GNN-4	[256, 128, 64]

Table 9: The final hyperparameter settings λ for MLP and CNN

Model	MLP-3	MLP-4	MLP-6	MLP-8	MLP-10	ResNet-18	VGG-19	ResNet-50
λ	0.025	7.487	0.054	0.254	0.416	0.05	0.05	0.40

Table 10: The final hyperparamter settings λ for GNN.

Dataset	Layers	GraphSAGE	GCN
PubMed	2	0.3909	0.0367
	3	0.1436	0.1994
	4	0.5485	0.8831
Amazon-Photo	2	0.0012	0.0069
	3	0.3485	0.0855
	4	0.0001	0.0024
Amazon-Computers	2	0.0014	0.0025
	3	0.0037	0.0008
	4	0.0021	0.0028
WikiCS	2	0.0274	0.0572
	3	0.0142	0.0669
	4	0.2226	0.0015

C.3 Hyper-parameter Settings

We have only one unique hyperparameter, i.e., λ in our methods, and the detailed settings are given in the Tab. 9 and Tab. 10. The search methods and search range are given in the main text. All the other hyper-parameters are set the same as baselines.

C.4 Details of Datasets

In this section, we describe detailed information about the public available benchmark data sets used in our experiments.

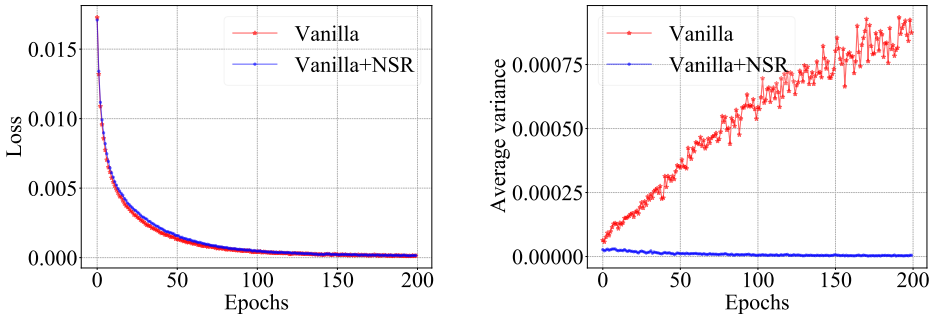
MNIST [22] is a handwritten digits dataset containing 60000 training images and 10000 testing images. It is used to examine the performances of MLP with different hidden layers. **CIFAR-10** [19] is used for ResNet-18 and VGG-19, which consists of 50,000 32×32 training color images and 10,000 testing images categorized as 10 classes. **ImageNet** is a benchmark dataset used for ResNet-50, which contains 14,197,122 annotated images with 1000 classes. For GNN, we selected four real-world graph datasets: **PubMed** [48] is a paper citation network where nodes represent documents and edges represent citation links. Both **Amazon-Photo** and **Amazon-Computers** [41] are subgraphs of Amazon co-purchase graph where nodes correspond to goods and edges connecting two nodes denote they are frequently bought together. **WikiCS** [30] is derived from Wikipedia whose nodes correspond to computer science articles and edges are hyperlinks. The license and source of these datasets are shown in Tab. 11 and Tab. 12, respectively. All the datasets are public and widely used in many research works. The collectors of the datasets are responsible for ensuring no privacy and consent issues.

Table 11: The license of datasets used in this paper.

Dataset	license
MNIST	Creative Commons Attribution-Share Alike 3.0 license
CIFAR-10	MIT license
ImageNet	MIT license
PubMed	NLM license
WikiCS	MIT license
Amazon-Photo	MIT license
Amazon-Computers	MIT license

Table 12: The source of datasets used in this paper.

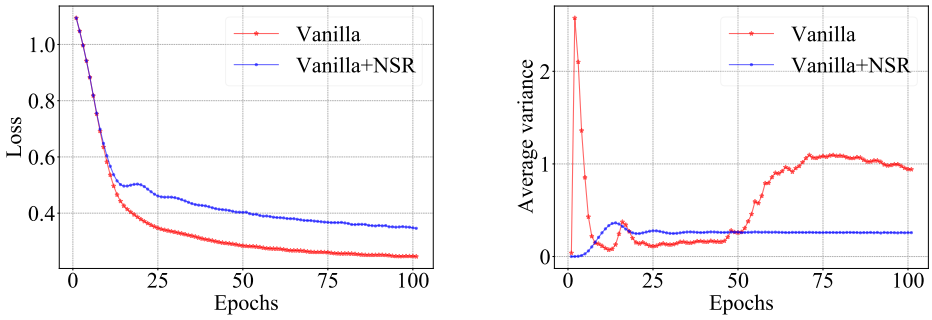
Dataset	link
MNIST	http://yann.lecun.com/exdb/mnist/
CIFAR-10	https://www.cs.toronto.edu/ kriz/cifar.html
ImageNet	https://image-net.org/index
PubMed	https://github.com/shchur/gnn-benchmark/raw/master/data/planetoid/
WikiCS	https://github.com/pmernyei/wiki-cs-dataset
Amazon-Photo	https://github.com/shchur/gnn-benchmark/raw/master/data/npz/
Amazon-Computers	https://github.com/shchur/gnn-benchmark/raw/master/data/npz/



(a) Cross entropy loss

(b) Intra-class variance

Figure 4: Training procedure of vanilla ResNet-18 and Resnet-18 with NSR on CIFAR-10. (a) and (b) illustrate the corresponding cross entropy loss and average intra-class response variance of these two models on training set.



(a) Cross entropy loss

(b) Intra-class variance

Figure 5: Training procedure of vanilla GraphSAGE-2 and GraphSAGE-2 with NSR on PubMed. (a) and (b) illustrate corresponding cross entropy loss and average intra-class response variance of these two models on the training set.

D Dynamics of GNN and CNN during training procedure

In this section, we study the training procedure of ResNet-18 (CNN) and GraphSAGE-2 (GNN) of the vanilla version and model with NSR. The cross entropy loss and the expectation of neuron intra-class response variance are demonstrated in Fig 4 and Fig 5. The results of GraphSAGE-2 and ResNet-18 show a similar tendency with MLP-4 discussed in Section 2.2. Both figures illustrate that the model with NSR shows smaller intra-class response variance which leads to high test accuracy shown in Sec 4.2.1. The overall increasing tendency of intra-class variance on GraphSAGE-2 is similar to the tendency of MLP as expected. However, there exist a few outliers. A potential reason is that the sampling process of GraphSAGE will introduce noise in the generated features which leads to perturbed responses of individual neurons during the early training phase.

E Time and space complexity analysis

In this section, we elaborate on details of the time and space consumption of NSR. The additional computation time overhead of adding NSR to the model is small. Tab. 13 shows the comparison of time consumption for one training epoch between the vanilla models and models with NSR respectively on MLP, ResNet-18 (CNN) and GraphSAGE-2 (GNN). We could see that the additional overhead is small especially for deeper networks like ResNet-18.

Table 13: Time (s) consumption of vanilla models and models with NSR on one training epoch.

Model	MLP-4	ResNet-18	GraphSAGE-2
Vanilla	2.19	15.45	0.0489
Vanilla+NSR	2.54	16.56	0.0569

The additional space consumption is also small. The space complexity for NSR can be denoted as $O(N \times J \times M)$, where N is the number of neurons in a layer, J is the number of categories and M is the length of memory queue. According to Fig. 6, the performance of MLP model is robust with the length of memory queue and a small memory queue length can still lead to a competing performance. So the length of memory queue can be set as a small constant. All experiments set the length of memory queue as 5. Therefore, the computation space complexity can be directly denoted as $O(N \times J)$. We show that this space complexity is more negligible than that of vanilla model, including all model weights and intermediate outputs of every layer. Using MLP as an example, the space complexity of weights of a layer can be represented as $O(N_{in} \times N)$, where N_{in} represents the input feature dimension. The space complexity of output for a layer can be denoted as $O(N \times B)$, where B denotes batch size. Since the number of categories J is usually smaller than the number

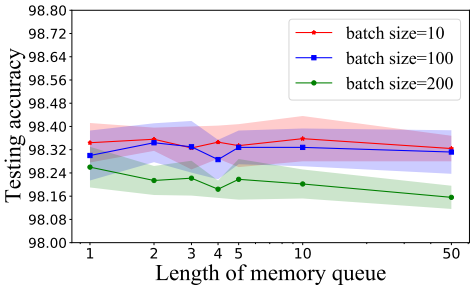


Figure 6: The testing accuracy of MLP-4 with NSR on MNIST with different memory queue length. The shadow indicates the standard deviation of the results obtained from five different runs with five different random seeds.

of neurons N_{in} and batch size B, the space complexity of NSR is smaller than that of weights and output for a layer. The space used for NSR is usually much smaller than the space used for storing weights and layer output.