# Partial Output Norm: Mitigating the Model Output Blow-up Effect of Cross Entropy Loss

**Anonymous authors**
Paper under double-blind review

## Abstract

Cross entropy loss is a very popular optimization objective and has been successfully applied for diverse classification tasks. The discrepancy between Cross Entropy (CE) objective and real classification target is not fully studied because researchers usually think such discrepancy is a must-pay price for a differentiable objective that can be optimized through gradient-based methods. In this paper, we carefully study such discrepancy and find out this discrepancy leads to the side effect that the model outputs have a certain useless growth tendency when the classification result is correct. We call such side effects as “model output blow-up effect”. This effect distracts CE objective from real effective updates, which brings a negative influence on the model training. To mitigate such a side effect, we introduce a partial normalization layer for regularizing model output to reduce its useless growth tendency. We further provide the theoretical analysis of our finds and our approaches. The experiment results show that the proposed partial normalization layer improves the model training, and it could be combined with other methods like weight decay to achieve big additional performance gain.

## 1 Introduction

Cross-Entropy (CE) objective is very popular for classification tasks. The deep neural networks (DNNs) trained with the cross-entropy objective have achieved state-of-the-art performance on various classification tasks in many different domains (Heaton (2018)). However, because the CE objective approximates sample label by estimating its probability, such approximation make it not exactly present the classification accuracy of the model. Therefore, minimizing CE loss does not necessarily decrease the classification error, which means there ’s discrepancy between cross entropy loss and model classification error. For example, for a binary classification model with two output nodes, given the classification target as [1, 0] for a sample, the model output A [1.8, 1.4] has smaller CE loss than output B [3.6, 2.8], i.e., 0.513 vs 0.371, while the classification error of A and B is the same (i.e., both are 0 as they all correctly classify this sample). People usually regard such discrepancy as a must-pay price because CE objective provides a good approximation about the classification error with a differentiable function, which can be optimized using gradients based methods.

In this paper, we carefully analyze this discrepancy between the CE objective and model classification and find out such discrepancy lead to the side effect that the model output have useless growth tendency when the classification result is correct. Take the example in the first paragraph as illustration, after proportional increase different dimensions of model output (e.g., [1.8, 1.4] * 2 = [3.6, 2.8]), the bigger model output (i.e., model output B) has smaller CE loss than A. Obviously, just proportional increasing different dimension of model output does not have any real help for classification tasks. Thus the CE objective guides neural networks training not only to better classification accuracy but also to such useless growth of model outputs. We call such side effect as ”Model Output Blow-up effect”. Such ”Model Output Blow-up effect” distracts CE objective from real effective update, which brings the negative influence on the model training.

To solve such problem, we here introduce a partial normalization layer named ”Outputnorm”, which takes the output logit of the model (denoted as original logit) as the layer input and transform it to the layer output (denoted as final logit). Then, such final logit, instead of the original logit, is used for calculating cross entropy loss. The basic idea of the partial normalization layer is simple: When the length of the original logit is bigger than a certain threshold, the partial normalizaton layer will scaling down the original logit linearly to obtain the final logit for meeting the length threshold. Otherwise it just keep the original logit unchanged as the final logit. Note that, it is important to keep the original logit unchanged when its length is small because scaling-up small original logit will make the final logit lie closer to the saturation region of the cross-entropy function which makes gradients smaller and slows down the training.

There is additional technique (i.e., the layer input stability regularization ) added in the partial normalization layer. Through such layer input stability regularization, the applied scaling coefficient of the partial normalization layer becomes more stable so as to benefit training process. The details and the theoretical analysis are introduced in the Section 3.

Such partial normalization layer can mitigate the ”Model Output Blow-up effect”. The intuition is that, because those big outputs (i.e., logits with bigger magnitude than the threshold) are scaled down before calculating cross entropy loss, the model does not have the tendency to uselessly grow the output logit (i.e., proportional increase different dimensions of model output logit). We provide both evaluation results and in-depth analysis to show its effectiveness.

There are some related works, such as weight decay (Krogh & Hertz (1991); Bos & Chug (1996)), that could also have influence on the model outputs. While, our work in this paper have a completely different motivated point, i.e., mitigating ”Model Output Blow-up effect” in our work vs reducing model complexity in weight decay regularization. Also, the derived solution is quite different. That is, we apply the normalization to those big output logits which is sample dependent, while weight decay directly operates on model parameters which has the same effects on all samples. In fact, we find that our methods and weight decay actually can complement each other well. The experiments on different datasets show that our methods can get big additional gain when combining with weight decay.

In sum, our paper has the following key contributions:

- We analysis the discrepancy between CE objective and model classification and figure out the side effect that the model output lengths have useless growth tendency to reduce cross entropy loss. We call such side effects as ”Model Output Blow-up effect”, which has negative influence to model training performance.
- We introduce the partial normalization layer, which can effectively mitigate such ”Model Output Blow-up effect” to improve the training performance. We provide the theoretical analysis and the in-depth experimental analysis on such approach.
- Our evaluations on different datasets with different model architectures show that our proposed method are effective themselves and can get big additional improvement when combining with weight decay.

## 2 Model Output Blow-up Effect

In this section, we introduce the identified Model Output Blow-up effect caused by Cross Entropy loss and analyze its influence on model training performance.

### 2.1 Background

Calculating the cross entropy loss contains two steps: first, the output logits of the network are feed into the Softmax layer to obtain the normalized outputs (i.e., summation equal to 1), which can be considered as an estimated output probability. The softmax layer formula is shown as equation 1. Among it, $\mathbf{x} = [x_1, x_2, ..., x_n]$ present the model output logits, $x_j \in \mathbb{R}$ present the output logit value of the $j^{th}$ class, $n$ is the total number of classes. Then, given the true label class index as $T$, such normalized output $\mathbf{x}_S$ is used to calculate the cross entropy loss to measure the classification loss, which is defined as equation 2.

$$\mathbf{x}_S = \text{Softmax}(\mathbf{x}) = \frac{e^{\mathbf{x}}}{\sum_{j=1}^{N} e^{x_j}} \tag{1}$$

$$H(\mathbf{x}_S, T) = -\log \frac{e^{x_T}}{\sum_{j=1}^{N} e^{x_j}} \tag{2}$$

For “Cross Entropy loss” appears below in this paper, it actually represents the Softmax Cross Entropy (SCE) loss, which contains the aforementioned two steps to calculate the loss from the output logits of the network.

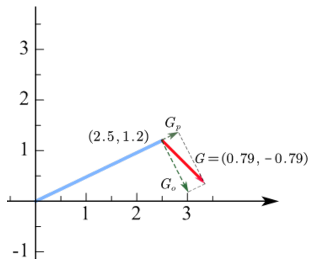

Figure 1: Gradient Sketch for node (2.5, 1.2) using CE loss for gradient calculating

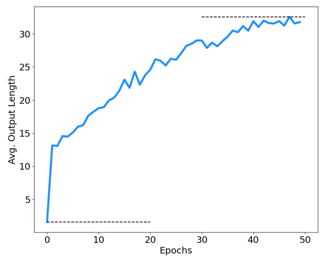

Figure 2: Avg. output length of MLP on MNIST dataset

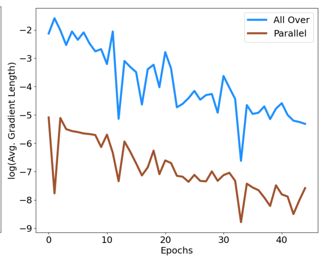

Figure 3: The natural logarithm of Overall gradient and Parallel gradient of MLP on MNIST dataset

## 2.2 Model Output Blow-up effect

We use an illustrative example shown in figure 1 to explain the model output blow-up effect of cross entropy loss. In more details, it is a simple binary classification task, x and y axis represent the output logits of the first and the second class, respectively. Here, assuming the detailed binary output logits $\mathbf{x}$ is [2.5, 1.2] and the true label is [1, 0], which means the model classifies the sample correctly. The calculated CE loss is 1.54 and the back-propagated gradient $G$ on the output logits is [0.79, -0.79]. We use the blue arrow line in the figure 1 presenting the output logits vector $\mathbf{x}$, red arrow line presenting the gradient vector $G$. From figure 1, we find that $G$ is not orthogonal to $\mathbf{x}$, we then decompose $G$ into orthogonal component $Go$ and parallel component $Gp$ to the output vector $\mathbf{x}$. Although both $Go$ and $Gp$ can reduce the cross entropy loss, only the update of $Go$ makes the prediction closer to the true label while $Gp$ does not have real help for training. It is because The orthogonal component $Go$ could change the model output direction, while the parallel component $Gp$ only proportionally enlarge different dimension of the output logits vector without changing its direction. Intuitively, such length-only-change on the output logits is equal to increasing model parameters linearly, which is useful neither for training nor for generalization.

From the illustrative example, we could see that, when the model make a correct prediction, the gradient back-propagated by CE loss on the model output logits has the component $Gp$ on enlarging the output length without changing its direction. This gradient component $Gp$ makes the model output grow and doesn’t contribute to better classification performance. We name this side effect as ”Model Output Blow-up effect”. Such effect may distract CE objective from real effective update, which brings the negative influence on the model.

In addition to the illustrative example, here are the empirical observations relevant to such effect. We train the MultiLayer Perception (MLP) on the MNIST dataset, and plot the training dynamics regarding to the output length and the gradient magnitude in Fig. 2 and Fig. 3, respectively. In more details, figure 2 shows that the average output length of the model grows up from 1.59 at the beginning to 31.75 at the end (i.e., 50 epochs). figure 3 shows the log magnitude of the parallel component $Gp$ and the whole gradient $G$ with red line and blue line, respectively. Interestingly, $Gp$ and $G$ share very similar shape, which seems to show there is always certain portion of gradient spent on only enlarging model output length without changing their direction. In the next section, we will give the theoretical proof on it.

### 2.3 Theoretical Definition and Analysis

We give the illustrative example and the empirical observations to show the model output blow-up effect in last section. Here, we give the formal definition of Model Output Blow-up Effect:

**Definition 2.1** (***Model Output Blow-up Effect***)**.** Model Output Blow-up Effect is a trend of continuously increasing the model output length caused by the optimization objectives during the optimization process, where the optimization trend of increasing the output length means that the directional derivative on the output length is always less than 0. Formally, the condition can be defined as:

$$\frac{\partial \mathcal{L}(\mathbf{x})}{\partial \|\mathbf{x}\|} < 0, \quad \text{for } \forall \mathbf{x} \in \mathcal{O}, \tag{3}$$

where $\mathcal{L}$ is the loss function, $\mathbf{x}$ is the model output logits, $\|\cdot\|$ is the l2-norm operation, and $\mathcal{O}$ is the constraint range of $\mathbf{x}$.

With the definition of Model Output Blow-up Effect, we have the following Lemma:

**Lemma 2.2.** *For a classification problem, when the applied deep learning model is optimized by the Cross Entropy loss using gradient-based optimizer, then it will lead to a Model Ouput Blow-up Effect.*

*Proof.* Let $\mathcal{L}$ be the Cross Entropy loss and $\mathbf{x}$ be the model output for an arbitrary sample, then we can calculate the directional derivative on the model output length:

$$\frac{\partial \mathcal{L}(\mathbf{x})}{\partial \|\mathbf{x}\|} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}(\mathbf{x})}{\partial x_i} \cdot \frac{\partial x_i}{\partial \|\mathbf{x}\|} = \sum_{i=1, i\neq m}^{N} \left( \frac{(x_i - x_m) \cdot e^{x_j}}{\left(\sum_{j=1}^{N} e^{x_i}\right) \cdot \|\mathbf{x}\|_2} \right) \tag{4}$$
$$\text{where} \quad x_m = \max(x_1, x_2, ..., x_N).$$

$N$ is the output dimension of $\mathbf{x}$ (i.e., the number of categories for a classification problem), and $m$ is the index of the maximum element of $\mathbf{x}$. It is obvious that $\frac{\partial \mathcal{L}(\mathbf{x})}{\partial \|\mathbf{x}\|}$ is always less than zero, so the Lemma is proved. □

The Lemma means CE loss back-propagated gradients $G$ always have positive parallel component $G_p$, which enlarges model output logits and causes Model Output Blow-up Effect.

## 3 New Regularization Layer: P-OutputNorm

We propose a new regularization layer, named P-OutputNorm, to mitigate the side effect of Cross Entropy Loss, i.e., Model Output Blow-up Effect. This layer includes two main components, a partial norm operation and a input stabilizer, which will be introduced in details together with theoretical analysis in the following subsections.

### 3.1 Partial Norm Operation

The proposed Partial Norm Operation is a sample-wise operator that takes the model output logits (i.e., the network output before Softmax) as input and normalize the logits by the following formulas:

$$\hat{\mathbf{x}} := \Gamma(\mathbf{x}) = \begin{cases} \mathbf{x}, & \text{if } \|\mathbf{x}\| \leq \delta, \\ \frac{\delta}{\|\mathbf{x}\|} \cdot \mathbf{x}, & \text{if } \|\mathbf{x}\| > \delta, \end{cases} \tag{5}$$

where $\mathbf{x} \in \mathbb{R}^N$ is the output logits (named original logits), $N$ is the number of categories of the classification task, $\hat{\mathbf{x}} \in \mathbb{R}^N$ is the partially normalized logits, $\delta$ is a threshold, $\Gamma$ is the function of Partial Norm Operation, and $\|\cdot\|$ is the L2-norm. Intuitively, when the length of $\mathbf{x}$ is larger than the given threshold $\delta$, the partial norm operation will scaling the length of $\mathbf{x}$ down linearly to be the threshold $\delta$. Otherwise it will keep the original logits unchanged. It is important to keep the original logit unchanged when its length is small because scaling-up small original logit will make the final logit lie closer to the saturation region of the cross-entropy function which makes gradients smaller and slows down the training, which is verified by our empirical analysis.

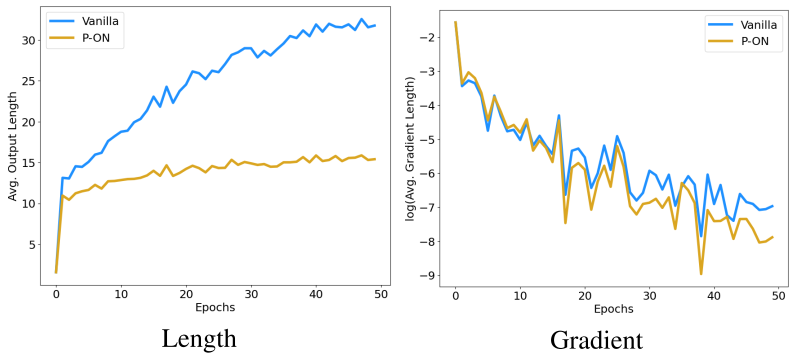

Figure 5: The natural logarithm of Avg. parameter gradient of MLP on MNIST comparing the baseline and our normalization

By using the partial norm operation, we can eliminate the Model Output Blow-up Effect with the following theoretical guarantee:

**Lemma 3.1.** *For a classification problem, when the applied deep learning model is optimized by the Cross Entropy loss with our proposed Partial Norm Operation, then there will not be a Model Ouput Blow-up Effect.*

*Proof.* Let $\mathcal{L}$ be the Cross Entropy loss and $\mathbf{x}$ be the model output (before the P-OutputNorm layer) for an arbitrary sample. When the length of $\mathbf{x}$ is smaller than the threshold $\delta$, than the optimization process is the same to the original one without the P-OutputNorm layer. According to the Lemma 2.2, the length of $\mathbf{x}$ will continuously increase until meeting the threshold $\delta$.

Then we can only analyze the case that the length of $\mathbf{x}$ is larger than the threshold $\delta$. The directional derivative on the model output length can be calculated as:

$$\begin{aligned}\frac{\partial \mathcal{L}(\hat{\mathbf{x}})}{\partial l} = \frac{\partial \mathcal{L}\left(\Gamma(\mathbf{x})\right)}{l} &= \lim_{\Delta l \to 0} \frac{\mathcal{L}(\Gamma\left(\mathbf{x}+\Delta l\right)) - \mathcal{L}(\Gamma\left(\mathbf{x}\right))}{\Delta l} \\ &= \lim_{\Delta l \to 0} \frac{\mathcal{L}(\frac{\delta}{\|\mathbf{x}\|}\cdot \mathbf{x}) - \mathcal{L}(\frac{\delta}{\|\mathbf{x}\|}\cdot \mathbf{x})}{\Delta l} = 0,\end{aligned} \tag{6}$$

where $l = \|\mathbf{x}\|$ is the length of $x$. The Lemma is proved. □

## 3.2 The Input Stability Regularization

The proposed Partial Norm Operation is a sample-level operation that only leverages the information of a single data sample while not considering the effect of the relationship between different samples. Since the common setting of the deep neural networks is mini-batch training, such relationships will significantly influence the stability of the training procedure. In this subsection, we will first theoretically analyze how the length of the model output affects the training stability and then propose a stability regularization that stabilizes the input of our proposed P-OutputNorm layer.

Through theoretical analysis, we find that the difference in the model output lengths between data samples will lead to the difference in optimization intensity for the different samples when applying partial norm operation. If the model output lengths varies considerably for different samples within a batch, the training process will be unstable. Formally, we have the following Lemma:

**Lemma 3.2.** *For the training process of a neural network with the partial norm operation, the gradient for the model logit of a sample is inversely proportional to the length of the model logit, i.e.,*

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}) \propto \frac{1}{\|\mathbf{x}\|}, \tag{7}$$

*where $\mathcal{L}$ is the loss function, and $\mathbf{x}$ is the model logit of a sample.*

*Proof.* We can directly calculate the partial derivative of loss to the model logit in the $j$-th dimension:

$$\frac{\partial \mathcal{L}(\mathbf{x})}{\partial x_j} = \frac{\sum_{i=1}^{n} e^{\frac{\delta \cdot x_i}{\|\mathbf{x}\|}}}{e^{\frac{\delta \cdot x_m}{\|\mathbf{x}\|}}} \cdot \frac{\delta \cdot e^{\frac{\delta \cdot x_m}{\|\mathbf{x}\|}} \cdot e^{\frac{\delta \cdot x_j}{\|\mathbf{x}\|}} \cdot (\frac{x_m \cdot x_j}{\|\mathbf{x}\|^2} - 1)}{\left(\sum_{i=1}^{n} e^{\frac{x_i}{\|\mathbf{x}\|}}\right)^2} \cdot \frac{1}{\|\mathbf{x}\|}. \tag{8}$$

It is obvious that the right two terms in the above equation is independent of the length of $\mathbf{x}$, since each element of $\mathbf{x}$ is divided by the length of $\mathbf{x}$. Then we can obtain the expression of the gradient:

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}) = \frac{1}{\|\mathbf{x}\|} \cdot \Psi(\tilde{\mathbf{x}}), \quad \text{where} \quad \tilde{\mathbf{x}} = \frac{\mathbf{x}}{\|\mathbf{x}\|}. \tag{9}$$

$\Psi$ is a function that is independent of the length of $\mathbf{x}$. The Lemma is proved. $\square$

From the Lemma, we can see that the variance of the scale of the gradients of different samples is linearly related to the variance of the reciprocal of the logit length for the samples.

On the basis of above, we introduce a layer input stability regularization contained in the partial normalization layer. The regularization calculates the length of output logits (the input of P-OutputNorm layer) for every sample and the standard deviation of the lengths, which can be formulated as:

$$\mathcal{L}_S = \alpha \cdot SD(\|\mathbf{x}\|_2) = \alpha \cdot \left(\mathbb{E}\left[(\|\mathbf{x}\|_2 - \mathbb{E}\left[\|\mathbf{x}\|_2\right])^2\right]\right)^{\frac{1}{2}} \tag{10}$$

where $\mathcal{L}_s$ represents the input stability regularization term for the partial normalization layer. $\alpha$ is the hyper-parameter. Adding the input stability regularization term to the main training target (Cross Entropy loss), the final regularized loss function can be written as:

$$\mathcal{L} = \mathcal{L}_C + \mathcal{L}_S \tag{11}$$

where the $\mathcal{L}_C$ represents the Cross Entropy loss.

## 4 EXPERIMENTS

In this section, we conduct experiments on a variety of datasets to examine the performance of our P-OutputNorm layer on three extensively used neural network architectures.

### 4.1 NETWORK ARCHITECTURES AND DATASETS

We arrange our experiments on three widely used network architectures which are Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), and Graph Neural Network (GNN). For CNN we use ResNet-18, ResNet-34 and ResNet-50 (He et al. (2016)) as our baseline, for GNN we use GCN (Kipf & Welling (2016) )and GAT (Velickovic et al. (2017)) with 2 layers each. The MLP we use in our experiments has three different layers which use ReLU (Nair & Hinton (2010)) as the active function.

As for the benchmark datasets used in our experiments, we apply MLP on the MNIST (Deng (2012)) dataset which contains 60,000 handwriting images for training and 10,000 for testing. CNNs are applied on CIFAR-10, CIFAR-100 and ImageNet datasets. CIFAR-10 (Krizhevsky et al. (2009)) contains 60,000 images with 10 categories, each has 5,000 training images and 1,000 testing images while CIFAR-100 (Krizhevsky et al. (2009)) consists of 60,000 images with 100 categories which have 500 training images and 100 testing images each. ImageNet (Deng et al. (2009)) contains 14,197,122 images which belongs to 1000 categories. On Graph Neural Network we apply PubMed, CiteSeer and Cora datasets. Cora (Sen et al. (2008)), CiteSeer (Yang et al. (2016)) and PubMed (Yang et al. (2016)) are citation network datasets in which nodes represent papers and edges represent citaions. PubMed has 19,717 nodes and 44,388 edges while CiteSeer has 3,327 nodes and 4,732 edges and Cora has 2708 nodes and 5433 edges.

| Dataset<br>Applied Model | MNIST<br>MLP-3 | CIFAR-10<br>ResNet-18 | CIFAR-100<br>ResNet-34 | ImageNet<br>ResNet-50 |
|---|---|---|---|---|
| Vanilla (%) | 1.64 ± 0.10 | 5.74 ± 0.09 | 25.47 ± 0.12 | 7.27 ± 0.17 |
| Vanilla + P-ON (%) | **1.14 ± 0.08** | **5.00 ± 0.5** | **23.85 ± 0.10** | **6.86 ± 0.03** |
| Gain | 30.48% | 12.89% | 6.36% | 5.64% |

Table 1: Error rate of applying our P-OutputNorm (P-ON) on MLP for MNIST, ResNet-18 for CIFAR-10, ResNet-34 for CIFAR-100 and top-5 error rate on ResNet-50 for ImageNet. Notably, We utilize the widely used training methods such as weight decay in the experiments.

| Dataset | Cora | | PubMed | |
|---|---|---|---|---|
| Applied Model | GCN | GAT | GCN | GAT |
| Vanilla (%) | 23.42 ± 0.02 | 26.70 ± 0.01 | 26.28 ± 0.02 | 25.50 ± 0.02 |
| Vanilla+P-ON (%) | **22.98 ± 0.01** | **22.48 ± 0.01** | **24.32 ± 0.01** | **24.56 ± 0.01** |
| Gain | 1.87% | 13.93% | 7.45% | 3.68% |

Table 2: Error rate of applying our P-OutputNorm (P-ON) on GCN and GAT for Cora and PubMed dataset. Notably, We utilize the widely used training methods as mentioned in Table 1.

| Dataset<br>Applied Model | MNIST<br>MLP-3 | CIFAR-10<br>ResNet-18 |
|---|---|---|
| Vanilla (%) | 98.16 | 92.74 |
| Vanilla + wd (%) | 98.36 | 94.26 |
| Vanilla + P-ON-Norm(%) | 98.55 | 93.54 |
| Vanilla + P-ON-Reg (%) | 98.55 | 93.15 |
| Vanilla + P-ON-Norm + wd(%) | **98.75** | **94.85** |
| Vanilla + P-ON-Reg + wd (%) | <u>98.63</u> | <u>94.83</u> |

Table 3: Error rate of applying our two methods in P-ON (regarding the normalizing operation as ”P-ON-Norm” and the regularizing operation as ”P-ON-Reg”) and combine them with weight decay on MLP and ResNet-18 for MNIST and CIFAR-10. **Notably**, the Vanilla model here doesn't apply weight decay method as the beyond tables does. The weight decay method is regarded as ”wd” in this table.

## 4.2 EXPERIMENT SETTINGS

For ResNet-50 on ImageNet, we follow the official implementation provided by torchversion library [1]. For graph datasets, we randomly split them into training, validation and testing set with ratio 6:2:2. We utilize different optimizer methods on the models. For MLP, we use SGD (Ruder (2016)) optimizing algorithm in training process. For CNN, we use Momentum (Ruder (2016)) optimizing algorithm while for GNN we apply Adam (Kingma & Ba (2014)). Training epochs are set to 100 on MLP and GNN as 200 for CNN. While using our method, we tune the hyper-parameter $\delta$ and $\alpha$ as mention before like most of the regularization methods.

## 4.3 EXPERIMENT RESULTS OF OUR P-OUTPUTNORM LAYER

In this subsection, we analyze the effectiveness of our P-OutputNorm layer. The results of our experiments are presented in Table 1 and Table 2. For CNN and MLP, we observe that our P-OutputNorm layer can improve the training performance in all datasets. As it is shown, the error rates of different training cases is reduced by at least 5.64% on ImageNet and at most 30.48% on MNIST.

For GNN models, we demonstrate the performance of our P-OutputNorm layer on GCN and GAT models for Cora and PubMed datasets. As it is shown in Table 2, for each training cases,

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

our P-OutputNorm layer achieve better training performance. On GAT for Cora dataset, our P-OutputNorm layer reduces 13.93% error rate to the model. GCN model achieves the average improvement of 4.66%.

Notably, we utilize the typical training methods such as weight decay and learning rate decay to our experiments. We further discover that our P-OutputNorm layer actually have extra benefits to ulteriorly enhance model training performances when combining with weight decay as we presents here. In subsection 4.4, we will demonstrate more evidence about this. It is also worth mentioned that the ResNets have already adopted Batch Normalization and Dropout regularization as widely used in the models. The success of Our P-OutputNorm layer proves that controling the growth gradients of Cross Entropy loss can improve the model training performance.

## 4.4 ABLATION STUDY

In this subsection, we decouple the two methods in our P-OutpuNorm probe into the performance of each methods. Furthermore, as it is mentioned in Section 1, our methods and the widely used weight decay are relevant as they both control the ourput length of the model. However, the motivated point of our work in this paper and weight decay is completely different while weight decay and our methods take different operations. To show that weight decay and our methods are complementary, we further probe into the performance of our methods combining with weight decay. For clearer explanation, we regard the normalizing operation in our P-ON as P-ON-Norm and the regularizing operation as P-ON-Reg.

As the results are shown in the Table 3, we could figure out that the two single methods in our P-OutputNorm can also outperform the model generalization ability. The P-ON-Norm reduces the error rates of Vanilla models by 21.19% on MLP and 11.02% on ResNet-18 and the P-ON-Reg reduces them by 21.19% on MLP and 5.65% on ResNet-18 while weight decay does 10.86% and 20.93%. While combining with weight decay, the P-ON-Norm achieve 32.06% reducing on MLP and 29.06% on ResNet-18 while the P-ON-Reg does 25.54% and 28.79%. The results show that our single training methods can achieve improvements on model and achieve better combining with each other and weight decay. Our methods and weight decay are actually complementary.

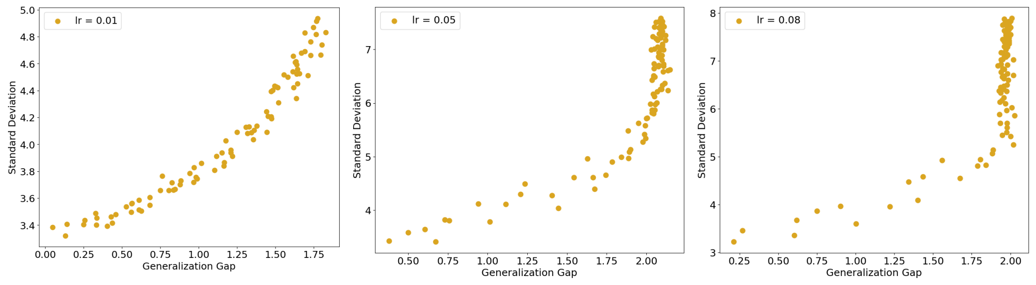

Figure 8: The relationship between model generalization gap and model output length standard deviation of MLP on MNIST for different learning rate 0.01, 0.05 and 0.08

## 4.5 THE EFFECT OF THE INPUT STABILITY REGULARIZATION

In the above subsection we observe that utilize the single method of our P-ON helps improve the model training performance. In section 3 we anylize why using P-ON-Norm can improve the model performance, we here anylize the effect of P-ON-Reg itself to the model.

As it is shown in Fig. 4.4, we could find that by defining the accuracy gap of model between training set and test set as the generalization gap, the standard deviation of the model output logits (the input of partial P-OutputNorm layer) lengths corresponds strongly to the model generalization gap. Thus we propose a mechanism of the P-ON-Reg that by using the regularization, the generalization gap of model can be reduced through training process.

## 5 Related Work

Cross Entropy loss is one of the most popular optimization objective used in the training of neural networks. CE loss function originates from information theory and has its theoretical explanation. It is used to measure the distance between the model prediction result and the true target. Although it is a very popular objective, there are also several papers reporting the weaknesses of Cross Entropy loss. One of the reported weaknesses which attracts much attention is the unsatisfactory performance on dealing with noisy data (Ghosh et al. (2017)). As Ghosh et al. (2017); Patrini et al. (2017) mentioned in their study, the Cross Entropy loss can lead the model to over-fitting on noisy labels which makes the model has poor generalization performance. To mitigate this problem, several altered loss functions were proposed. The work of Generalized Cross Entropy (GCE) (Zhang & Sabuncu (2018)) uses the negative Box-Cox transformation strategy and a hyperparameter to balance between Mean Absolute Error (MAE) and Cross Entropy for better robustness to label noise, while Symmetric Cross Entropy (Wang et al. (2019)) utilizes similar strategy as it combines Cross Entropy and Reverse Cross Entropy. All these works focus on dealing with noise label to improve the generalization ability, while our work is motivated by the identified side effect (i.e., the model output blow-up effect) of CE loss on training, which are quite different. Accordingly, the proposed techniques are quite different. All these works design the new objective function, while we introduce the additional network layer to mitigate the side effect.

Weight decay is a relevant technique that decreases the model parameters a little bit in every update to penalize the increase of the magnitude of the model parameters. Different from our work's motivation, i.e., mitigating the identified side effect of CE loss on training, weight decay is usually considered as the techniques to constrain the model complexity so as to improve the generalization ability. Therefore, the motivation of the two techniques are quite different. In addition, although penalizing the magnitude of the model parameters in weight decay can also regularize the model output to certain extent, its applied regularization technique and the resulting effect are different from ours. In more details, we propose the partial normalization, which is applied only to those big output logits instead of all samples, and the normalization coefficient is related to sample itself. In other words, our technique is sample dependent, while weight decay directly operates on model parameters which is sample independent and has the same effects on all samples.

## 6 Conclusion

In this paper, we identify and formally define the side effect, i.e., model output blow-up effect, of cross entropy loss, and theoretically prove its existence on arbitrary conditions. We further propose a partial normalization layer P-OutputNorm to mitigate such side effects, which contains the partial normalization operation and the input stability regularization. We provide theoretical proof on its effectiveness. Also, the evaluations using different networks on different benchmark datasets show the promising performance of the proposed technique.

## References

Siegfried Bos and E Chung. Using weight decay to optimize the generalization ability of a perceptron. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pp. 241–246. IEEE, 1996.

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*, pp. 248–255. Ieee, 2009.

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.

Aritra Ghosh, Himanshu Kumar, and P Shanti Sastry. Robust loss functions under label noise for deep neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

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*, pp. 770–778, 2016.

Jeff Heaton. Ian goodfellow, yoshua bengio, and aaron courville: Deep learning, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

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

Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. Making deep neural networks robust to label noise: A loss correction approach. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1944–1952, 2017.

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

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *stat*, 1050:20, 2017.

Yisen Wang, Xingjun Ma, Zaiyi Chen, Yuan Luo, Jinfeng Yi, and James Bailey. Symmetric cross entropy for robust learning with noisy labels. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 322–330, 2019.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pp. 40–48. PMLR, 2016.

Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31, 2018.