Stochastic Loss Function

Qingliang Liu, Jinmei Lai
State Key Lab of ASIC and System, School of Microelectronics, Fudan University, Shanghai, China
{qlliu17, jmlai}@fudan.edu.cn

Abstract

Training deep neural networks is inherently subject to the predefined and fixed loss functions during optimizing. To improve learning efficiency, we develop *Stochastic Loss Function* (SLF) to dynamically and automatically generating appropriate gradients to train deep networks in the same round of back-propagation, while maintaining the completeness and differentiability of the training pipeline. In SLF, a generic loss function is formulated as a joint optimization problem of network weights and loss parameters. In order to guarantee the requisite efficiency, gradients with the respect to the generic differentiable loss are leveraged for selecting loss function and optimizing network weights. Extensive experiments on a variety of popular datasets strongly demonstrate that SLF is capable of obtaining appropriate gradients at different stages during training, and can significantly improve the performance of various deep models on real world tasks including classification, clustering, regression, neural machine translation, and objection detection.

In recent years, deep neural networks (DNNs) have achieved significant success in a wide variety of pattern recognition and machine learning fields, ranging from unsupervised learning to supervised learning (Wu et al. 2018; Li et al. 2019). Given a predefined and fixed loss function, the back-propagation algorithm provides an effective and efficient way to calculate gradients to minimize the loss on the training dataset. Relying on the gradients with optimization methods, DNNs are gradually in a position to handle various tasks, such as image classification and clustering, the object detection, and amongst others (He et al. 2016; Ren et al. 2017; Chang et al. 2017; 2019).

Although the achievements in the literature are brilliant, it is unclear whether the gradients generated from the utilized loss function are the most appropriate. Intuitively, training a network can be recast as a process of teaching a student to learn a task, *i.e.*, adjusting parameters with gradients yielded from a loss function. In order to learn better, effective teaching involves progressively and dynamically refining the teaching strategy based on reflection and feedback from students. That is, the most important thing in optimizing deep networks is computing gradients of network weights for minimizing loss functions.

For training a better deep models, lots of efforts have been made to generate appropriate gradients for a better learning (Narasimhan 2018; Huang et al. 2019; Grabocka, Scholz, and Schmidt-Thieme 2019; Streeter 2019; Chebotar et al. 2019; Shu et al. 2019), *e.g.*, handcrafted or adaptive loss functions (de Boer et al. 2005; Nguyen and Sanner 2013; Liu et al. 2016; Barron 2019), curriculum learning (Bengio et al. 2009; Jiang et al. 2014), teacher-student model (Wu et al. 2018; Agarwal et al. 2019), and so on. Despite the observable advances, these methods still suffer from the optimizing way of the loss, *i.e.*, fixed or iterative optimization between network weights and loss parameters. This coarse optimizing manner indicates that network weights and loss parameters can not be adjusted together. As a result, such violent decoupling inherently limits the performance of these methods.

In order to obtain more accurate gradients to accelerate the learning process or improve the learning performance, we develop stochastic loss function to dynamically and automatically generating appropriate loss functions to train deep networks in same round of back-propagation. Different from typical learning settings in which the loss function of a machine learning model is predefined and fixed, in our framework, the loss function of a machine learning model is dynamically selected during training in a differentiable manner. In the same round of learning, the parameters in the network are optimized with the standard back-propagation.

To sum up, the main contributions of this work are:

- A stochastic loss function is developed to dynamically generate appropriate gradients, which are in a position to automatically adjusted with the performance on the validation set in a differentiable manner.
- Benefiting from the differentiability of the operations in stochastic loss function, it can be seamlessly optimized with the standard back-propagation algorithm in the same round of back-propagation for the requisite efficiency.
- Extensive experiments demonstrate that our stochastic loss function can provide instructive gradients to train networks in a differentiable manner, with both effectiveness and efficiency.

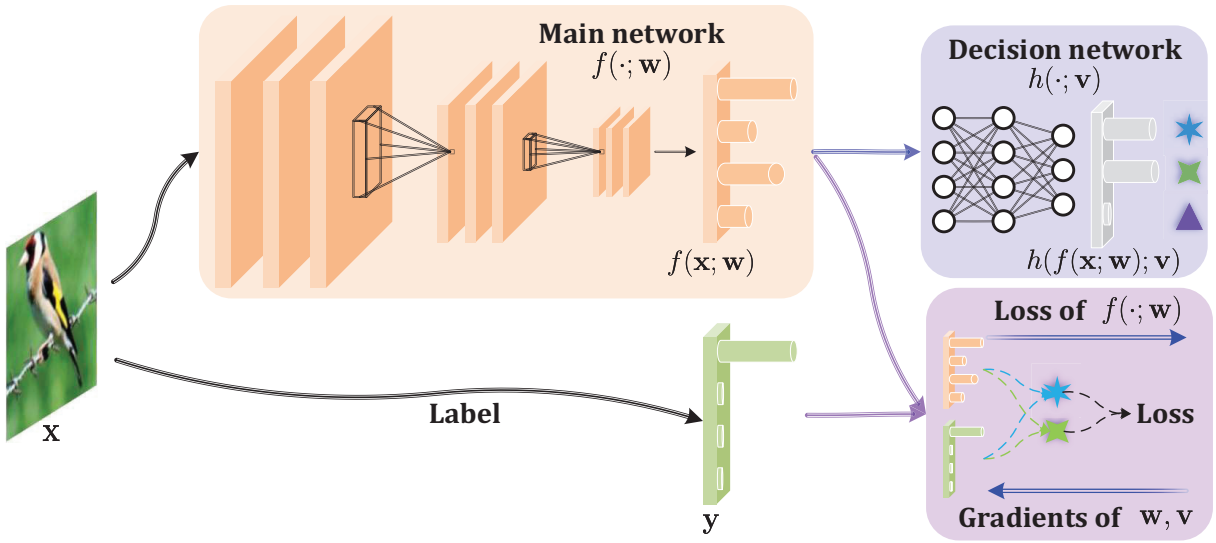

Figure 1: A conceptual visualization for our stochastic loss function. Given a main network $f(\cdot;\mathbf{w})$ and a decision network $h(\cdot;\mathbf{v})$, a group of loss functions are predefined. During the forward propagation, for the input $\mathbf{x}$ and output $\mathbf{y}$, the main network generates the estimated output $\mathbf{p} = f(\mathbf{x};\mathbf{w})$, and the loss functions can be selected by the decision network with Gumbel Softmax. Then, the loss in SLF is obtained by combing these selected loss functions. During the backward propagation, the standard back-propagation is in a position to calculate the gradients of the network weights $\mathbf{w}$ and the loss parameters $\mathbf{v}$.

Loss Function Selection

We establish the excellent Stochastic Loss Function (SLF) to dynamically and automatically generate appropriate loss functions to train deep networks in the same round of back-propagation. However, it is naturally difficult to optimize because of the nested relationship between loss parameters and network weights. To handle this issue, the SLF model parameterizes loss functions with binary codes, and devotes to jointly learning loss parameters and network weights in a differentiable way. Intuitively, the main process of SLF is illustrated in Figure 1.

Before introducing our approach, we first briefly review the loss function selection. For a specific loss function $\ell \in \mathcal{L}$, it always corresponds to a loss $\mathcal{H}(\ell_{\mathbf{v}}, \mathbf{w}) \in \mathbb{R}$ with loss parameters $\mathbf{v}$ and network weights $\mathbf{w}$. Intrinsically, the goal in the loss function selection is to find a loss function $\ell_{\mathbf{v}} \in \mathcal{L}$ that minimizes the loss $\mathcal{H}(\ell_{\mathbf{v}}, \mathbf{w}^*)$, where the network weights $\mathbf{w}^*$ are obtained by minimizing the loss $\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{H}(\ell_{\mathbf{v}}, \mathbf{w})$, *i.e.*,

$$\min_{\mathbf{v}} \ \mathcal{H}(\ell_{\mathbf{v}}, \mathbf{w}^*), \ \ s.t.\ \mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{H}(\ell_{\mathbf{v}}, \mathbf{w}). \tag{1}$$

Parameterizing loss functions

For simplicity, we denote stochastic loss function with n ordered loss functions as $\mathcal{L} = \{\ell_1, \cdots, \ell_n\}$, including mean squared error, categorical hinge, cross-entropy, and so on. With these loss functions, the stochastic loss function can be formulated as

$$\begin{aligned} &\mathcal{H}(\mathcal{L}, \mathbf{w}; \mathbf{a}, \mathbf{b}) = \sum_{\ell_i \in \mathcal{L}} a_i \cdot b_i \cdot \ell_i(f(\mathbf{x};\mathbf{w}), \mathbf{y}), \\ &s.t. \ \sum_{i=1}^{n} a_i \cdot b_i = 1, \\ &\quad a_i \in \{0,1\},\ b_i \geq 0,\ 1 \leq i \leq n, \end{aligned} \tag{2}$$

where $f(\cdot;\mathbf{w})$ denotes a network with the weights $\mathbf{w}$ that attempts to map the input $\mathbf{x}$ to the output $\mathbf{y}$, ℓ_i is the i-th loss function, $\mathbf{a} = [a_i, \cdots, a_n] \in \{0,1\}^n$ and $\mathbf{b} = [b_i, \cdots, b_n] \in \mathbb{R}^n$ signify the selective code and the weighting coefficient in the stochastic loss function $\mathcal{H}$, respectively. For the selective code, $a_i = 1$ signifies a binary weight to indicate whether ℓ_i is selected in the stochastic loss function and $a_i = 0$ otherwise. In contrast, the weighting coefficient $\mathbf{b}$ is a probability vector to generate a more precise loss function based on such selected loss functions (*i.e.*, $a_i = 1$). For a network, by such definition, a loss function can be obtained by learning the selective code $\mathbf{a}$ and the weighting coefficient $\mathbf{b}$, which implies that we can learn the code $\mathbf{a}$ to select the optimal loss from $\mathcal{L}$.

Naturally, the selective code $\mathbf{a}$ and the weighting coefficient $\mathbf{b}$ can be treated as a specific decomposition of a probability vector. However, a major distinction between the traditional probability vector is that the combining of the selective code $\mathbf{a}$ and the weighting coefficient $\mathbf{b}$ is sparse. That is, the sparsity of such combining is always smaller than $\|\mathbf{a}\|_0$, which can be readily optimized. Benefiting from such sparsity, noisy loss functions can be absolutely omitted to guarantee the stability of gradients, leading to an effective and efficient loss function.

Continuous Relaxation

For such discrete optimization problem, a straightforward solution is to use reinforcement learning. However, since the reinforcement learning is generally established as a Markov decision process, temporal-difference is employed to take structural decisions. As a result, the reward will only be observable until the loss function is selected and the network is validated for accuracy, which is always subject to delayed rewards in temporal-difference learning.

To eliminate such deficiency, we manage the problem from a differentiable angle, *i.e.*, solving the discrete optimization problem in Eq. (2) in a continuous space. For this purpose, we recast the task of selecting loss function to a softmax over all possible loss functions, *i.e.*,

$$\begin{aligned} &\mathcal{H}(\mathcal{L}, \mathbf{w}) = \sum_{\ell_i \in \mathcal{L}} B(p_i) \cdot b_i \cdot \ell_i(f(\mathbf{x}; \mathbf{w}), \mathbf{y}), \\ &s.t.\ \sum\nolimits_{i=1}^{n} B(p_i) \cdot b_i = 1,\ \sum\nolimits_{i=1}^{n} p_i = 1, \\ &\quad p_i \geq 0,\ b_i \geq 0,\ B(p_i) \in \{0, 1\},\ 1 \leq i \leq n, \end{aligned} \tag{3}$$

where $\mathbf{p} \in \mathbb{R}^n$ is a probability vector, $p_i \in \mathbb{R}$ implies the credit of selecting the i-th loss function, and the function $B(\cdot)$ indicates a binary function that can be utilized for selecting loss function discretely.

Decision Network

Inspired by curriculum learning (Bengio et al. 2009; Jiang et al. 2014), the selective code $\mathbf{a}$ and the weighting coefficient $\mathbf{b}$ are generated according to the output of the network $f(\mathbf{x}; \mathbf{w})$. That is, how to choose the loss functions is determined by the estimated output. For this purpose, a decision network $h(f(\mathbf{x}; \mathbf{w}); \mathbf{v})$ is established for guiding the selection and combination of loss functions. Formally, our SLF model can be rewritten as:

$$\begin{aligned} &\mathcal{H}(\mathcal{L}, \mathbf{w}, \mathbf{v}) = \sum_{\ell_i \in \mathcal{L}} B(p_i) \cdot b_i \cdot \ell_i(f(\mathbf{x}; \mathbf{w}), \mathbf{y}), \\ &s.t.\ \sum\nolimits_{i=1}^{n} B(p_i) \cdot b_i = 1, \\ &\quad b_i \geq 0,\ B(p_i) \in \{0, 1\},\ 1 \leq i \leq n, \\ &\quad \mathbf{p} = h(f(\mathbf{x}; \mathbf{w}); \mathbf{v}), \end{aligned} \tag{4}$$

As mentioned above, such formula implies that we have to deal with a bi-level optimization problem considering the nested relationship between loss function parameters and network weights. To handle this issue, we parameterize loss functions with binary codes, and devote to jointly learning loss parameters and network weights in a differentiable way.

Loss Sampling and Gradient Weighting

By introducing the binary function $B(\cdot)$, the discrete selective code and the weighting coefficient can be jointly obtained in a differentiable way. However, two essential problems require to be handled. First, how to design a binary function $B(\cdot)$ to yield the discrete selective code, while maintaining effective gradients. Second, the discrete selective code and the weighting coefficient for selecting loss functions and weighting gradients are deeply coupled, because of the similarity. In the following, we devote to handling such two problems by multiply sampling according to a single learnable probability vector.

Gumbel-Softmax

A natural formulation for representing discrete variable is to use the categorical distribution. However, partially due to the inability to back-propagate information through samples, it seems rarely applied in deep neural networks. In this work, we resort to the Gumbel-Max trick (Gumbel 1954) for enabling back-propagation and representing the process of taking decision as sampling from a categorical distribution, in order to perform loss sampling and gradient weighting in a principled way. Specifically, given a probability vector $\mathbf{p} = [p_1, \cdots, p_n]$ and a discrete random variable with $P(L = k) \propto p_k$, we sample from the discrete variable L by introducing the Gumbel random variables. To be more specific, we let

$$L = \arg \max_{k \in \{1, \cdots, n\}} \log p_k + G_k, \tag{5}$$

where $\{G_k\}_{k \leq n}$ is a sequence of the standard Gumbel random variables, and they are typically sampled from the Gumbel distribution $G = -\log(-\log(X))$ with $X \sim U[0, 1]$. An obstacle to directly using such approach in our problem is that the argmax operation is not really continuous. One straightforward way of dealing with this problem is to replace the argmax operation with a softmax. Formally, the Gumbel-Softmax (GS) estimator can be expressed as

$$\hat{L}_k = \frac{\exp\left(\left(\log p_k + G_k\right)/\tau\right)}{\sum_{k=1}^{n} \exp\left(\left(\log p_k + G_k\right)/\tau\right)},\ 1 \leq k \leq n, \tag{6}$$

where $\hat{L}_k$ indicates the probability that p_k is the maximal entry in $\mathbf{p}$, and τ is a temperature. When $\tau \to 0$, $[\hat{L}_1, \cdots, \hat{L}_n]$ converges to an one-hot vector, and in the other extreme it will become a discrete uniform distribution with $\tau \to +\infty$.

From the formulation in Eq. (6), we see that the Gumbel-Softmax estimator pertains solely to deal with the problems that only one category requires to be determined, *i.e.*, the outputs are one-hot vectors not any binary code. In our stochastic loss function, however, optimal gradients may be combined from multiple loss functions. One direct way of managing such issue is to map all possible operation combinations to a discrete space. Unfortunately, combination modes between limited loss functions are uncountable, which indicates that they are not mapped to a discrete space.

Multiple Gumbel-Softmax

For an effective and efficient way to yield appropriate gradients, the Gumbel-Softmax is utilized repeatedly to select various loss functions and weight gradients jointly, which is inspired by the following perspective. That is, the output of Gumbel-Softmax depends on the elements in the probability vector $\mathbf{p} = [p_1, \cdots, p_n]$. Denoting the Gumbel-Softmax as a function $\mathcal{G}(\cdot)$, it always outputs an one-hot vector $\bar{\mathbf{p}}_i$, *i.e.*,

$$\bar{\mathbf{p}}_i = \mathcal{G}(\mathbf{p}), \tag{7}$$

Algorithm 1 Stochastic Loss Function

Require: Dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}$, loss functions $\mathcal{L} = \{\ell_i\}_{i=1}^{n}$, networks $f(\cdot;\mathbf{w})$ and $h(\cdot;\mathbf{v})$, sampling times K.
Ensure: Trained main network $f(\cdot;\mathbf{w}^*)$.

```
Randomly initialize parameters w in the main network f(·;w) and v in the decision network h(·;v);
for number of training iterations do
    for (x, y) ∈ D do
        p = f(x;w);                                  ▷ the estimated output of the main network with the parameter w
        for each time step k = 1, 2, ··· , K do
            p̂_k = G(h(p;v));                         ▷ selecting loss functions with the decision network and Gumbel Softmax
        end for
        p̂ = (1/K) Σ_{k=1}^{K} p̂_k;                   ▷ weighting and combing the loss functions according to Eq. (10)
        H(L, w, v) = Σ_{ℓ_i ∈ L} p̂_i · ℓ_i(f(x;w), y);   ▷ computing the loss of our SLF according to Eq. (11)
        Update w and v by minimizing Eq. (11);       ▷ updating the parameters w and v with the standard back-propagation
    end for
end for
```

where $\bar{\mathbf{p}}_i$ indicates that the i-th element in $\bar{\mathbf{p}}_i$ is equal to 1, and the others are 0. Specifically, we have

$$P(\mathcal{G}(\mathbf{p}) = \bar{\mathbf{p}}_i) = p_i, \tag{8}$$

where signifies that the output of the Gumbel-Softmax $\mathcal{G}(\cdot)$ is not deterministic, but variable. Benefiting from this variability, the selective code and the weighting coefficient can be jointly obtained.

Given a probability vector $\mathbf{p}$, specifically, the selective code and the weighting coefficient are simultaneously generated by multiply sampling with Gumbel-Softmax. According to the Gumbel-Softmax function $\mathcal{G}(\cdot)$, a sequence of one-hot vectors can be obtained as follows

$$\{\hat{\mathbf{p}}_1, \cdots, \hat{\mathbf{p}}_K\} = Repeat\{\mathcal{G}(\mathbf{p}), K\}, \tag{9}$$

where $Repeat\{\mathcal{G}(\mathbf{p}), K\}$ means that the Gumbel-Softmax function $\mathcal{G}(\mathbf{p})$ repeatedly performs K times for the number of K one-hot vectors $\{\hat{\mathbf{p}}_1, \cdots, \hat{\mathbf{p}}_K\}$. By combing these one-hot vectors, we have

$$\hat{\mathbf{p}} = \frac{1}{K}\sum_{i=1}^{K}\hat{\mathbf{p}}_i,\ \hat{\mathbf{p}} \in \mathbb{R}^n \tag{10}$$

where $\hat{p}_i$ is the i-th element in $\hat{\mathbf{p}}$, and can be treated as an approximation of $B(p_i)\cdot b_i$ in Eq. (3). That is, the stochastic loss function can be rewritten as

$$\begin{aligned} &\mathcal{H}(\mathcal{L}, \mathbf{w}, \mathbf{v}) = \sum_{\ell_i \in \mathcal{L}} \hat{p}_i \cdot \ell_i(f(\mathbf{x};\mathbf{w}), \mathbf{y}), \\ &s.t.\ \hat{\mathbf{p}} = \frac{1}{K}\sum\nolimits_{i=1}^{K}\hat{\mathbf{p}}_i, \\ &\quad \{\hat{\mathbf{p}}_1, \cdots, \hat{\mathbf{p}}_K\} = Repeat\{\mathcal{G}(\mathbf{p}), K\}, \\ &\quad \mathbf{p} = h(f(\mathbf{x};\mathbf{w});\mathbf{v}),\ \mathbf{p} \in \mathbb{R}^n. \end{aligned} \tag{11}$$

Intrinsically, such formulation in Eq. (11) can be treated as a general loss function. For example, only a single loss function will be selected if $K = 1$ is met. When $1 < K < n$, multiple loss functions may be combined for generating more superior and precise gradients. Extremely, it will gradually tends to be a softmax, as K increases to infinity.

In summary, we illustrate the stochastic loss function in Algorithm 1. Given a main network $f(\cdot;\mathbf{w})$ and a decision network $h(\cdot;\mathbf{v})$, a group of loss functions are predefined. During the forward propagation, for the input $\mathbf{x}$ and output $\mathbf{y}$, the main network generates the estimated output $\mathbf{p} = f(\mathbf{x};\mathbf{w})$, and the loss functions can be selected by the decision network with the multiple Gumbel Softmax, *i.e.*, as described in Eq. (9) and Eq. (10). Then, the SLF loss in Eq. (11) is obtained by combing these selected loss functions. During the backward propagation, the standard back-propagation is in a position to calculate the gradients of the network weights $\mathbf{w}$ and the loss parameters $\mathbf{v}$. For minimizing the SLF loss in Eq. (11), the optimization algorithm (*e.g.*, SGD and ADAM) can be employed to adjust these weights/parameters simultaneously. Conclusively, the main network $f(\cdot;\mathbf{w})$ is trained well to fit the samples in $\mathcal{D}$.

Experiments

In this section, we systematically carry out extensive experiments to verify the capability of our SLF on various pattern recognition and machine learning tasks, including image classification, image clustering, neural machine translation, regressive on non-Euclidean structured signals, and objection detection. Furthermore, extensive ablation experiments are conducted to systematically and comprehensively analyze the proposed stochastic loss function.

Experimental Setting

The hyper-parameters in our experiments are set as follows. In each experiment, our SLF model always inherits the same setting in the compared baslines, including network architectures (*e.g.*, activation functions, initilizations, bach sizes, *etc.*), learning rates, and the optimizors, except of the loss functions. For comparison, several popular loss functions are utilized, such as the cross entropy loss (de Boer et al. 2005), the large-margin softmax loss (Liu et al. 2016), the smooth 0-1 loss function (Nguyen and Sanner 2013), the adaptive robust loss function (Barron 2019), and the L2T-DLF loss function in (Wu et al. 2018). Note that there are tiny differents for each specific task, which will be mentioned in each experiment. For a reasonable evaluation, we carry out

Table 1: The classification results on MNIST. The best results are highlighted in **bold**.

Dataset	Model	C-E	Smooth	L-M Softmax	L2T-DLF	ARLF	SLF
MNIST	MLP	0.9806	0.9811	0.9817	0.9831	0.9814	**0.9852**
MNIST	LeNet	0.9902	0.9906	0.9912	0.9923	0.9906	**0.9935**
CIFAR-10	ResNet-8	0.8755	0.8792	0.8866	0.8918	0.8646	**0.8983**
CIFAR-10	ResNet-20	0.9125	0.9147	0.9198	0.9237	0.9153	**0.9302**
CIFAR-10	ResNet-32	0.9249	0.9258	0.9299	0.9305	0.9224	**0.9364**
CIFAR-10	WRN	0.962	0.9619	0.9631	0.9658	0.9593	**0.9714**
CIFAR-10	DenseNet	0.9646	0.9652	0.9663	0.9692	0.9629	**0.9763**
CIFAR-100	ResNet-8	0.6021	0.6048	0.6107	0.6173	0.6024	**0.6265**
CIFAR-100	ResNet-20	0.6767	0.6799	0.6835	0.6903	0.6775	**0.6992**
CIFAR-100	ResNet-32	0.6962	0.6988	0.7044	0.7075	0.6953	**0.7145**

10 random restarts and the average results are used for comprehensive comparisons.

Classification

As an essential machine learning problem, classification is an indispensable experiment for the stochastic loss function. To validate its capability, we carry out a series of image classification tasks on three frequently-used datasets, including MNIST, CIFAR-10, and CIFAR-100. For each dataset, several popular deep neural networks are employed to demonstrate the capability. Specifically, the classification task can be treated as an optimization problem to minimize the following formulate

$$\min_{\mathbf{w}} \mathbf{E}(\mathbf{w}) = \sum_{i} L(f(\mathbf{x}_i; \mathbf{w}), \mathbf{y}_i).$$

where $L(f(\mathbf{x}_i; \mathbf{w}), \mathbf{y}_i)$ is a loss function to measure the loss between the observable label $\mathbf{y}_i$ and the estimated label $f(\mathbf{x}_i; \mathbf{w})$, and $\mathbf{w}$ signifies the model parameters in the function g. In this experiment, the candidate loss functions in $\mathcal{L}$ includes mean squared loss, mean absolute loss, hinge loss, cosine proximity loss, and binary cross-entropy loss.

In Table 1, we report the quantitative results of the diverse loss functions on the MNIST, CIFAR-10, and CIFAR-100 datasets. From the table, the results assuredly prove the capability of our SLF, which can yield better gradients and improve the baselines with a larger margin consistently. Further analysis, several tendencies can be observed from Table 1. First, the superiority of each loss function is always consistently exist for every dataset and deep model. This means that how to choose a loss function is significant for a better model, and the impact is uncertainty, but deterministic. This also shows the necessity for learning a better loss function. Second, the advantage of the learnable loss functions are not necessarily better than the handcrafted loss functions, *e.g.*, L-M softmax is always better than the ARLF. It indicates that the valuable prior knowledge will effectively elevate the performance of loss functions. Third, the promotions of SLF on each datasets are stable, *e.g.*, the classification performance of ResNet (He et al. 2016) will be steadily improved as the depth increases, which implies that SLF has outstanding capability to generate appropriate gradients for various deep networks, not merely limited to tiny networks (*e.g.*, MLP and LeNet on MNIST).

Clustering

To verify the performance of our stochastic loss function to manage the deep unsupervised learning, we perform the clustering tasks on the CIFAR-10 (Krizhevsky and Hinton 2009), ImageNet-10 (Chang et al. 2017), and ImageNet-Dog (Chang et al. 2017) datasets. These datasets are popular datasets utilized in the clustering tasks. There are a collections of 60000, 13000, and 19500 color images with the sizes of $32 \times 32 \times 3$, $96 \times 96 \times 3$, and $96 \times 96 \times 3$. In this experiment, we follow the previous work DAC in (Chang et al. 2017) to perform the clustering tasks on these datasets. Formally, DAC recasts the clustering problem as a binary pairwise-classification problem, *i.e.*,

$$\min_{\mathbf{w}} \mathbf{E}(\mathbf{w}) = \sum_{i,j} L(r_{ij}, f(\mathbf{x}_i, \mathbf{x}_j; \mathbf{w})).$$

By iteratively choosing labeled pairwise-samples and training the network $f(\cdot, \cdot; \mathbf{w})$ with the label features (Chang et al. 2017), each sample can learn an one-hot representation for clustering. Contrary to the binary cross-entropy loss in (Chang et al. 2017), the stochastic loss function is utilized to achieve better clustering performance, and the candidate loss functions in $\mathcal{L}$ includes mean squared loss, mean absolute loss, hinge loss, cosine proximity loss, and binary cross-entropy loss. For the results, Adjusted Rand Index (ARI), Normalized Mutual Information (NMI) and Accuracy (ACC) are utilized for evaluating, which range in $[0, 1]$ and higher scores mean more precious results.

The results of the compared methods are illustrated in Table 2. From the table, we note that our stochastic loss function can dramatically improve the performance of DAC with significant margins on every datasets. It strongly demonstrates that SLF is capable of yielding more precise gradients for adjusting weights in deep networks for clustering. Furthermore, contrary to the classification problem, we observe that higher improvements are obtained in clustering. A possible reason is that DAC is sensitive to gradients during learning. That is, biased gradients always result in choosing the wrong pairwise-samples. Then, these samples will train the deep models in a wrong direction. As a result, the clustering performance is degraded.

Regressive on non-Euclidean structured signals

In contrast to the classification and regressive problems, the regressive task attempts to map a high-dimensional rep-

Table 2: The clustering results of the methods on the experimental datasets. The best results are highlighted in **bold**.

Dataset	CIFAR-10	CIFAR-10	CIFAR-10	ImageNet-10	ImageNet-10	ImageNet-10	ImageNet-Dog	ImageNet-Dog	ImageNet-Dog
Metric	NMI	ARI	ACC	NMI	ARI	ACC	NMI	ARI	ACC
AE (Bengio et al. 2006)	0.2393	0.1689	0.3135	0.2099	0.1516	0.3170	0.1039	0.0728	0.1851
SAE (Bengio et al. 2006)	0.2468	0.1555	0.2973	0.2122	0.1740	0.3254	0.1129	0.0729	0.1830
DAE (Vincent et al. 2010)	0.2506	0.1627	0.2971	0.2064	0.1376	0.3044	0.1043	0.0779	0.1903
DeCNN (Zeiler et al. 2010)	0.2395	0.1736	0.2820	0.1856	0.1421	0.3130	0.0983	0.0732	0.1747
SWWAE (Zhao et al. 2015)	0.2330	0.1638	0.2840	0.1761	0.1603	0.3238	0.0936	0.0760	0.1585
CatGAN (Springenberg 2015)	0.2646	0.1757	0.3152	0.2250	0.1571	0.3459	0.1213	0.0776	0.1738
GMVAE (Dilokthanakul et al. 2016)	0.2451	0.1674	0.2908	0.1934	0.1683	0.3344	0.1074	0.0786	0.1788
JULE-SF (Yang, Parikh, and Batra 2016)	0.1919	0.1357	0.2643	0.1597	0.1205	0.2927	0.1213	0.0776	0.1738
JULE-RC (Yang, Parikh, and Batra 2016)	0.1923	0.1377	0.2715	0.1752	0.1382	0.3004	0.0492	0.0261	0.1115
DEC (Xie, Girshick, and Farhadi 2016)	0.2568	0.1607	0.3010	0.2819	0.2031	0.3809	0.1216	0.0788	0.1949
DAC+softmax	0.4065	0.3135	0.5013	0.4253	0.3225	0.5122	0.2253	0.1005	0.2676
DAC+C-E (Chang et al. 2017)	0.3959	0.3059	0.5218	0.3944	0.3019	0.5272	0.2185	0.1105	0.2748
DAC+curriculum learning	0.3793	0.2802	0.4982	0.3693	0.2837	0.5026	0.1815	0.0953	0.2455
DAC+Smooth	0.4349	0.3274	0.4538	0.5742	0.5072	0.6681	0.2174	0.1177	0.2518
DAC+ARLF	0.3987	0.2744	0.4607	0.5121	0.5186	0.6460	0.2316	0.1144	0.1821
DAC+L1	0.4186	0.3367	0.4584	0.5636	0.5121	0.6577	0.2334	0.1157	0.2461
DAC+L2	0.4379	0.3399	0.4778	0.5827	0.5218	0.6737	0.2362	0.1241	0.2644
DAC+L4	0.4062	0.3015	0.4516	0.5813	0.4878	0.6363	0.2091	0.0938	0.2347
DAC+**SLF**	**0.4454**	**0.3457**	**0.5400**	**0.6524**	**0.5758**	**0.7345**	**0.2734**	**0.1571**	**0.3156**

Table 3: Squared correlations on DPP4. The best results are highlighted in **bold**.

Loss / Model	C-E	RL	AC	Softmax-M	L2T-DLF	SLF
LCNs	0.2250	0.2222	0.2106	0.2260	0.2290	**0.2429**
DFNs	0.2140	0.2085	0.2067	0.2228	0.2188	**0.2334**
ECC	0.2490	0.2394	0.2370	0.2475	0.2523	**0.2620**
SCNs	0.2480	0.2384	0.2459	0.2566	0.2482	**0.2588**
GCNs	0.2580	0.2483	0.2443	0.2508	0.2627	**0.2671**
ChebNets	0.2650	0.2634	0.2587	0.2703	0.2692	**0.2793**

resentation into a scalar. For a comprehensively analysis, in this experiment, the stochastic loss function is utilized to deal with the regressive tasks on non-Euclidean structured signals. Specifically, graph networks are modeled to predict the molecular activity on the DPP4 dataset. In the DPP4 dataset, there are 8193 molecules with 2796 features severally. In this experiment, both spatial and spectral graph networks are taken as baselines (Chang et al. 2018; Zhang et al. 2018), including the local connected networks (LCNs) (Bruna et al. 2013), the dynamic filters based networks (DFNs) (Verma, Boyer, and Verbeek 2017), the edge-conditioned convolution (ECC) (Simonovsky and Komodakis 2017), the spectral networks (SCNs) (Henaff, Bruna, and LeCun 2015), the Chebyshev based SCNs (ChebNets) (Defferrard, Bresson, and Vandergheynst 2016), and the graph convolution networks (GCNs) (Kipf and Welling 2016).

The results of the compared methods are illustrated in Table 3. Inherently, the results mainly depend on two aspects, *i.e.*, model and loss. Specifically, the former determines the upper limit of the model, and the latter affects whether the model can reach the upper limit. From the Table 3, our SLF achieves the best performance for all the employed models. It verifies that SLF enables to generate more appropriate gradients to improve the performance of models. That is, our SLF is capable of managing the regressive task and the tasks defined on non-Euclidean domains.

Neural Machine Translation

The task of neural machine translation is employed to validate the effectiveness of SLF on deep recurrent neural networks. Following the previous work (Wu et al. 2018), the IWSLT-14 dataset is utilized to evaluate the translation tasks between German and English. For a fair comprehensive, a single layer LSTM model (*i.e.*, LSTM-1) and a deeper translation model stacking two LSTM layers (*i.e.*, LSTM-2) are employed as baselines. Furthermore, as a popular self-attention mechanism model that achieves superior performance on many neural machine translation tasks, the Transformer architecture (Lin et al. 2017) is utilized as a typical baseline. For the loss functions, five frequently-used losses are used in this experiments, including the maximum likelihood estimation loss (C-E), the reinforcement learning (RL) loss (Ranzato et al. 2016), the loss specified via actor-critic (Bahdanau et al. 2017), the softmax-margin loss, and the loss learned with L2T-DLF (Wu et al. 2018).

Table 4: The BLEU scores on IWSLT-14 German-English task. The best results are highlighted in **bold**.

Loss / Model	C-E	RL	AC	Softmax-M	L2T-DLF	SLF
LSTM-1	27.28	27.53	27.75	28.12	29.52	**31.02**
LSTM-2	30.86	31.03	31.21	31.22	31.75	**32.14**
Transformer	34.01	34.32	34.34	34.46	34.80	**35.53**

We report the experimental results in Table 4. Note that the results in the table clearly demonstrate that SLF is in a position to guide the utilized RNNs to achieve the superior performance, contrary to the compared loss functions. It is worth mentioning that almost 1 point is improved compared with the much stronger Transformer model, which strongly verifies the effectiveness of our SLF model.

Objection Detection

As a high-level image processing problem, objection detection consists of two basic tasks, *i.e.*, a regressive task of locating objects and a classification task of classifying the located objects. Because of the high efficiency and the end-to-end learning, the YOLO-v1 model (Redmon et al. 2016) is utilized as a basic model in this experiment for verifying the effectiveness of our SLF. Compared with the loss defined in traditional classification tasks, the loss in the YOLO model is a multi-part loss function, *i.e.*,

$$\lambda_{coord}\ell_{coord} + \lambda_{pred}\ell_{pred} + \lambda_{obj}\ell_{obj} + \lambda_{noobj}\ell_{noobj},$$

Table 5: Real-Time systems on PASCAL VOC 2007. The bold subscripts mean the performance improvements.

Real-Time Detectors			
Detectors	Train	mAP	FPS
100Hz DPM	2007	16.0	100
30Hz DPM	2007	26.1	30
Fast YOLO	2007+2012	52.7	155
YOLO	2007+2012	63.4	45
Fast YOLO+SLF	2007+2012	$55.1_{\mathbf{+2.4}}$	155
YOLO+SLF	2007+2012	$65.4_{\mathbf{+2.1}}$	45
Less Than Real-Time Detectors			
Detectors	Train	mAP	FPS
Fastest DPM	2007	30.4	15
R-CNN Minus R	2007	53.5	6
Fast R-CNN	2007+2012	70.0	0.5
Faster R-CNN VGG	2007+2012	73.2	7
Faster R-CNN ZF	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21
YOLO VGG-16+SLF	2007+2012	$69.6_{\mathbf{+3.2}}$	21

where ℓ_{coord} and ℓ_{pred} indicate the loss to predict coordinates and classes, ℓ_{obj} and ℓ_{noobj} measure the confidences of boxes, four hyper-parameters $\{\lambda_{coord}, \lambda_{pred}, \lambda_{obj}, \lambda_{noobj}\}$ can be always balanced the weights of these losses. In the work (Redmon et al. 2016), $\{\lambda_{coord} = 5, \lambda_{pred} = 1, \lambda_{obj} = 1, \lambda_{noobj} = 0.5\}$ is set.

In this experiment, we employ SLF to learn these hyper-parameters for a better detector. From the Table 5, the results signify that SLF enables to automatically adjust hyper-parameters for a better performance, contrary to YOLO and Faster RCNN (Ren et al. 2017). Quantificationally, SLF has at least $+2\%$ improvement with each baseline, which shows that SLF possesses the prominent superiority on more complex tasks (*e.g.*, detection), not just toy tasks (*e.g.*, classification). In addition, this experiment also shows that SLF provides a possible way for automatically adjusting hyper-parameters during training deep networks.

Ablation study

In this subsection, we perform extensive ablation studies on diverse datasets to synthetically analyze our SLF. Intuitively, all the results are illustrated in Figure 2.

Contribution of Gumbel-Softmax In order to empirically demonstrate the contribution of the Gumbel-Softmax technical, SLF-Softmax that utilizes softmax to balance the weights of different loss functions is designed for comparison. Different from the Gumbel-Softmax, technically, Softmax always selects all the loss functions to generate gradients. From Table 2 and Figure 2 (a), SLF with Gumbel-Softmax can achieve better performance than SLF-Softmax. The main reason is some noisy loss functions may be selected in SLF-Softmax, which generates noisy gradients. In contrast, SLF suffices to filter noisy loss functions, and achieve high robustness and appropriate gradients.

Impact of sampling times K We carry out experiments on the MNIST and CIFAR-10 datasets to analyze the sensitivities to the number of sampling time K. The softmax is utilized as a baseline for comparison. Figure 2 (a) indicates

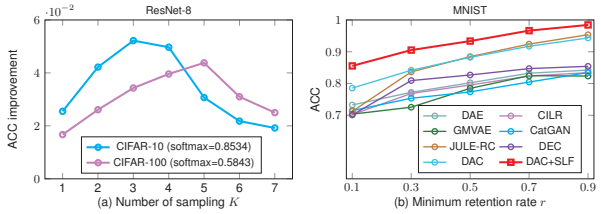

Figure 2: Ablation studies for our stochastic loss function.

that the optimal sampling time K is varied with different datasets. Specifically, on MNIST and CIFAR-10, the best sampling times are 3 and 4, relying on the same group of loss functions. Such results are expected, *i.e.*, small number of loss function may guide the model to select more robust but necessary losses. High sampling times may introduce some noisy losses and degrade the performance conclusively.

Performance on Imbalanced Datasets To further study the performance of SLF on imbalanced datasets, we establish a series of datasets from MNIST to execute an additional clustering experiment. Following the previous work (Xie, Girshick, and Farhadi 2016), a series of subsets can be randomly sampled from MNIST with different minimum retention rates. For a minimum retention rate r, images of class 0 will be hold with probability r and class 9 with probability 1, with the other classes linearly between 0 and 9. From the Figure 2 (b), we observe that our DAC+SLF model is more robust than the other clustering methods. Especially, we improve the clustering accuracy more than 5% point, with SLF only. A main reason is that Gumbel-Softmax suffices to adaptively select better loss functions for appropriate gradients and achieves better performance conclusively.

Conclusion

We present a conceptually simple yet powerful stochastic loss function to achieve more appropriate gradients for training deep models. Technically, a group of loss functions are predefined according to the human knowledge. By generalizing the Gumbel-Softmax, we develop multiple Gumbel-Softmax, which is in a position to select and combine these loss functions to generate more appropriate gradients. Extensive experiments on popular datasets strongly evidence that our stochastic loss function outperforms current losses on various tasks, including classification, clustering, regression, translation, and object detection. It will be interesting in the future to automatically learn these loss functions, beyond the predefined ones, to reduce the perceived intervention and improve the generality of the loss function.

References

Agarwal, R.; Liang, C.; Schuurmans, D.; and Norouzi, M. 2019. Learning to generalize from sparse and underspecified rewards. In *ICML*.

Bahdanau, D.; Brakel, P.; Xu, K.; Goyal, A.; Lowe, R.; Pineau, J.; Courville, A. C.; and Bengio, Y. 2017. An actor-critic algorithm for sequence prediction. In *ICLR*.

Barron, J. T. 2019. A general and adaptive robust loss function. In *CVPR*.

Bengio, Y.; Lamblin, P.; Popovici, D.; and Larochelle, H. 2006. Greedy layer-wise training of deep networks. In *NIPS*.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *ICML*.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2013. Spectral networks and locally connected networks on graphs. *CoRR* abs/1312.6203.

Chang, J.; Wang, L.; Meng, G.; Xiang, S.; and Pan, C. 2017. Deep adaptive image clustering. In *ICCV*.

Chang, J.; Gu, J.; Wang, L.; Meng, G.; Xiang, S.; and Pan, C. 2018. Structure-aware convolutional neural networks. In *NIPS*.

Chang, J.; Meng, G.; Wang, L.; Xiang, S.; and Pan, C. 2019. Deep self-evolution clustering. *T-PAMI*.

Chebotar, Y.; Molchanov, A.; Bechtle, S.; Righetti, L.; Meier, F.; and Sukhatme, G. S. 2019. Meta-learning via learned loss. *CoRR* abs/1906.05374.

de Boer, P.; Kroese, D. P.; Mannor, S.; and Rubinstein, R. Y. 2005. A tutorial on the cross-entropy method. *Annals OR*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*.

Dilokthanakul, N.; Mediano, P.; Garnelo, M.; Lee, M.; Salimbeni, H.; Arulkumaran, K.; and Shanahan, M. 2016. Deep unsupervised clustering with gaussian mixture variational autoencoders.

Grabocka, J.; Scholz, R.; and Schmidt-Thieme, L. 2019. Learning surrogate losses. *CoRR* abs/1905.10108.

Gumbel, E. J. 1954. *Statistical theory of extreme values and some practical applications: a series of lectures*. Number 33. US Govt. Print. Office.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*.

Henaff, M.; Bruna, J.; and LeCun, Y. 2015. Deep convolutional networks on graph-structured data. *CoRR* abs/1506.05163.

Huang, C.; Zhai, S.; Talbott, W.; Martin, M. B.; Sun, S.; Guestrin, C.; and Susskind, J. 2019. Addressing the loss-metric mismatch with adaptive loss alignment. In *ICML*.

Jiang, L.; Meng, D.; Yu, S.; Lan, Z.; Shan, S.; and Hauptmann, A. G. 2014. Self-paced learning with diversity. In *NIPS*.

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *CoRR* abs/1609.02907.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images. *Master's Thesis, Department of Computer Science, University of Torono*.

Li, C.; Lin, C.; Guo, M.; Wu, W.; Ouyang, W.; and Yan, J. 2019. AM-LFS: automl for loss function search. *CoRR* abs/1905.07375.

Lin, Z.; Feng, M.; dos Santos, C. N.; Yu, M.; Xiang, B.; Zhou, B.; and Bengio, Y. 2017. A structured self-attentive sentence embedding. In *ICLR*.

Liu, W.; Wen, Y.; Yu, Z.; and Yang, M. 2016. Large-margin softmax loss for convolutional neural networks. In *ICML*.

Narasimhan, H. 2018. Learning with complex loss functions and constraints. In *AISTATS*.

Nguyen, T., and Sanner, S. 2013. Algorithms for direct 0–1 loss optimization in binary classification. In *ICML*.

Ranzato, M.; Chopra, S.; Auli, M.; and Zaremba, W. 2016. Sequence level training with recurrent neural networks. In *ICLR*.

Redmon, J.; Divvala, S. K.; Girshick, R. B.; and Farhadi, A. 2016. You only look once: Unified, real-time object detection. In *CVPR*.

Ren, S.; He, K.; Girshick, R. B.; and Sun, J. 2017. Faster R-CNN: towards real-time object detection with region proposal networks. *T-PAMI*.

Shu, J.; Xie, Q.; Yi, L.; Zhao, Q.; Zhou, S.; Xu, Z.; and Meng, D. 2019. Meta-weight-net: Learning an explicit mapping for sample weighting. *arXiv preprint arXiv:1902.07379*.

Simonovsky, M., and Komodakis, N. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *CVPR*.

Springenberg, J. 2015. Unsupervised and semi-supervised learning with categorical generative adversarial networks.

Streeter, M. 2019. Learning effective loss functions efficiently. *CoRR* abs/1907.00103.

Verma, N.; Boyer, E.; and Verbeek, J. 2017. Dynamic filters in graph convolutional networks. *CoRR* abs/1706.05206.

Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; and Manzagol, P. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*.

Wu, L.; Tian, F.; Xia, Y.; Fan, Y.; Qin, T.; Lai, J.; and Liu, T. 2018. Learning to teach with dynamic loss functions. In *NeurIPS*.

Xie, J.; Girshick, R.; and Farhadi, A. 2016. Unsupervised deep embedding for clustering analysis. In *ICML*.

Yang, J.; Parikh, D.; and Batra, D. 2016. Joint unsupervised learning of deep representations and image clusters. In *CVPR*.

Zeiler, M.; Krishnan, D.; Taylor, G.; and Fergus, R. 2010. Deconvolutional networks. In *CVPR*.

Zhang, Q.; Jin, Q.; Chang, J.; Xiang, S.; and Pan, C. 2018. Kernel-weighted graph convolutional network: A deep learning approach for traffic forecasting. In *ICPR*.

Zhao, J.; Mathieu, M.; Goroshin, R.; and LeCun, Y. 2015. Stacked what-where auto-encoders.